University of California

Los Angeles

Cross-Layer Approaches for Monitoring, Margining and Mitigation of Circuit Variability

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical Engineering

by

Liangzhen Lai

2015

Abstract of the Dissertation

Cross-Layer Approaches for Monitoring, Margining and Mitigation of Circuit Variability

by

Liangzhen Lai

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2015

Professor Puneet Gupta, Chair

With technology scaling, circuit performance has become more sensitive to various sources of variability, including manufacturing variations, ambient fluctuations, and circuit wear-out. These increased variations have created new challenges for conventional hardware guard-banding, as the additional design margin diminishes the benefits of technology scaling. This dissertation aims at reducing total system design margin with cross-layer approaches on monitoring, margining and mitigation of circuit variability.

Since hardware and software adaptation can be used to reduce design margin with the exposed hardware variability provided by hardware monitors, we start by proposing two different types of performance monitors that can achieve better monitoring accuracy and smaller monitoring overhead. We also demonstrate the use of these performance monitors in system adaptation with our end-to-end implementation of software testbeds.

We also study the dynamic variations and reliability margining problem in presence of monitor-and-actuate adaptation and emerging system contexts. In a system with monitor-and-actuate adaptation, dynamic variations require extra margin for monitor and actuate latencies. We analyze and study the margining problem considering different choices of the monitor and actuator types. System reliability margining strategies are also proposed for circuits in the “dark silicon” era, where the low-level design margin should consider the contexts of high-level power/thermal constraints.

Last, we propose a clock gating methodology to mitigate the aging induced clock skew, which is difficult to monitor and resolve through adaptation. For certain phenomena and variation sources, for example, soft error rates at different location/altitude, we also propose system/cloud-based monitors. An emulation platform is built to study the impacts of dynamic power management schemes on system reliability.

The dissertation of Liangzhen Lai is approved.

Milos Ercegovac

Sudhakar Pamarti

Mani Srivastava

Puneet Gupta, Committee Chair

University of California, Los Angeles

2015

To mine.

Table of Contents

List of Figures

List of Tables

Vita

2010	Bachelor of Engineering (Honors Research Option), Electronic Engineering, Hong Kong University of Science and Technology.
2012	Master of Science, Electrical Engineering, UCLA.

Publications

T. B. Chan, P. Gupta, A. Kahng, and L. Lai, "DDRO: A Novel Performance Monitoring Methodology Based on Design-Dependent Ring Oscillators," in *IEEE International Symposium on Quality Electronic Design*, Mar. 2012.

L. Lai, V. Chandra, R. Aitken, and P. Gupta, "SlackProbe: A low overhead in situ online timing slack monitoring methodology," in *IEEE/ACM Design, Automation and Test in Europe*, Mar. 2013.

L. Wanner, S. Elmalaki, L. Lai, P. Gupta, and M. Srivastava, "VarEMU: An Emulation Testbed for Variability-Aware Software," in *ACM International Conference on Hardware/Software Codesign and System Synthesis*, Oct. 2013.

L. Lai and P. Gupta, "Accurate and inexpensive performance monitoring for variability-aware systems," in *Proc. Asia and South Pacific Design Automation Conference*, Jan. 2014.

L. Lai, V. Chandra, P. Gupta "Evaluating and Exploiting Impacts of Dynamic Power Management Schemes on System Reliability" in *International Conference on Compilers, Archi-*

tecture and Synthesis for Embedded Systems, Oct. 2015

G. Leung, L. Lai, P. Gupta, and C. O. Chui, "Device- and Circuit-Level Variability Caused by Line Edge Roughness for Sub-32nm Finfet Technologies," *Electron Devices, IEEE Transactions on*, vol.59, no.8, pp.2057,2063, Aug. 2012

T.-B. Chan, P. Gupta, A. B. Kahng, and L. Lai, "Synthesis and Analysis of Design-Dependent Ring Oscillator (DDRO) Performance Monitors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol.22, no.10, pp.2117,2130, Oct. 2014

L. Lai, V. Chandra, R. Aitken, and P. Gupta, "BTI-Gater: An Aging-Resilient Clock Gating Methodology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, pp. 180-189, Jun. 2014.

L. Lai, V. Chandra, R. Aitken, and P. Gupta, "SlackProbe: A Flexible and Efficient In Situ Timing Slack Monitoring Methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, pp. 1168-1179, Aug. 2014.

L. Wanner, L. Lai, A. Rahimi, M. Gottscho, P. Mercati, C.-H. Huang, F. Sala, Y. Agarwal, L. Dolecek, N. Dutt, P. Gupta, R. Gupta, R. Jhala, R. Kumar, S. Lerner, M. Subhasish, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, D. Sylvester, and Y. Zhou, "NSF Expedition on Variability-Aware Software: Recent Results and Contributions," *De Gruyter Information Technology (it)*, vol. 57, pp. 181-198, June 2015.

CHAPTER 1

Introduction

Semiconductor manufacturing process scaling has been the primary driver for innovations and performance/power improvements in computer systems. With technology scaling to nano-scale range, hardware variability continues to increase due to increasing amounts of manufacturing variability, ambient fluctuations and circuit wear-out (e.g., N/PBTI, HCI etc.) degradations. Considering the power and performance variability as shown in Fig. 1.1, these increased variations have led to increased margin in designing reliable integrated systems, which takes up significant amounts of system's performance/power and diminishes the potential benefits of scaling.

This dissertation aims at reducing total system design margin, including efforts in developing methodologies for monitoring, margining and mitigation of circuit variability. These methodologies are mainly cross-layer approaches that can span multiple layers of system abstraction. We focus on cross-layer approaches due to the following reasons:

- System adaptation can be made more efficient with cross-layer approaches. By capturing and exposing hardware variability to the software/system level, corresponding hardware and software adaptation can opportunistically reduce or even eliminate the design margin [Gup12].

- System design margin is determined by the accuracy and efficiency of the entire monitor-and-actuate adaptation process. Over-optimizing a single component without considering the full-system may not improve the system margin, as components from other layers may be the bottleneck.

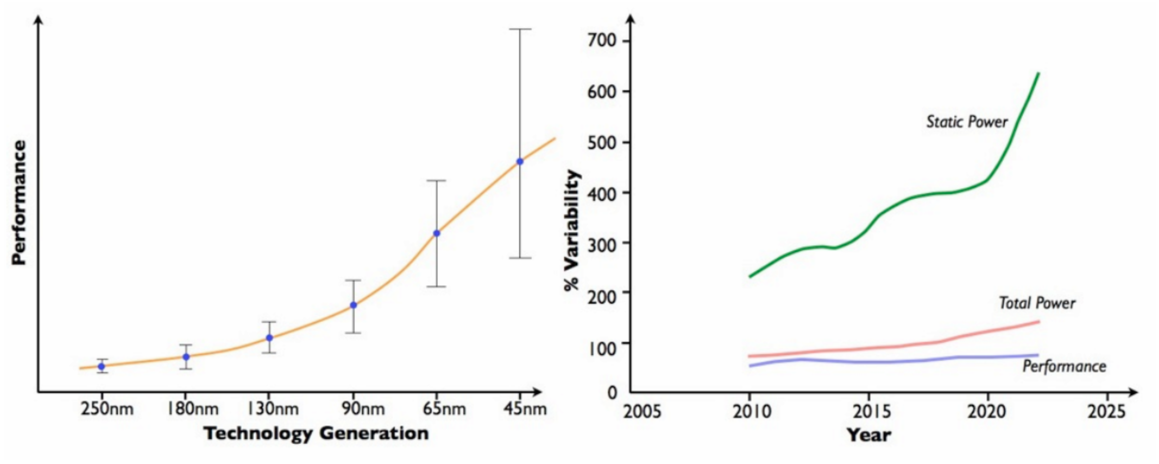


Figure 1.1: Power and performance variability increase with technology [itr09]

- Design margin is usually derived and applied during hardware design time with pessimistic assumptions. Certain system/architecture-level information can be used to reduce the margin by relaxing some of these assumptions.
- System margin may be dominated by the existence of certain pathological scenarios. These pathological scenarios may be caused by behaviors from different layers of system abstraction. Cross-layer approaches may be required as single-layer approaches may be inadequate to avoid these scenarios.
- Some hardware-related phenomena are difficult to monitor by hardware monitors alone. For example, soft error rate is extremely difficult to monitor due to the rare occurrence nature and unique ambient (i.e., location/altitude) dependence. Cross-layer approaches can use system and even cloud-based information for better monitoring.

1.1 Proposed Margin Reduction Approaches

Our efforts in margin reduction can be classified as three parts: monitoring, margining and mitigation.

1.1.1 Accurate and Inexpensive Monitoring for Variability-Aware System

For monitoring, we focus on the circuit performance variability, due to the following reasons:

(i) Circuit delay change is the manifestation of most of the variability effects and many reliability-related phenomena. Reducing delay margin effectively reduces the design margin caused by these variability sources and reliability issues.

(ii) Unlike power, which behaves like "sum of variables", circuit delay is dictated by the slowest path, i.e., behaving like "maximum of variables". This creates additional complexity and unique context-dependence for delay-related issues, including its modeling, monitoring and margining.

(iii) Under many circumstances, delay variation and other types of variations are actually interchangeable. For example, power variation can be translated into delay variation through voltage scaling or body-biasing. Therefore, reducing delay margin can reduce other types of design margin as well.

If hardware variability can be captured and exposed to the software/system level during runtime, corresponding hardware and software adaptation can opportunistically reduce or even eliminate the design margin [Gup12]. The quality of these adaptation schemes depends on the quality of the monitors. Therefore, part of the proposed research focuses on improving the quality (i.e., accuracy and overhead) of these delay monitors.

There are mainly two classes of delay monitors that aim at monitoring circuit performance (i.e., measuring circuit critical path delay). They are replica monitors and in situ monitors.

Replica monitors, also known as canary circuits, [Dra07,Bhu06,Liu10,Cha12b] are standalone circuits which are designed to mimic the timing behavior of the original circuits. By measuring the replica circuit delay, one can estimate the delay of the original circuits. Typically, the replicas are simple circuits, e.g., simple paths or ring oscillators. Therefore, replicas are usually non-intrusive and with low overhead, but may be inadequate to cover the heterogeneity within the original circuits. Furthermore, replica monitors may fail to capture the variation that is local to real circuits such as random manufacturing variations and circuit aging.

In situ monitors measure the delay directly from the circuit paths [Fic10,Kim13,Bow09]. They can accurately capture the real path delay, but with significant overhead, especially when a large number of registers are timing critical.

We propose two circuit performance methodologies. Design-dependent ring oscillator (*DDRO*) [Cha13b] designs and leverages *multiple* replicas to accurately monitor circuit performance. *SlackProbe* [Lai13] inserts in situ monitors at both path intermediate nets and path endpoints, which can greatly reduce the overhead of in situ monitoring.

1.1.2 Margining for Emerging Systems

For system with adaptation, the margin derivation should consider the entire monitor-and-actuate process. Over or under optimizing any single component can result in wasting resources or bottlenecking the performance. We classify the dynamic variation margin and analyze different parts with the choices of monitor/actuator types.

Even though design margin is usually determined at hardware design time without full knowledge of the higher layers of the system stack, certain architecture-level behavior information can be utilized to better estimate the necessary hardware design margin. For example, voltage boosting is a popular architecture-level technique for high-end processors, which use higher supply voltage to achieve temporary performance gain when permitted by the thermal conditions. This makes the boosted supply voltage and maximum operating temperature the worst-case aging conditions for design margining. However, typical power consumption under voltage boosting exceeds the chip's thermal design power (TDP). This makes it impossible to retain the worst-case aging conditions for long time. Therefore, if the architecture-level power/thermal models and management policies are available, the hardware may use a better-than-worst-case operating condition for deriving the necessary margin.

1.1.3 Reliability Mitigation

Some types of delay variation are difficult to resolve through adaptation. For example, high-level adaptations have limited effects on clock skew due to aging in clock distribution network, because the adaptation effect will be applied on all clock paths and thus will cancel each other out.

The clock distribution network usually has balanced structure and invariant signal pattern, which makes it very robust against N/PBTI-induced delay degradation. However, this may not hold for clock distribution networks with clock gating features [Liu12]. The imbalanced clock signal pattern due to clock gating can result in imbalanced degradation on different clock paths, causing N/PBTI-induced clock skew.

We propose *BTI-Gater*, which consists of cross-layer solutions to mitigate N/PBTI-induced clock skew. Two Integrated Clock Gating (ICG) cell circuits are proposed to balance the clock signal pattern. A methodology is also proposed to select the appropriate ICG cell, based on the micro-architecture and architecture context. An example of software sleep scheduling is described in conjunction with *BTI-Gater* to avoid certain pathological aging scenarios.

For certain reliability phenomena, such as soft error rates, it is extremely difficult to make an estimate using hardware monitors. So we have proposed system/cloud-based virtual sensing to capture varying ambient conditions for soft error rates. We also develop a hardware evaluation platform based on an embedded/mobile development board and standard Linux kernel. We demonstrate the use of our platform to evaluate the system's power and radiation-induced soft error rate in presence of system power management schemes, with different application workloads and various hardware design configurations.

1.2 Thesis Outline

The objective of this dissertation is to reduce the system design margin with cross-layer approaches. The key contributions of this work can be summarized as follows:

- *DDRO* replica delay monitoring methodology is proposed in Chapter 2. *DDRO* designs and leverages *multiple* replica monitors to accurately monitor circuit performance.
- *SlackProbe* in situ delay monitoring methodology is proposed in Chapter 3. *SlackProbe* inserts in situ delay monitors at both path intermediate nets and path endpoints, which can greatly reduce the overhead of in situ monitoring.
- Dynamic variation margining, in presence of monitor-and-actuate adaptation, is described in Chapter 4. We classify and analyze different parts of the system margin for monitoring and actuation.
- Hardware reliability margining for circuits in the "dark silicon" era is presented in Chapter 5. We propose methodologies for deriving the reliability margin considering the high-level power/thermal constraints.
- *BTI-Gater* clock gating methodology is proposed in Chapter 6. *BTI-Gater* consists of cross-layer solutions to eliminate N/PBTI-induced clock skew.
- Our study on evaluating and exploiting impacts of dynamic power management schemes on system reliability is presented in Chapter 7. We propose system/cloud-based soft error rate monitors and simple forbidden-state based policies for monitoring and adaptation.

CHAPTER 2

DDRO Replica Monitor

2.1 *DDRO* Motivation and Overview

In order to accurately estimate the circuit delay, replica monitors should be designed to follow the timing behavior of original circuits under variations. We know that the circuit's performance is determined by delay of the slowest path, i.e., critical path. Therefore, the replica should be designed to follow the timing behavior of the critical path.

Due to variations, there may be a number of potential critical paths and they may behave differently under variations, which makes a single replica monitor inadequate. The motivation of *DDRO* is shown in Fig. 2.1, in which each dot represents the delta delay of one critical path under variations of PMOS threshold voltage $Vthp$ (y-axis) and NMOS threshold voltage $Vthn$ (x-axis). The critical path delay sensitivities form natural clusters, which implies that *multiple* replica monitors can be designed to cover these clusters.

An overview of *DDRO* monitoring strategy is shown in Fig. 2.2. First, we extract critical paths of a design and characterize their delay sensitivities to variation sources. Second, we cluster the critical paths based on the sensitivities, and synthesize *DDROs* to match delay sensitivity of the clusters. By matching *DDRO* and cluster delay sensitivities, we ensure that the synthesized *DDROs* have good correlation with the critical paths. Since we use only standard cells (gates) to synthesize the *DDROs*, the design and placement of *DDROs* can be easily integrated with conventional implementation flows. Based on *DDRO* frequencies, we can estimate chip delay during manufacturing or runtime.

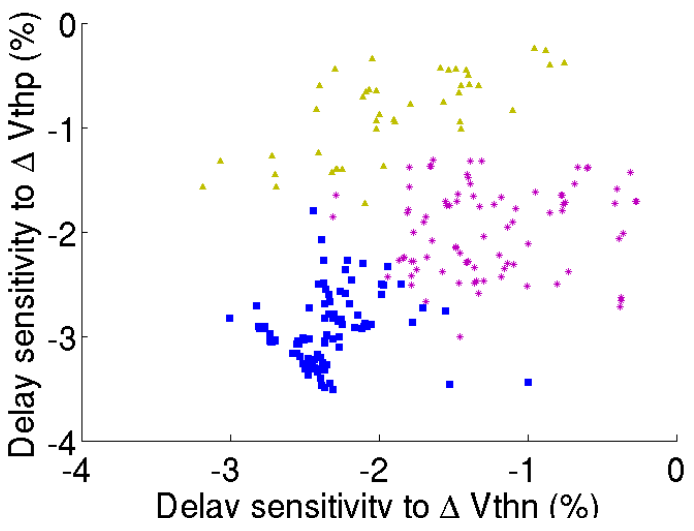


Figure 2.1: Every dot represents delay sensitivities of a critical path in a design. In this example, we cluster the paths into 3 different clusters indicated by different colors. The results are based on SPICE simulation using commercial 45nm process technology. Critical paths are extracted from AES [ope].

2.2 Path Sensitivity Extraction and Clustering

Delay sensitivity of path i (V_i^{path}) can be obtained using finite differences, i.e., taking the $\Delta delay$ by perturbing each variation source by 1σ. After characterizing all path delay sensitivities, we can cluster the critical paths. The objective function of the clustering is defined as

$$\text{minimize} \sum_{i=1}^{N} (w_i \times |\mathbf{V}_i^{path} - \mathbf{V}_k^{ro}|) \tag{2.1}$$

where the summation is taken over paths i in cluster k, and w_i is the probability of a critical path delay exceeding the clock period of the design. The weight factor w_i is added so that we can impose a higher penalty for having mismatched delay sensitivities on a path with higher probability to fail (less timing slack). Detailed derivation of w_i can be found in [Cha13b].

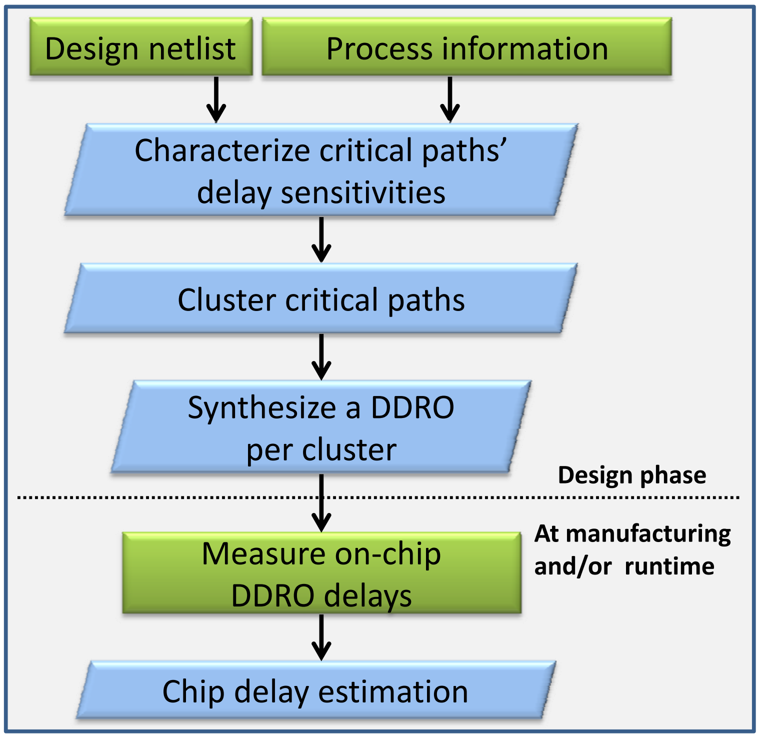

Design netlist
Process information
Characterize critical paths' delay sensitivities
Cluster critical paths
Synthesize a DDRO per cluster
Design phase
At manufacturing and/or runtime
Measure on-chip DDRO delays
Chip delay estimation

Figure 2.2: Overview of *DDRO* design methodology.

2.3 *DDRO* Synthesis

2.3.1 Gate Module

For the ease of synthesizing *DDRO*, we use gate module as basic building block. A gate module is constructed as several identical gates connected in series as illustrated in Fig. 2.3.

This repeated pattern can help decouple the load and slew interaction between gates. Simulation results in Fig. 2.4 show that the sensitivity difference due to different input slew and output load is reduced from 0.15% to 0.03%, as the number of stages in a gate module increases from 1 to 15. In this work, we use 5-stage gate modules as a trade-off between stability of sensitivity and total area of a gate module. For a gate with multiple input pins, gate delays through different input pins will have different delay sensitivities. Thus, each gate module type is defined with respect to a specific input pin. Extra input pins of a multi-

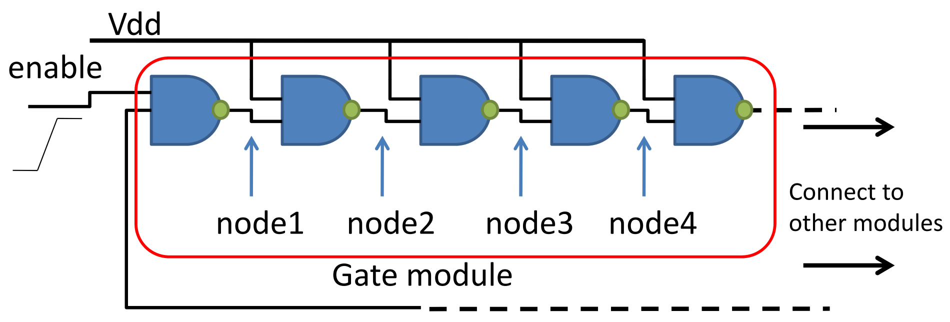


Figure 2.3: Illustration of a gate module in a *DDRO*.

input gate are assigned to high or low to make a gate module inverting or buffering (see Fig. 2.3).

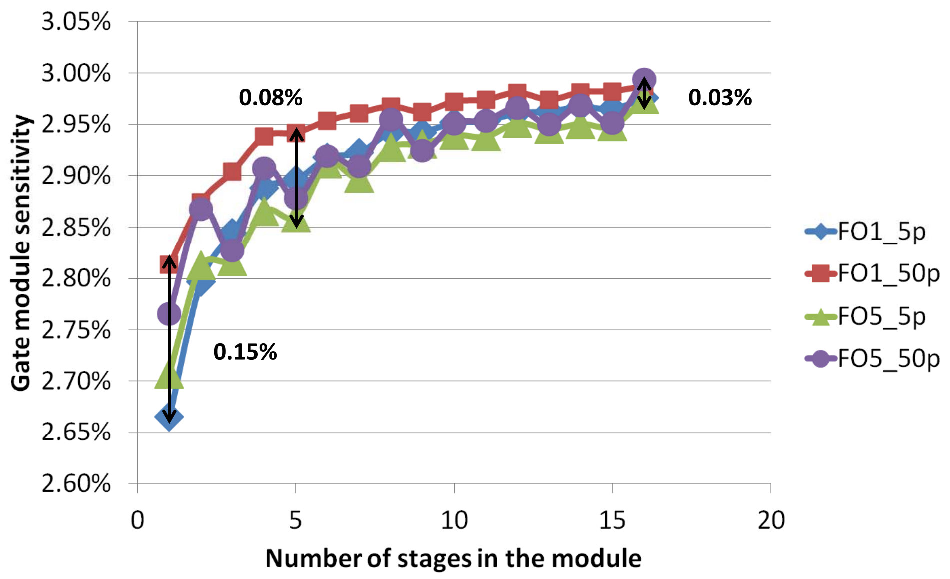


Figure 2.4: Simulation results show that the sensitivities under different input slews {5ps, 50ps} and output loads {FO1, FO5} combinations converge as the number of stages in a gate module increases.

Since the interconnect also affects path delay sensitivity, we use different wirelength in building our gate modules. Gate modules with different wirelength are considered as different instance types even if they have the same gate type.

2.3.2 ILP-based Synthesis

Given a delay sensitivity target ($\mathbf{V}^{ro}$) obtained from path clustering, we want to choose the number of each gate module in a *DDRO*, so that the delay sensitivities of the *DDRO* match the targeted delay sensitivities. Because of its unique structure, gate module delay variation is less sensitive to input slew and output load. Since each gate module type is instantiated a discrete number of times, we can formulate *DDRO* synthesis as an integer linear programming (ILP) problem, where the integer variable is the number of certain gate modules in the synthesized *DDRO*. Appropriate constraints are applied to limit the maximum RO length. Detailed description of gate module and ILP formulation can be found in [Cha13b].

2.4 Delay Estimation

2.4.1 Path-based Estimation

As shown in Fig. 2.2, at manufacturing and/or runtime, *DDRO* delay is measured and used to estimate chip delay.

Each critical path delay can be estimated by *DDROs*. Given M *DDROs*, we can represent delay sensitivity of each path ($\mathbf{V}_i^{path}$) as a linear combination of *DDRO* delay sensitivity($\mathbf{V}_k^{ro}$ ($k = 1, ..., M$)) as in Equation (2.2).

$$\mathbf{V}_i^{path} = \sum_{k=1}^{M} b_{ik} \cdot \mathbf{V}_k^{ro} + \mathbf{V}_i^{res\_path} \tag{2.2}$$

where b_{ik} is a $[1 \times M]$ matrix containing constant coefficients and $\mathbf{V}_i^{res\_path}$ is a $[1 \times Q]$ matrix that represents the decomposition residue.

Using b_{ik}, we can estimate the path delay by:

$$\begin{aligned} d_i^{path} &= d_i^{nom\_path}(1 + \sum_{k=1}^{M} \overbrace{(b_{ik}\mathbf{V}_k^{ro} \cdot \mathbf{g})}^{\text{measurable}} + \underbrace{u_i}_{\text{uncertainty}}) \\ \text{where } \ u_i &= l_i^{path} + \mathbf{V}_i^{res\_path} \cdot \mathbf{g} \end{aligned} \tag{2.3}$$

where $\mathbf{g}$ is global variation vector, l_i^{path} is the local variation of the path. Equation (2.3) shows that d_i^{path} consists of a measurable term and an uncertainty term. While the value of the *measurable* term can be determined from the delays of *DDROs*, the value of the *uncertainty* term cannot be measured directly.

2.4.2 Cluster-based Estimation

The path-based estimation method requires one estimation for each critical path, which can cause significant computation and storage overhead. To reduce the overhead, we propose to utilize the clustering nature of critical paths and to have one estimation for each path cluster. We calculate the maximum delay of paths in each cluster using the method in [Vis06], assuming that the means of path delays correspond to their nominal values. The outcome of this step gives us the expected maximum delay. But more importantly, it also extracts the sensitivity of the maximum delay to variation sources ($\mathbf{V}^{max}$). Similar to the path-based approach, we treat the cluster as a pseudo path with delay sensitivity $\mathbf{V}^{max}$. Simulation results show that the estimation error of this approximation approach compared to the reference method is very small.

2.5 Simulation and Silicon Result Highlights

To validate *DDRO* performance monitoring methodology, we synthesized, placed and routed processor benchmark circuits using a commercial 45nm SOI technology. Some simulation results of *DDRO* on Cortex-M0 [Corb] are highlighted in Fig. 2.5. For global variation only case, using more *DDROs* can dramatically decrease mean overestimation (i.e., required delay margin). But this benefit becomes less in presence of local variation. The results show that the estimation error of cluster-based approximation, compared to the path-based approach, is very small.

A testchip was taped out with *DDRO*-based performance monitoring using a 45nm IBM SOI technology with dual-V_{th} (RVT and HVT) libraries. The testchip has an ARM Cortex-M3 microprocessor [Corc] with five *DDROs*. For comparison, we also implemented several inverter-based ROs in the testchip. The silicon measurement results are shown in Fig. 2.7. The measurement results show that by using five *DDROs*, we can reduce the mean delay estimation error by 35% (from 2.3% to 1.5%) compared to generic inverter-based ROs.

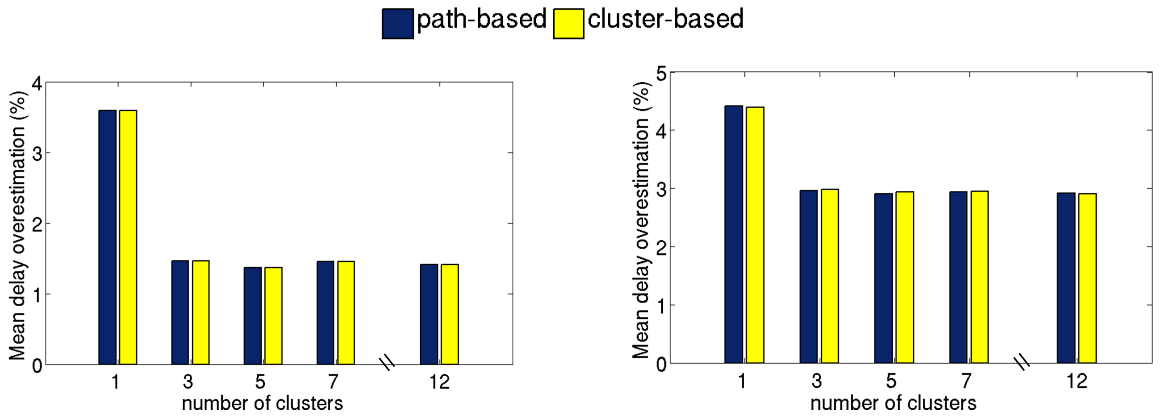


Figure 2.5: Simulation results of *DDRO* on Cortex-M0 with global variation only (left) and both global and local variations (right).

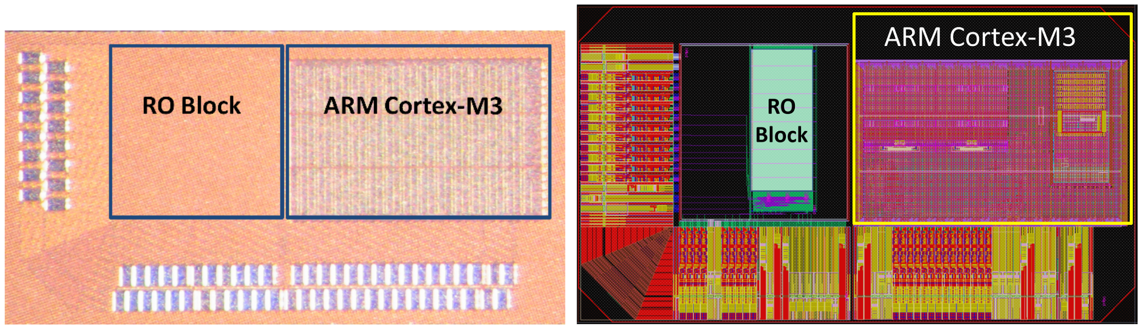


Figure 2.6: Testchip die photo and layout illustration

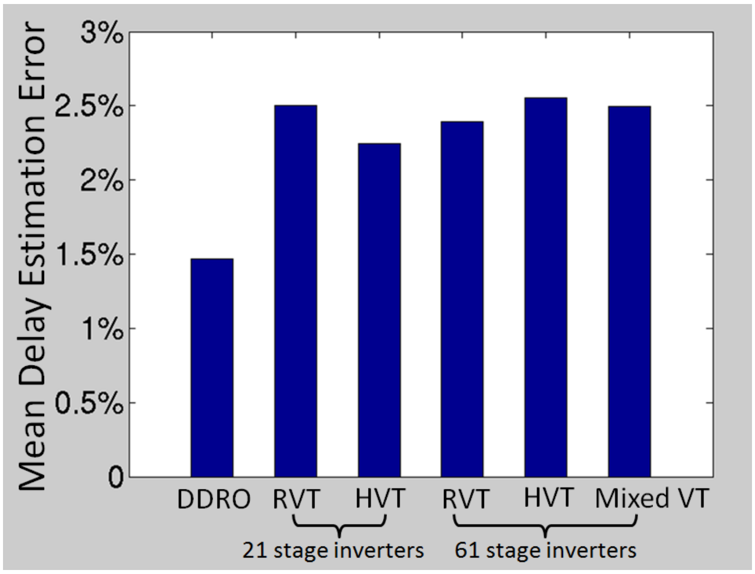


Figure 2.7: Mean delay estimation error obtained from *DDROs* and inverter-based ROs. Estimation errors are calculated by taking the absolute difference between normalized estimation and normalized chip delay. RVT and HVT are the two Vt options for the inverters.

2.6 *RedCooper*: A Testbed for Variability-aware System

In this section, we will present our implementation of a complete end-to-end system [Aga] that demonstrates the use of hardware monitors with both hardware and software adaptation. We first describe the software adaptation concept of variability-aware software duty-cycling. Then we present our testbed implementation and variability-aware system demonstration.

2.6.1 Variability-Aware Software Duty-Cycling

For battery-powered embedded sensing system, the total lifetime energy is usually constrained. In order to meet the lifetime requirements, one particularly common power management techniques is duty cycling, where the system is at default in a sleep state and woken up periodically to attend to pending tasks and events. A system with higher duty cycle may, for example, sample sensors for longer intervals or at higher rates, increasing data quality. A typical application-level goal is to maximize quality of data through higher duty cycles, while meeting a lifetime goal. Conventional approach determines duty cycles by either worst-case power specifications or datasheet power values, which may be heavily guardbanded. If the system's power consumption can be measured and exposed to the software, the application may be able to adapt its duty cycle rate and increase its QoS opportunistically according to the hardware power consumption status [Wan13a].

For example, for a fixed lifetime energy budget E and specified targeting lifetime constraints L, the software duty cycle rate DC can be calculated through the following equation [Wan13a]:

$$\begin{aligned} P_A \cdot DC + P_S \cdot (1 - DC) &= \frac{E}{L} \\ DC &= \frac{E - L \cdot P_S}{L \cdot (P_A - P_S)} \end{aligned} \tag{2.4}$$

where P_A and P_S are the active and sleep power consumption respectively. By determining P_A and P_S on a per-instance basis, the duty cycle may be tailored to maximize active time for each individual sensor under a given deployment scenario (temperature profile, lifetime requirement, battery capacity).

There are several different ways that such an opportunistic stack may be organized as shown in Fig. 2.8. The scenarios differ in how the sense-and-adapt functionality is split between applications and the operating system. In this work, we use the last scenario, where variability is largely offloaded to the operating system.

2.6.2 Testbed Implementation and System Demonstration

The testbed is built upon *DDRO* testchip (Fig. 2.6). On the testchip, there are on-chip performance monitors (*DDRO*) and leakage sensors. For the purpose of demonstrating software duty-cycling, we also implement on-board power sensors to measure the power consumption of the testchip. the power of the Cortex-M3 processor and the (on-chip) SRAM memory are measured separately. A picture of the testbed board is shown in Fig. 2.11. Because *DDRO*

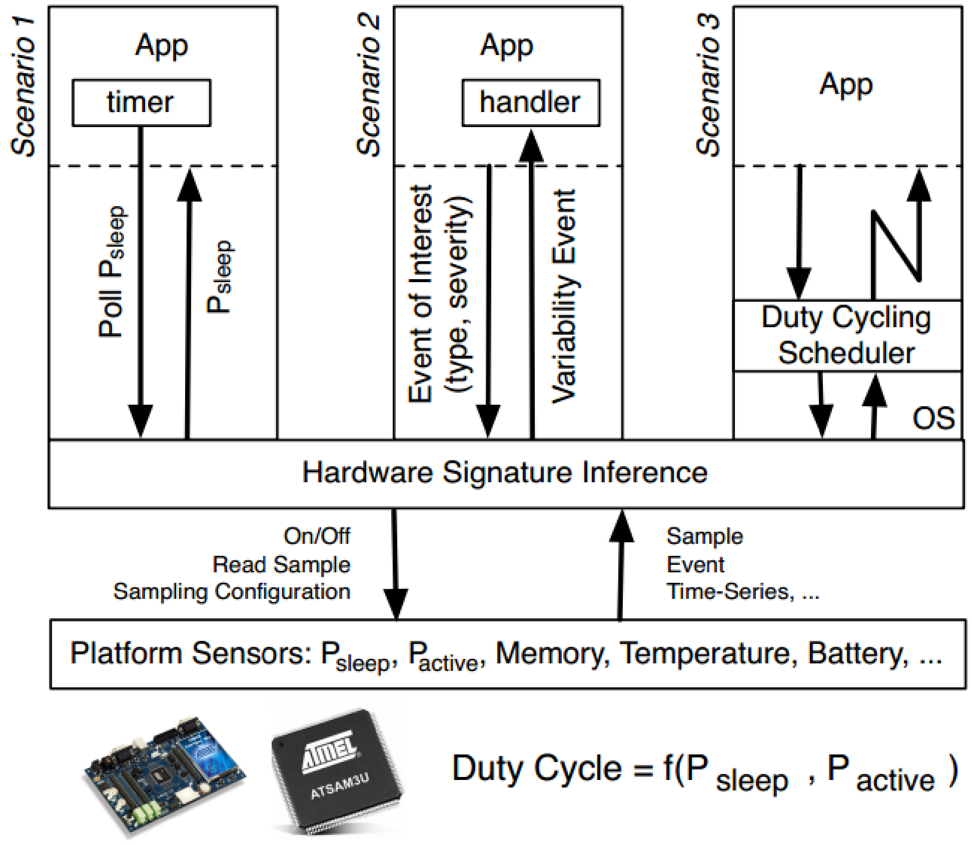


Figure 2.8: Designing a software stack for variability-aware duty cycling [Wan13a]

and the processor are implemented as separate blocks when taping-out the testchip, we use on-board MCUs [mbe] to sample the monitor readings and feed into the on-chip SRAM.

The software running on the M3 core is shown in Fig. 2.10. The operating system (OS) is based on CoOS [CoO]. Three tasks are implemented within the OS:

- Task I sends the sensing request by writing to a pre-specified memory location. Upon seeing the request, the on-board MCUs will start reading the sensor values and write the sensor readings directly to certain memory locations.
- Task II acts as the central adaptation center, which reads the sensor readings, including *DDRO* frequencies, current drawn by the M3 core, current drawn by the on-chip

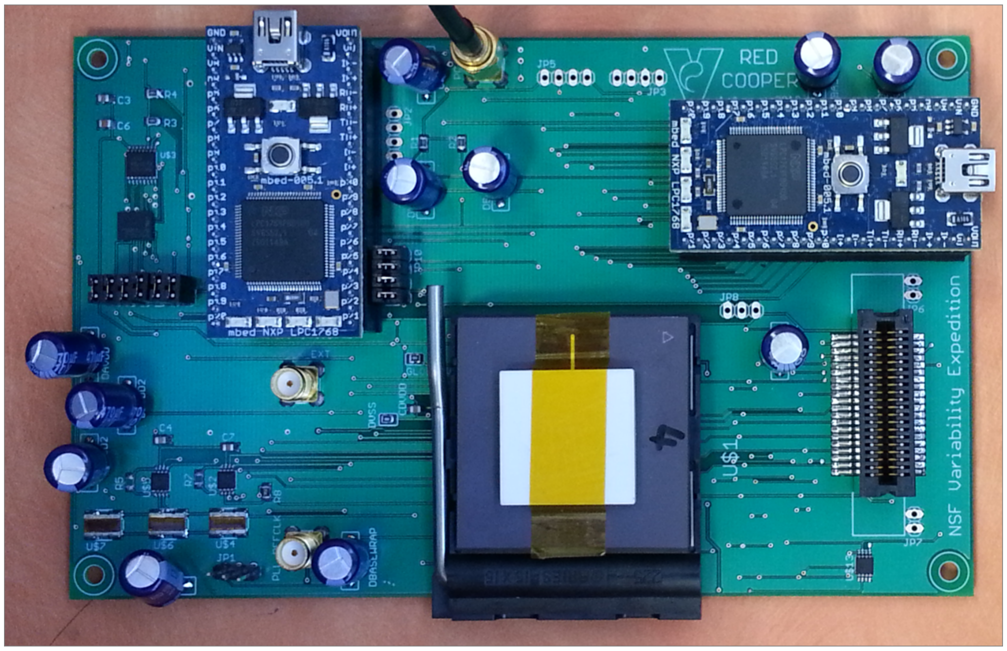


Figure 2.9: *RedCooper* Testbed.

SRAM, and the on-chip leakage sensor. *DDRO* frequencies are used to calculate the performance slack and determine the adjusted voltage. The current and leakage sensor values are used to calculate the feasible duty cycle using Equation (2.4). The duty cycle rate is further translate to the number of iterations for Task III.

- Task III is the main application running in the OS, which calculates the value of π with its best effort under the constrained duty cycle.

In this demo, all three OS tasks are fired every 10 seconds. We set the processor to run at a fixed 600 MHz clock frequency. The active power includes both the core and SRAM power consumption. The sleep power includes the SRAM power and the projected leakage power of the core.

A snapshot of the entire hardware is shown in Fig. 2.11. We have two copies of *RedCooper* running side-by-side. Each testbed is equipped with an LCD reporting the system status.

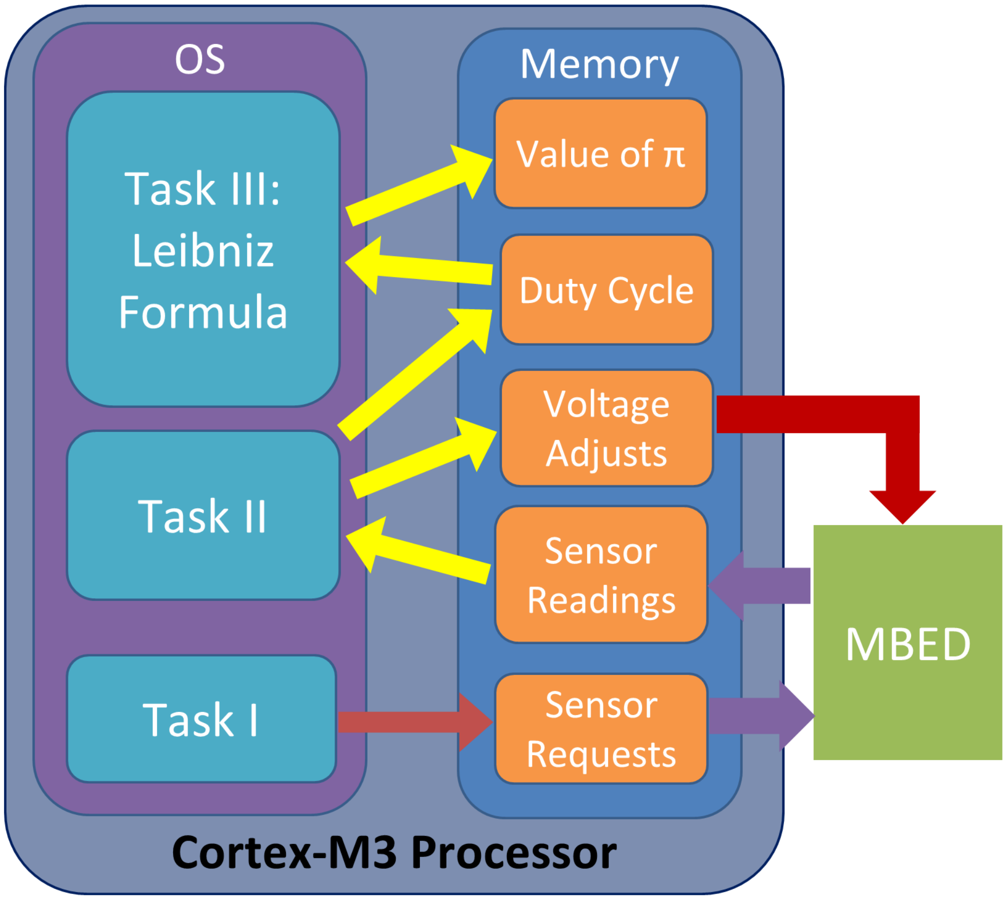


Figure 2.10: Software Adaptation Illustration.

Some results are highlighted in Table 2.1.[1] At room temperature, Instance B system has smaller sleep power than Instance A, which implies the potential of achieving high duty cycle rate. Therefore, the adaptive software set the number of iterations (DC) at 49 for Instance B and the calculation error is smaller. After heating up Instance A from room temperature to about $45°C$, the system shows its capability for both runtime software and hardware adaptation (see last column in Table 2.1). The voltage is boosted to compensate the performance loss. Software duty-cycle is reduced to compensate the increased power consumption. If the system is without hardware sensors and designed for the worst-case scenario (including process variations and temperature fluctuations), the number of iterations

[1]The complete demo video can be found at http://nanocad.ee.ucla.edu/Main/Codesign

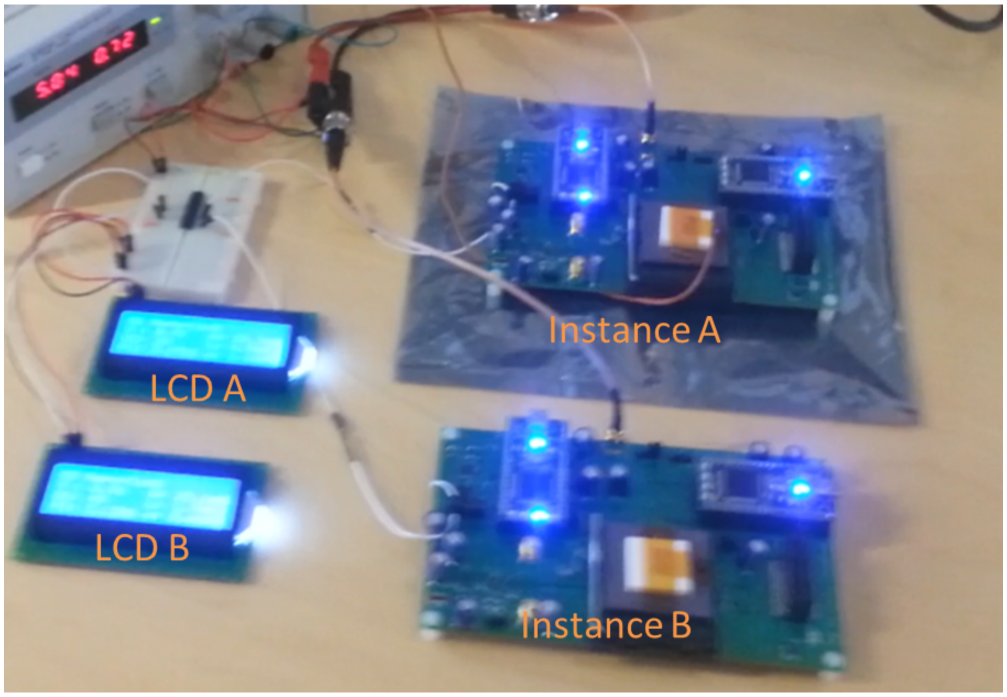


Figure 2.11: A snapshot of the demo hardware.

(DC), as in this demo, will be at most 18 (the case for Instance A after heat-up) with 1.865% calculation error. With the hardware sensors and adaptations, we are able to achieve 49 iterations and calculation error as small as 0.668%.

Table 2.1: System Demonstration Result Highlights

	Instance A (room temperature)	Instance B (room temperature)	Instance A (after heat-up)
Supply Voltage	0.93 V	0.96 V	0.96 V
Active power	23.81 mW	24.29 mW	26.50 mW
Sleep power	4.74 mW	1.63 mW	7.58 mW
DC	27	49	18
Calculated π	3.1028	3.1206	3.2002
Calculation error	1.235%	0.668%	1.865%

CHAPTER 3

SlackProbe In Situ Monitoring Methodology

With *DDRO* methodology, replica monitors can predict global variation near perfectly. But the accuracy is still limited and constrained by local variation. In situ monitors can capture the actual path delay, including local variation, but usually incur significant overhead. We observe that most of existing work focuses on monitoring destination registers. In this chapter, we introduce a novel and flexible in situ monitoring methodology, *SlackProbe*, which inserts monitors at both path endpoints and path intermediate nets.

3.1 *SlackProbe* Introduction

With increasing amounts of manufacturing variability, ambient fluctuation and circuit wear-out (e.g., NBTI, HCI etc.), it is necessary to identify chip delay either statically (e.g., speed binning) or dynamically with both hardware and software adaptive schemes [Gup12]. There are various classes of monitors that are targeted at measuring circuit path delay.

Canary or replica circuits [Dra07, Cha12a] are stand-alone circuits which are intended to mimic the timing behavior of the original circuits. The delay of the real circuit can be estimated through measuring delay of the replicas. Tunable [Tsc09] and design-dependent [Cha12a] replica circuits can reduce the mismatch of the real circuit and replica. Replica monitors are usually non-intrusive, but may fail to capture the variations that are local to real circuits such as random manufacturing variations and circuit aging.

In situ monitors measure the delay directly from the circuit paths. Fick et al. [Fic10] use a Time-to-Digital Converter (TDC) to measure the critical path delay. Wang et al. [Wan08] measure delay by reconstructing the critical path as Ring-Oscillators (ROs). Another approach to measure circuit path delay is to measure the timing slack. Since critical paths typically end at registers, special flip-flops can be used as slack monitors. Razor [Ern03, Das09] uses customized flip-flops to detect timing failures due to setup time violation and correct them through a pipeline flush or architectural replay. Similar approaches that reduce timing margin, but not to the point of failure, include delaying data signals [Fuk09], advancing clock signals [Reb10] or using different flip-flop structures [Eir07, Kur10, Aga07, Das10, Sat07].

In situ monitors can accurately capture the real path delay, but with significant overhead, especially when large number of registers are timing critical. Some methods can be used to reduce the overhead (e.g., [Hir12]), but with a loss in accuracy. Better monitor designs, e.g., [Kim13], can also reduce the overhead, but are still fundamentally limited by the large number of monitors requited. We observe that existing methods focus *exclusively* on monitoring path endpoints (i.e., destination registers). In this work, we propose *SlackProbe*, a low overhead, in situ, on-line timing slack monitoring methodology. *SlackProbe* monitors in situ timing slack of selected circuit nets, including intermediate nets along circuit paths, which is more power and area efficient. This work is an extension of [Lai13]. The key contributions of our work include the following:

(i) We propose a novel slack monitoring methodology allowing placing monitors at intermediate nets along circuit paths

(ii) We develop a metric named "*opportunism window*", which allows us to flexibly select the set of critical paths to be monitored.

(iii) We formulate and convert the path-based monitor insertion formulation into a edge-based Linear Programming (LP) problem and solve it near its theoretical lower bound

(iv) We perform a thorough analysis of the potential benefits and caveats of such a monitoring methodology, including monitoring cost, monitoring efficiency and observability, timing margin and design perturbation.

The rest of the chapter is organized as follows: Section 3.2 gives an overview of the proposed monitoring methodology. Section 3.3 describes the critical path selection and circuit graph reduction process. Section 3.4 presents the monitor insertion problem formulation and solution. Section 3.5 discusses the monitor cost metrics and comparison between *SlackProbe* and conventional approaches. Section 3.6 presents the experimental results. Section 3.7 concludes this chapter.

3.2 *SlackProbe* Overview

3.2.1 Monitor Working Principle

The monitor working principle is shown through an example in Fig. 3.1. If a monitor is inserted at an intermediate node A, a "probe", which consists of delay matching gates and a transition detector, is connected to A through a minimum size inverter. Signal transitions at node A are transferred through the delay chain to the transition detector and compared with the incoming clock edge. If the transition is close to its Required Arrival Time (RAT), i.e., within the margin window as in Fig. 3.1, a corresponding signal transition will arrive at node E after the clock edge. This triggers the transition detector and flags a signal indicating an impending delay failure.

The monitor inserted at node A is capable of monitor the delay of all critical paths passing through A. As shown in Fig. 3.1, instead of monitoring all four destination registers, *SlackProbe* can use only two monitors while achieving the same path coverage.

Different transition detector designs as in [Reb10, Bow09]. can be applied here. *SlackProbe* also allows monitors to be inserted at path endpoints where monitors as in [Ern03, Das09, Fuk09, Eir07] can be used as well. Since the additional margin makes the monitor detect an impending timing failure rather than an actual one, there is no datapath metastability issue as raised and discussed in [Bow09]. The metastability issue of the monitor signal either results in a more pessimistic detection or is guardbanded by the monitor delay margin.

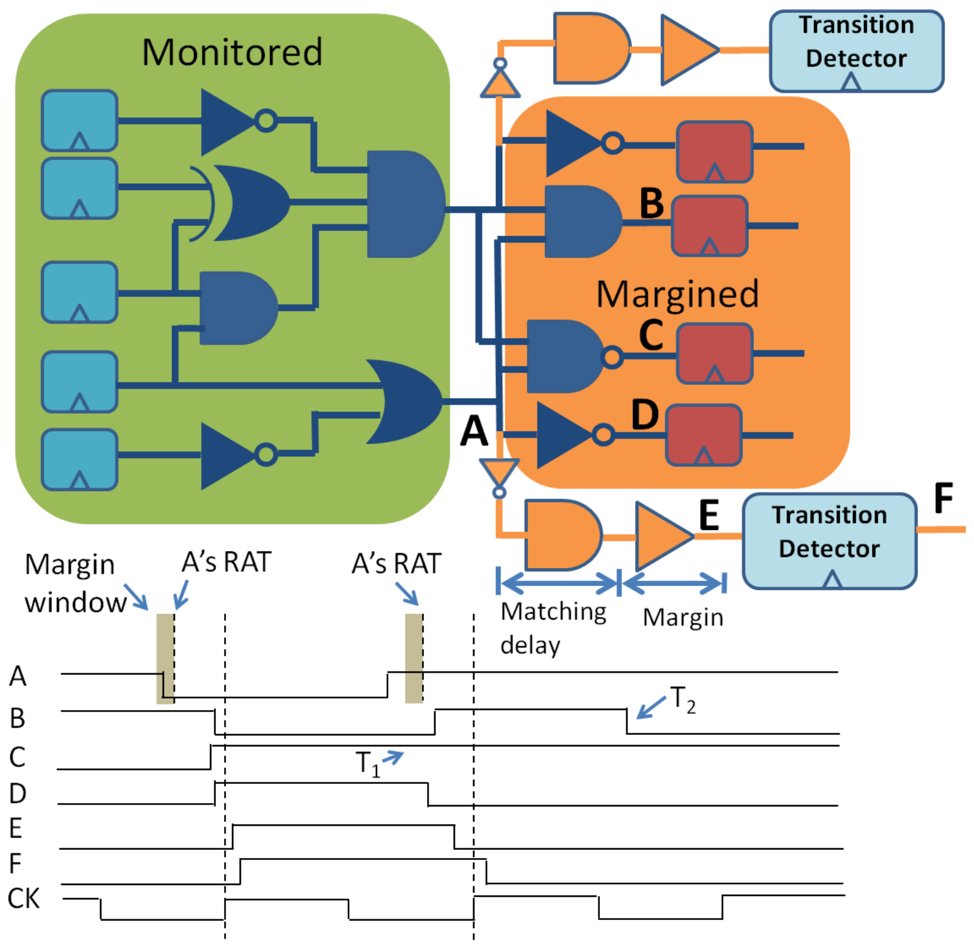


Figure 3.1: *SlackProbe* working principle. As shown in the timing diagram, compared to inserting monitors at destination registers, the monitor inserted at A can monitor the path delay even when the transition does not propagate to the destination register (i.e., T_1 at C). But the monitor inserted at node A cannot capture transitions that do not pass through A (i.e., T_2 at B).

3.2.2 Monitor Insertion Flow

With the proposed monitoring strategy, the problem now becomes *when*, *where* and *how* to insert these monitors. In this work, we propose the monitor insertion flow as in Fig. 3.2. Different alternatives will also be discussed and compared against conventional approaches.

Monitor insertion starts with a placed and routed design, as the timing information is more accurate at this stage. Since we only care about timing-critical paths, a path selection process is applied to extract timing-critical paths and to construct the critical path graph. Then, monitor locations are picked from the graph using our proposed method. For each of the monitor locations, a delay cell path is synthesized. The final insertion flow is similar to Engineering Change Order (ECO) where the monitors are incrementally placed and routed. ECO metrics like those in [Lee12] are applied when picking monitor locations to minimize the perturbation to the original design.

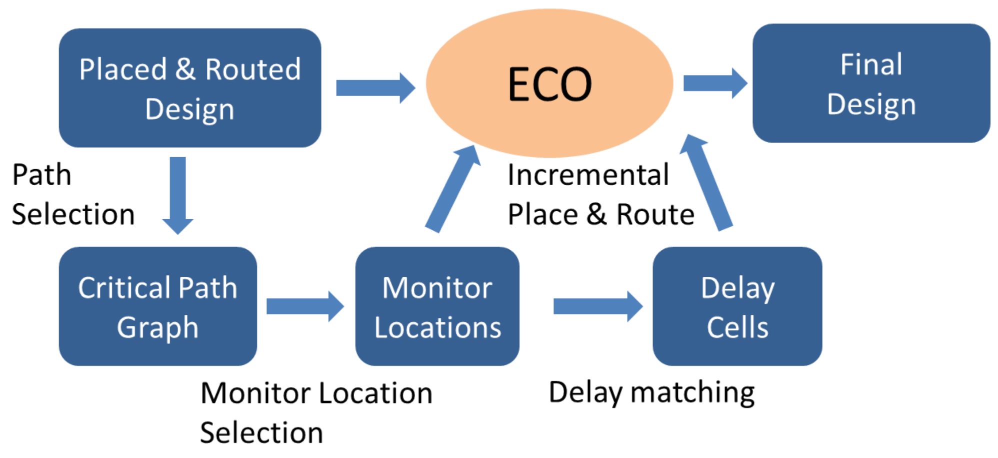


Figure 3.2: Monitor insertion flow

3.2.3 Possible Applications of the Monitors

Since different applications will have different requirements for the monitor, we discuss possible monitor applications here as examples before introducing the detailed implementation flow.

One possible application is to use the monitors as timing-failure event predictors and combine them with some of the existing error resilience mechanisms like in [Das09, Cho14, Foj13]. In this case, the monitors have to capture all signal transition events that may result in timing errors.

Another possible application is to use the monitors as speed sensors which indicate whether current operation condition is close to possible timing failure or not. This can be used by systems with adaptation capabilities like Adaptive Voltage Scaling (AVS), Adaptive Body Bias (ABB) or Dynamic Voltage and Frequency Scaling (DVFS) to account for manufacturing variations, ambient conditions as well as circuit aging effects such as Negative Bias Temperature Instability (NBTI), Positive Bias Temperature Instability (PBTI) and Hot-Carrier Injection (HCI). Since variations are either static or changing slowly, the monitor requirements can be relaxed to capture only the delay changes instead of all transition events.

3.3 Path Selection and Graph Reduction

3.3.1 Path Selection Criteria

As shown in Fig. 3.2, given a placed and routed design, the first step is to identify the part of the design that may be timing critical. Depending on the application, different criticality criteria may be applied for the selection process.

In this work, we propose a flexible path selection method by introducing user-defined *opportunism window*. As illustrated in Fig. 3.3, *opportunism window* and monitor margin (discussed in detail in Section 3.4.1) will dictate the potential monitoring benefits. Typical worst-case design sets its delay margin for the worst-case scenario, i.e., all chips will run at a frequency slow enough for the worst-case chip delay regardless of what the actual delay is. In the presence of monitors, we may reduce the design margin and decrease the default operating clock period. The paths whose worst-case delay exceeds the default operating clock period should be selected and monitored. The amount of design margin reduction is called *opportunism window*, within which the circuit will operate opportunistically at its best effort. The size of *opportunism window* determines the total number of circuit paths to be monitored, thus affecting the monitoring overhead. There is a natural trade-off between monitoring benefits and monitoring overhead when deciding the *opportunism window* and monitor delay margin. We will discuss and explore this trade-off in later sections and experiments.

This path selection method does not require any knowledge of correlation in the variations between the delay of different paths. Therefore, it can be used to select paths for applications like aging sensors, where exact delay degradations are context dependent with little pre-assumed correlation.

Another possible path selection method is based on the process corners as described in [Lai13], where paths are selected based on their timing slacks at particular process corners. Other path selection methods such as statistical methods [Xie10] can also be used in *SlackProbe* depending on the specific applications.

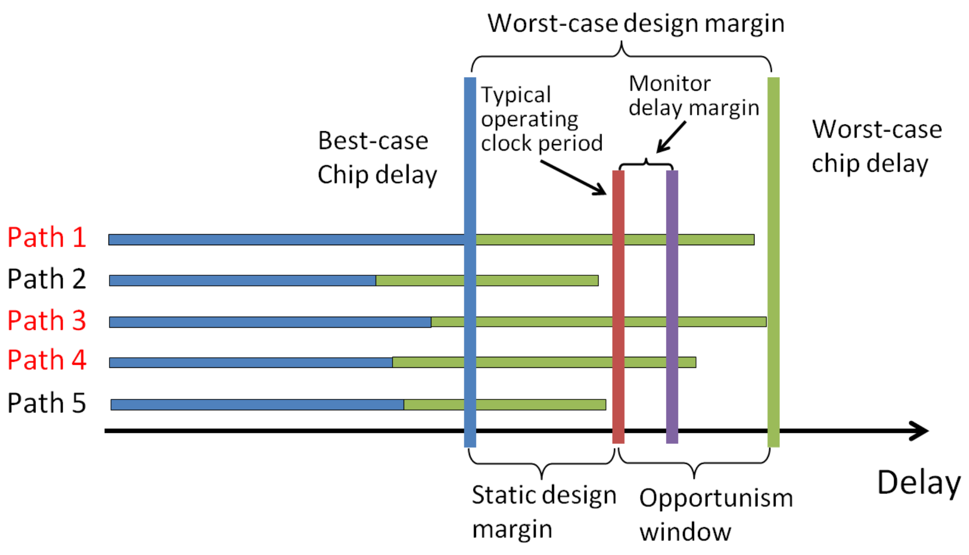


Figure 3.3: Opportunism window is the margin saving comparing to worst-case design. Monitor delay margin is the delay margin of the delay matching chain which will be discussed in detail in Section 3.4.1.

3.3.2 Circuit Graph Reduction

Although *opportunism window* is a path-based criteria, we can utilize the block-based property of Static Timing Analysis (STA) and construct a circuit graph which contains all the selected critical paths.

If a path is selected as timing critical as illustrated in Fig. 3.3, the path's worst-case delay falls into the *opportunism window*. Equivalently, if we run STA using the worst-case corner library, it will also imply that the path's timing slack at the worst-case corner is smaller than the size of *opportunism window*. Based on the following two key properties of STA:

(i) The slacks of all pins along a path are equal or smaller than slack of the path

(ii) The slack of a path equals to the largest slack of all pins along that path

We can obtain a reduced circuit graph by removing the pins with slacks larger than the size of *opportunism window* and then removing the unconnected gates/nets. The critical paths in the original circuit graph are still preserved in the reduced graph because of property 1. All paths in the reduced graph are timing critical because of property 2. Therefore, we can work on the reduced graph instead of the original circuit graph when analyzing the monitor insertion. Unless otherwise specified, circuit graph discussed in the rest of the chapter refers to the reduced graph.

3.4 Monitor Location Selection

In this section, we describe our proposed monitor location selection method. We first discuss how monitors interact with different paths. Then we formulate the monitor location selection problem and describe our proposed solution.

3.4.1 Monitor Coverage and Delay Margin

After selecting the critical paths and obtaining the reduced circuit graph, *SlackProbe* will determine the set of candidate locations for monitor insertion. Before analyzing the location selection problem, we first discuss how paths are monitored by the inserted slack monitors.

3.4.1.1 Delay Margin for Paths

If the monitor is placed at some intermediate net, the path delay before the monitor can be captured by the monitor. But some extra delay margin will be required for the remaining part of the path. As shown in Fig. 3.4, there are three types of relations between monitor and a path:

(i) The path passes through the net, for example path A in Fig. 3.4. Since the delay up to G4 can be captured by the monitor, the delay path should account for the delay uncertainty of G6.

(ii) The path branches out at some net before the monitor, for example path B in Fig. 3.4. Depending on the application and gate type of G4, the monitor may be treated as being inserted between G3 and G5 with G4 as part of the delay matching. If the application is speed sensing, path B can be considered as being monitored with delay uncertainty of G4, G5 and G8. If the application is event detection, only G4 with gate types that are transparent to signal transitions (e.g., inverter, buffer) are allowed.

(iii) None of the path instances fall in the fan-in cone of the monitor net. In this case, we consider that the path is not monitored.

3.4.1.2 Delay Margin for Monitors

Although different paths may require different delay margin, each monitor will have only one margin matching chain. The monitor margin should account for worst delay uncertainty after

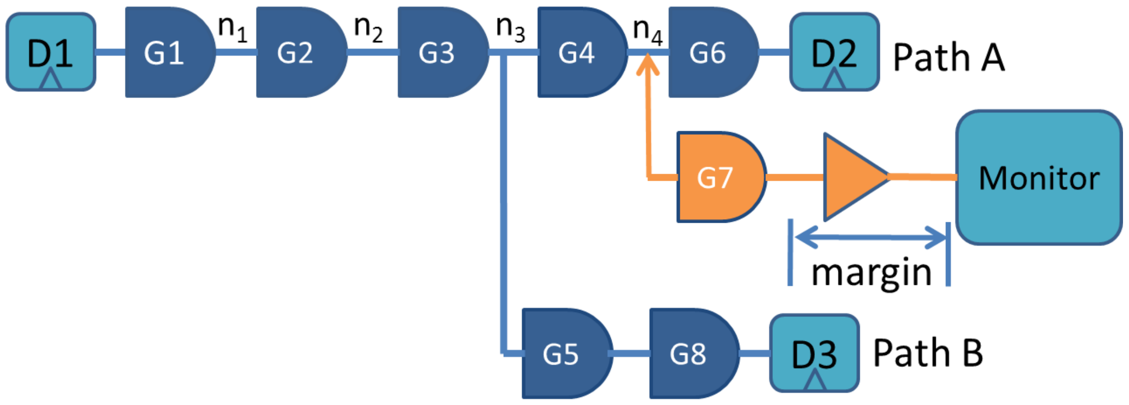


Figure 3.4: Example of path-monitor pairs

monitor insertion point and guarantee that the delay chain is always slower than margined part of monitored circuit paths.

In the example in Fig. 3.4, the delay margin should account for the mismatch between original path (i.e., from n_4 to D_2) and delay matching chain (i.e., from n_4 to the monitor). Theoretically, the best case delay of the delay matching chain should match the worst case delay of the original path. But this may be too pessimistic since the delay is likely to be correlated. Similar to on-chip variation modeling, in this work the delay chain is designed so that its delay at typical process corner matches the worst case delay of the original path. In Fig. 3.4, the equivalent delay margin in this case equals the delay of $G6$ at slow process corner (i.e., delay of the delay matching chain at typical process corner) minus the delay of $G6$ at typical process corner. This margin is considered as the delay uncertainty of $G6$.

The required delay margin for each circuit nets can be calculated based on conventional STA tool flow. We first perform timing analysis using typical process corner libraries and obtain the arrival time d_i^{tt} and timing slack s_i^{tt} for each net. Then we perform timing analysis using slow process corner libraries and obtain the arrival time d_i^{ss} and timing slack s_i^{ss} as well. The required delay margin can be derived by:

$$Margin_i = (s_{tt} - s_{ss}) - (d_{ss} - d_{ss}) \tag{3.1}$$

Methods as in [Cha12a] or [Tsc09] can also be applied to synthesize a replica-like path to reduce the margin for delay mismatch due to global variation.

3.4.1.3 Overall Margin

Because the final delay margin for the entire circuit will be dominated by the monitor with the largest margin, we define the delay margin cost as the maximum monitor delay margin constraint ϵ on each monitor. The delay margin ϵ is used to determine the set of feasible monitor candidate locations and the corresponding circuit nets that can be monitored.

For example, for a given ϵ, we can analyze the implication of inserting a monitor at a net n_i. If the margin required by n_i is smaller than ϵ, all paths passing through n_i will be monitored. Depending on the application, we may also want to consider another net n_j in the fan-in cone of n_i (like n_3 in Fig. 3.4). If the margin required by paths branching out at n_j is also smaller than ϵ (like path B in Fig. 3.4), n_j can also be monitored by the monitor at n_i. All paths passing through n_j will be monitored. Therefore, we can define a set I_{n_i} as the nets that can be included as being monitored by the monitor at n_i.

If we represent a path as the set of nets it passes through, the timing margin constraint can be stated as: with a given monitor delay margin constraint ϵ, for any critical path P_k, there exists a monitor at net n_i, such that $P_k \cap I_{n_i} \neq \emptyset$. For different candidate locations, monitor insertion may have different cost in terms of power, design perturbation etc. Here we use c_i as the cost associated with net n_i. The detailed derivation of c_i will be discussed in Section 3.5.

3.4.2 Problem Formulation

For a given circuit, a graph can be constructed by making the nets as nodes $\mathbf{N} = [n_1 n_2 ...]^T$ and interconnect as directed edges. We denote $\mathbf{x} = [x_1 x_2 ...]^T$ as the decision vector where

$x_i = 1$ when a monitor is inserted at n_i and 0 otherwise.

Since inserting a monitor at n_j implies that all nodes in I_{n_j} are monitored, we define set $O_{n_i} := \{n_j | n_i \in I_{n_j}\}$. A vector $\mathbf{y} = [y_1 y_2 ...]^T$ can be derived as:

$$y_i = \sum_{n_j \in O_{n_i}} x_j \tag{3.2}$$

So $y_i \geq 1$ implies that n_i is monitored because there exists some $x_j = 1$ and $n_i \in I_{n_j}$. Equation (3.2) can be represented as matrix representation $\mathbf{y} = \mathbf{Q}\mathbf{x}$.

A critical path matrix can be generated from the circuit graph as:

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots \\ p_{2,1} & p_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \textit{where } p_{k,i} = \begin{cases} 1 & \textit{if } n_i \in P_k \\ 0 & \textit{otherwise} \end{cases} \tag{3.3}$$

Assuming that the cost vector is $\mathbf{c} = [c_1 c_2 ...]^T$ and $\mathbf{1}$ is a row vector of 1's, the monitor insertion problem is formulated as follows:

$$\begin{aligned} \text{minimize:} \quad & \mathbf{c}^T \cdot \mathbf{x} \\ \text{subject to:} \quad & \mathbf{P} \cdot \mathbf{Q} \cdot \mathbf{x} \geq \mathbf{1} \\ & x_i \in \{0, 1\} \end{aligned} \tag{3.4}$$

3.4.3 Problem Conversion

Solving (3.4) directly is computationally intractable for any reasonable size of circuit because of the large number of paths. We will convert the problem into a solvable edge-based problem.

Since all entries in $\mathbf{P}$ and $\mathbf{Q}$ are either 0 or 1, entries in $\mathbf{P} \cdot \mathbf{Q}$ are non-negative integers and some entries may be larger than 1. This is not necessary for $\mathbf{P} \cdot \mathbf{Q} \cdot \mathbf{x} \geq \mathbf{1}$ given that $x_i \in \{0, 1\}$. We can derive a new matrix $\mathbf{A}$ with $a_{k,i}$ equals 1 if corresponding entry in $\mathbf{P} \cdot \mathbf{Q}$ is non-zero, and 0 otherwise. The new constraint $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{1}$ is equivalent to $\mathbf{P} \cdot \mathbf{Q} \cdot \mathbf{x} \geq \mathbf{1}$ for the optimization problem in (3.4). Therefore, we can replace $\mathbf{P} \cdot \mathbf{Q} \cdot \mathbf{x} \geq \mathbf{1}$ with $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{1}$ for (3.4).

Then we relax the constraint on x_i as $x_i \geq 0$. [1] This gives us an LP problem with its dual as:

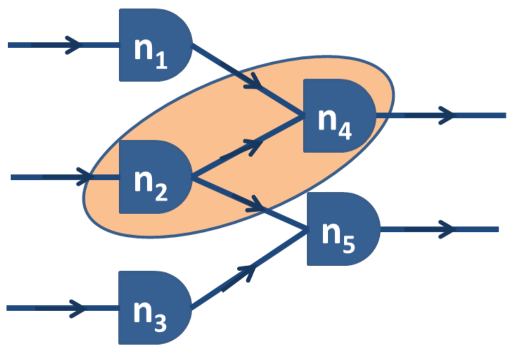
$$\begin{aligned} \text{maximize:} \quad & \mathbf{1}^T \cdot \mathbf{f} \\ \text{subject to:} \quad & \mathbf{A}^T \cdot \mathbf{f} \leq \mathbf{c} \\ & \mathbf{f} \geq 0 \end{aligned} \tag{3.5}$$

If we add two dummy nodes n_s and n_t as the beginning and ending nodes for all paths, (3.5) becomes similar to but not exactly the same as a maximum flow problem with $\mathbf{f}$ being the dedicated flow along each path.

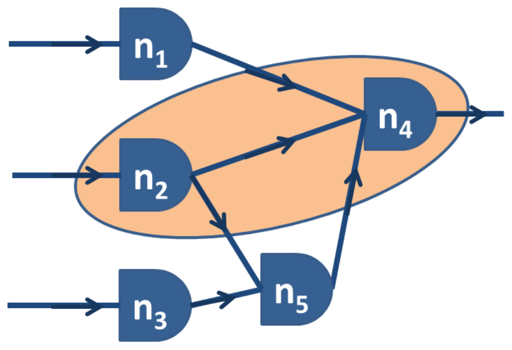
Converting (3.4) into the LP problem (3.5) still does not reduce the problem size. To make the problem solvable, we will convert (3.5) into a formulation with edge-based variables and constraints. We denote $\mathbf{e} = [\ldots\ e_{i,j}\ \ldots]^T$ as the total network flow on the edges. $e_{i,j}$ equals the sum of all path flow that goes from n_i to n_j and satisfies the flow conservation constraint. Since all path flow starts from the source node n_s, the objective in (3.5) is equivalent to maximizing the total flow out from n_s.

If we look at the i-th row of the constraint $\mathbf{A}^T \cdot \mathbf{f} \leq \mathbf{c}$ in (3.5), on the left-hand side is the sum of all path flows that pass through nodes in I_{n_i}. The key enabling observation is that I_{n_i} is defined with respect to the delay margin, which is monotonic in topological order. Therefore, all nodes in I_{n_i} are connected (see example in Fig. 3.5(a)). If we group I_{n_i} as one single pseudo node in the graph, there will be no cyclic flow paths that exit I_{n_i} and enter again (see example in Fig. 3.5(b)). All paths that pass through nodes in I_{n_i} will pass through one and only one of the edges that goes into I_{n_i}. In the example in Fig. 3.5(a), the path flows that pass through I_{n_4} are $\{n_1, n_4\}$, $\{n_2, n_4\}$ and $\{n_2, n_5\}$, of which the sum equals all the edge flow that goes into I_{n_4}, i.e., $e_{1,4} + e_{s,2}$. Therefore, we can replace the left part of the constraint with the sum of all incoming edge flow of pseudo node I_{n_i}.

[1] constraint $x_i \leq 1$ is not necessary in this case because monitor cost $\mathbf{c} > 0$ and $a_{k,i} \in \{0, 1\}$



(a) Treat I_{n_4} as a node



(b) Example of cyclic flow

Figure 3.5: Example of treating the nets in $I_{n_4} := \{n_2, n_4\}$ as a node. Cyclic flow as in (b) is impossible because margin at n_2 must be larger than margin at n_5. In (b), $n_2 \in I_{n_4}$ implies $n_5 \in I_{n_4}$.

The converted edge-based formulation becomes:

$$\begin{aligned} \text{maximize: } & \sum_{\forall i} e_{s,i} \\ \text{subject to: } & \forall n_i, \sum_{\forall j} e_{j,i} = \sum_{\forall k} e_{i,k} \\ & \forall n_i, \sum_{n_j \notin I_{n_i}, n_k \in I_{n_i}} e_{j,k} \leq c_{n_i} \\ & \mathbf{e} \geq 0 \end{aligned} \tag{3.6}$$

In this edge-based LP formulation, the number of variables equals m_E and the number of constraints equals $m_E + 2m_N$, where m_E is the total number of edges and m_N is the total number of nodes. For all our benchmarks, solving (3.6) takes less than a minute.

3.4.4 Problem Solution

Because of the relaxation on x_i, the result of (3.6) will be a lower bound of that in (3.4).

To derive an exact solution of the monitor locations, we take the solution of (3.6) and extract out the set of all nets n_i with the edge flow into I_{n_i} equals c_i. This set will be a valid solution which satisfies constraints in (3.4). Then we identify the highest cost net that is unnecessary to maintain the constraint and delete it. This process is repeated until no more nets can be deleted. Our experiments show that in most of the cases, we can get the result that is close to its lower bound.

3.5 Monitor Cost Metrics and Comparison Analysis

In this section, we discuss monitor cost metrics and perform a thorough comparison between *SlackProbe* and conventional approaches of monitoring at path endpoints only.

3.5.1 Monitor Power Cost

Since the monitors are inserted as ECO, there is no direct area overhead added to the original design, but the monitors will introduce additional power overhead. Different monitoring locations have different power overheads because circuit nets have different signal switching probabilities. The length of the matching delay chain will also affect the power consumption. Therefore, power overhead can be modeled as:

$$p_o + (\lambda_i p + l)d_i \tag{3.7}$$

where λ_i is the signal switching probability of net n_i, p is the estimated dynamic power overhead per unit matched delay, l is the estimated leakage power overhead per unit matched delay, d_i is the delay of the path that is going to be synthesized for the monitor at n_i, and p_o is the static power overhead of the transition detector, which includes additional clock power and leakage power. The derivation of λ_i will be discussed later in Section 3.5.4.

3.5.2 Layout Interference Cost

The monitor insertion is considered as ECO. ECO cost such as layout disturbance and timing disturbance should be considered. Layout disturbance can be evaluated by taking local layout congestion. To minimize the timing disturbance, the monitor uses a minimum size inverter to "probe" at the monitoring nets. The estimated timing slack after the inverter insertion can be used for timing disturbance evaluation. Similar to [Lee12], we use a linear model to evaluate the design perturbation cost of inserting a monitor at net n_i as:

$$a_t \cdot exp(-1 \times s_{n_i}) + a_r \cdot r_{n_i} \tag{3.8}$$

where s_{n_i} is the estimated timing slack after inserting the inverter, r_{n_i} is the layout congestion around n_i, a_t and a_r are weighting parameters for different cost considerations.

3.5.3 Implementation Flow Comparison

The proposed implementation flow of *SlackProbe* was described in Section 3.2.2 and Fig. 3.2, of which the starting point is a placed and routed design and the monitors are inserted as additional circuit through ECO, while conventional endpoint monitoring approaches, (e.g., [Foj13, Reb10]), decide the monitor locations at an earlier design phase and insert monitors by replacing the endpoint flip-flop cells.

Typically, conventional approaches determine the timing-critical flip-flops after synthesis and replace them with the monitor flip-flops. During later place and route, different sets of timing constraints are applied separately to regular flip-flops and the monitor flip-flops. The timing constraints are set to make sure that only paths ending at the monitor flip-flops can be timing critical under variations. This approach has relatively simpler physical implementation, but may unnecessarily speed up unmonitored paths and result in a sub-optimal design. While the approach as in Fig. 3.2 preserves the optimized design, but incurs more complicated physical implementation and design perturbation. Although both approaches can be applied to *SlackProbe*, the analysis and solution in this work is based on the approach as in Fig. 3.2.

3.5.4 Monitoring Efficiency and Observability

Other than the implementation flow, the monitoring efficiency and observability is another important difference between *SlackProbe* and conventional approaches. In situ monitor relies on timing-critical signal transitions along critical paths to monitor delay changes. Depending on the circuit structure and logic function, monitors inserted at different locations will encounter different sets of signal transitions and different number of timing-critical signal transitions.

As shown in Fig. 3.1, all signal transitions at the inserted net will trigger the monitor. But only timing-critical signal transitions can be used to detect impending timing failures. So the monitoring efficiency is determined by the proportion of signal transitions that are timing critical. Compared to conventional approaches, *SlackProbe* identifies and inserts monitors at more timing-critical parts of the circuit. It is expected to have a larger fraction of timing-critical transitions and thus have higher monitoring efficiency.

The other difference between *SlackProbe* and conventional approaches is the monitoring observability. Due to logic masking, signal transitions passing through intermediate nets may be masked by side inputs of the logic gates and cannot propagate to the path endpoint. In this case, the monitor inserted at intermediate net will detect the signal transition and use it to detect potential timing failure due to any instances along that path, while the monitor inserted at endpoint will not see the transition. For instance, in the example shown in Fig. 3.1, the signal at node A switches from logic low to logic high during the second clock cycle. This signal transition can be detected by the monitor inserted at node A. Due to logic masking, however, this signal transition cannot propagate to path endpoint at node C (shown as $T1$ in Fig. 3.1). Therefore, a monitor inserted at path endpoint node C will miss this event and cannot use it to detect impending delay failures. Since all transitions at the path endpoints imply transitions at the intermediate nets but not vice versa, *SlackProbe* can detect more signal transitions, thus is expected to have higher monitoring coverage than conventional approaches. In this work, we define the monitoring observability as the average number of monitored nets per cycle.

In situ monitor relies on timing-critical signal transitions to monitor the critical path delay. For system that uses in situ monitor as speed sensors, this may limit the capability of monitoring dynamic variations as non-deterministic lag exists between the time when dynamic variation happens and the time when variation is captured (as opposed to some replica monitors [B11, D13] which are used to monitor fast-varying variation). Therefore, higher monitoring observability means higher monitoring confidence and allows the system to monitor more fast-varying variation and achieve faster actuation response. For example, in the case of adaptive system, higher monitor confidence implies faster adaptation response (e.g., voltage/frequency adaptation) and less design margin for the reaction time. One exception is a razor-like system with recovery mechanisms as in [Foj13, Ern03] where the purpose of monitoring is to capture real timing failures. Higher observability means more recovery events even though there may not be real timing errors (i.e., false positive). In this case, the unnecessary recovery may increase the overhead.

Since different circuit nets will have different probabilities of seeing timing-critical transitions, monitor location selection will affect the monitoring efficiency and observability. In order to evaluate the monitoring observability, other than the signal switching probability λ_i, we need to extract the signal critical switching probability γ_i for each node n_i.

Timing-aware verilog simulation can be used to extract γ_i, but is much slower than a pure functional simulation. In this work, we make an approximation based on the properties of the reduced graph as described in Section 3.3.2. by assuming that a critical path/subpath is activated if there is an ordered sequence of switches for all nets along the path/subpath. To extract γ_i, we first run functional verilog simulation on the gate-level netlist and obtain the Value Change Dump (VCD) file. Then we apply our proposed algorithm (see Algorithm 14) for each simulated cycle and identify the sets of nodes in the reduced graph with regular switches and with timing-critical switches.

To obtain the switching probability λ_i and critical switching probability γ_i, we keep two

Algorithm 1 γ_i derivation

```
Input: VCD output for current simulation cycle
Output: $A[m_N]$ switching flags
 $B[m_N]$ critical switching flags
for all $n_i$ do
    unset $A[i]$ and $B[i]$
end for
for each $n_i$ in VCD that belongs to the reduced graph do
    set $A[i]$
    if $n_i$ has no fan-in then
        set $B[i]$
    else if $B[j]$ is set for any $n_i$'s fan-in node $n_j$ then
        set $B[i]$
    end if
end for
```

counters for each node to record the total number of regular switches and timing-critical switches. For each simulated cycle, we increment the counters if corresponding flag is set. The corresponding switching probabilities will be the counter output divided by the total number of simulated cycles.

3.5.5 Clock Matching

Another difference brought by inserting monitors at intermediate nets is monitored flip-flop clock matching. Since *SlackProbe* can insert monitors at intermediate nets, monitors may be used to monitor multiple flip-flops. Depending on their relative locations in the clock tree, these flip-flops may have different potential clock skew variations, which should be accounted for by monitor delay margin in *SlackProbe.* Conventional approaches that use one monitor per flip-flops do not have the same issue.

For *SlackProbe*, the monitors sharing same monitor should have small distance on the logic path graph. We expect their distance on the clock tree to be small as well. To evaluate and verify this, we include clock matching checking feature in *SlackProbe* implementation. For each of the monitors, we traverse downwards on the circuit graph and record the set of all downstream flip-flops and their common ancestor on the clock tree. In all experiments we ran, the maximum clock delay from the flip-flops' common ancestor to the flip-flops is at most 5% of the corresponding clock period. Therefore we expect clock matching to have little impact on *SlackProbe* delay margin.

3.5.6 Overall Monitor Insertion Cost Function

In [Lai13], a cost function c_i for each net n_i is defined, and combines the monitor power cost and design perturbation cost as:

$$c_i = p_o + (\lambda_i p + l)d_i + a_t \cdot exp(-1 \times s_i) + a_r \cdot r_{n_i} \tag{3.9}$$

where parameters are chosen for correct normalization. In this work, we derive a new cost function with monitor efficiency and observability awareness by normalizing the dynamic power cost with the critical switching probabilities γ_i as:

$$c_i = p_o + (\frac{\lambda_i}{\gamma_i}p + l)d_i + a_t \cdot exp(-1 \times s_i) + a_r \cdot r_{n_i} \tag{3.10}$$

where same parameter normalization is applied. In this cost function, power cost is calculated based on the power consumption per monitoring event, which represents the actual monitoring cost.

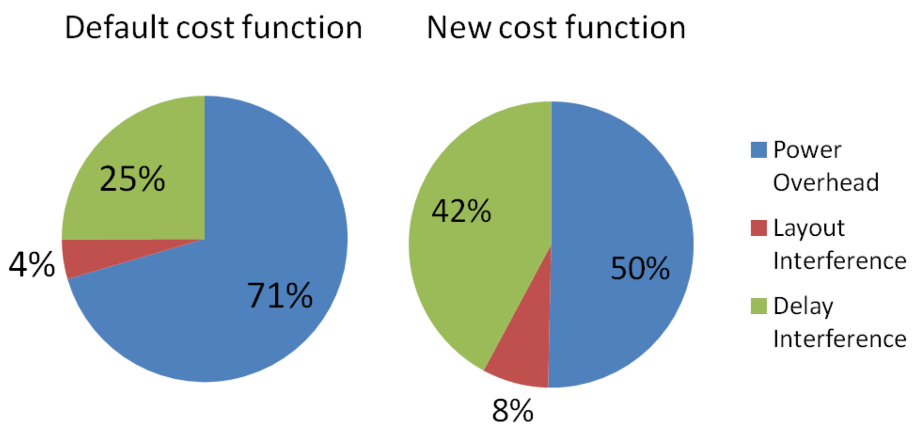


Figure 3.6: Monitor cost breakdown examples for a sample circuit node.

Monitor cost breakdown plots for a sample circuit node are shown in Fig. 3.6. The circuit node considered in Fig. 3.6 has $\lambda_i \approx 0.7$, $\gamma_i \approx 0.07$, $d_i \approx 0.1$ ns for both rising and falling transitions, $s_i \approx 0.05$ ns for both rising and falling transitions, neighborhood cell area utilization of about 0.8 and neighborhood routing utilization of 0.8. As discussed earlier, most but not all applications prefer higher monitoring observability. In the experiments, we will use the cost function in Equation (3.9) as our default cost function. The cost function in Equation (3.10) will be applied to show the improvement of *SlackProbe*'s monitoring efficiency and observability over conventional approaches.

3.6 Experimental Results

3.6.1 Experiment Setup

To evaluate the effectiveness of our monitoring methodology and problem solution, we use three commercial processor benchmarks and implement them using a commercial sub-32nm process technology and libraries. The timing libraries are all at 0.81 V. The implementation is done using Cadence toolchain [Cad]. The implementation information is listed in Table 3.1. Gate-level Verilog simulation and Algorithm 14 is used to obtain the net switch probabilities and the critical switching probabilities. In this work, we use Dhrystone [Wei84] benchmarks for all three processors. For the input/benchmark dependence, relevant analysis and discussion can be found in [LRK09].

Since different applications will have different requirements for the monitors, we have implemented three different monitor insertion methods:

- *Baseline:* This is the referenced baseline method which inserts a monitor at every critical path endpoint.
- *SlackProbe Event Detection:* This method aims at event detection. It allows the monitors to be placed at path intermediate nets. To ensure the detection of all switching events, it allows including additional nets in I_{n_i} only when they are connected to the monitor through inverter or buffer.
- *SlackProbe Speed Sensing:* This method aims at speed sensing. It allows the monitors to be placed at path intermediate nets and allows including nets in the fan-in cone regardless of the gate type.

3.6.2 Results on Different Processor Benchmarks

For this experiment, the path selection is done through the opportunism window approach. We define the typical operating clock period as the clock period reported by timing analysis with typical process corner libraries. For example, in the case of Processor A as shown in Table 3.1, the opportunism window size is the clock period difference between typical corner and slow corner, i.e., 0.22 ns. Delay margin ϵ is set to be 5% of the typical operating clock period. Table 3.2 summarizes the experimental results for the methods on different processor benchmarks. The total monitor cost are normalized with respect to the lower bound cost of *SlackProbe Speed Sensing*. In all three benchmarks, by allowing inserting monitors at path intermediate node and extra delay margin, the total number of monitors is reduced by almost an order of magnitude.

3.6.3 Impact of Opportunism Window

In order to evaluate the monitoring benefit/overhead trade-off, we pick processor A and apply different path selection criteria (i.e., different opportunism window sizes) to it.

Fig. 3.7 presents the results of different path selection criteria. For each case, the timing margin ϵ is kept at 5%. The paths are selected as critical if their delay at slow process corner falls into the corresponding opportunism window. The monitor count comparison of *SlackProbe* and baseline is shown in the log-scale plot in Fig. 3.8. Compared to the baseline method, our methods show on average 12X reduction in the number of monitors for event detection and 16X for speed sensing over all cases.

3.6.4 Results on Different Monitor Delay Margin

To show the trade-off between delay margin and monitor count, we also sweep the delay margin ϵ for processor A and plot the corresponding monitor count and monitor cost in Fig. 3.9. The path selection criteria are the same as that in Table 3.2 (i.e., defining the opportunism window with typical operating clock period obtained from typical process corner libraries). By allowing more timing margin, the number of monitors decreases for both

[2]The path endpoints include the circuit primary outputs as well. Some flip-flops use both D and scan-in pins multiplexers to select different data inputs. They are treated as different endpoints here.

Table 3.1: Implementation information of the processor benchmarks

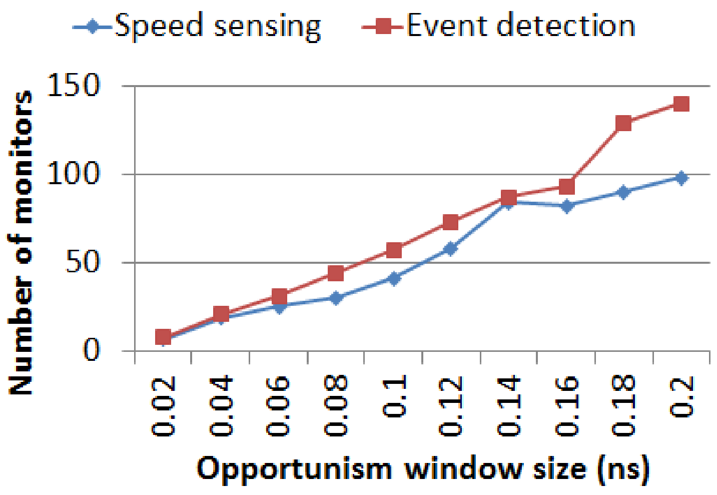
Processor	A	B	C
Gate count	10434	30296	58815
Register count	1191	3344	9516
Clock period at typical corner	1.01 ns	1.22 ns	1.43 ns
Clock period at slow corner	1.23 ns	1.48 ns	1.72 ns
Critical gate count	7246	21385	21630
Critical register count	852	2116	4195
Critical path endpoint count [2]	1256	2637	4993

Table 3.2: Experimental results on the processor benchmarks

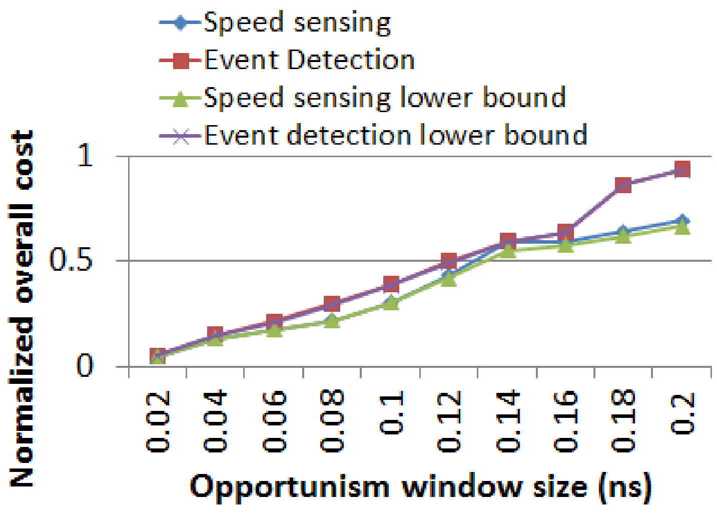
	Processor	A	B	C
Baseline	Monitor count	1256	2637	4993
	Normalized cost	12.16	8.95	14.05
SlackProbe Event Detection	Monitor count	148	480	510
	Normalized cost	1.34	1.69	1.57
	Normalized cost lower bound	1.34	1.59	1.53
SlackProbe Speed Sensing	Monitor count	113	311	374
	Normalized cost	1	1.07	1.08
	Normalized cost lower bound	1	1	1

methods. We also tried different weighting parameters of the monitor insertion cost (i.e., a_t, a_r etc.). Since monitor location selection depends more on the circuit topology, weighting parameters do not significantly change the total number of monitors. However, they do affect the monitor locations locally, especially for the speed sensing case, where different monitor location candidates can have similar critical path coverage but different monitor insertion cost.

In all experiments, our proposed solution achieves results equal or very close to the



(a) Monitor count



(b) Monitor cost

Figure 3.7: Monitor count and cost for processor A with different opportunism window size theoretical lower bound.

3.6.5 Extrapolation for Low Voltage Scenarios

In the experiments, timing libraries at 0.81 V is used. A lower supply voltage can potentially increase the total amount of variation. A sample circuit path delay at slow corner with

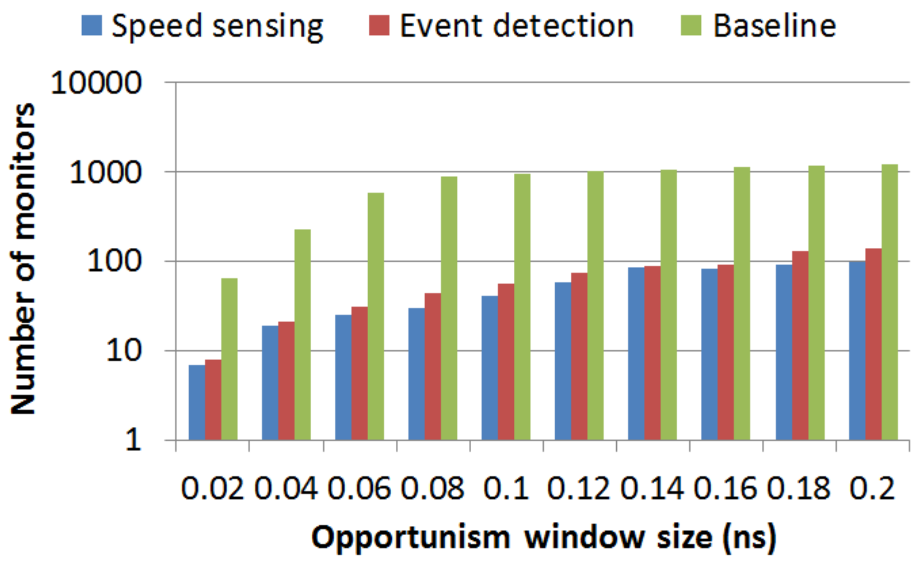
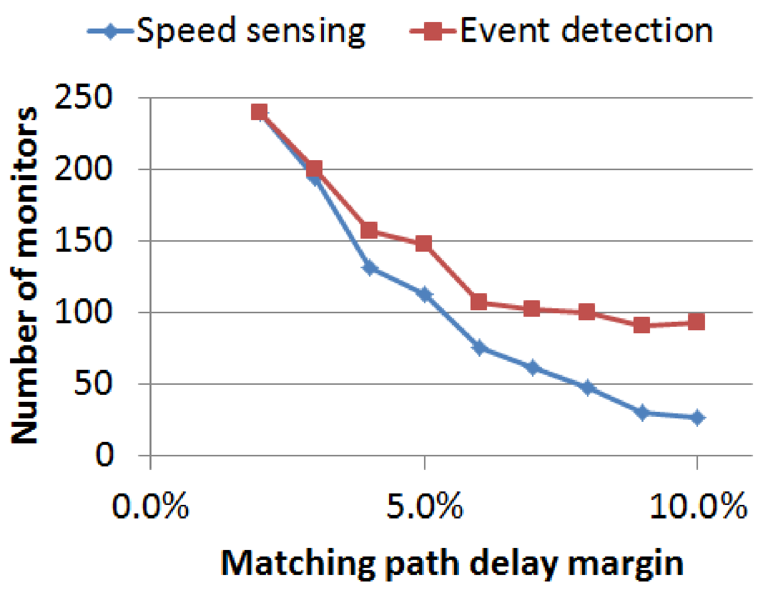


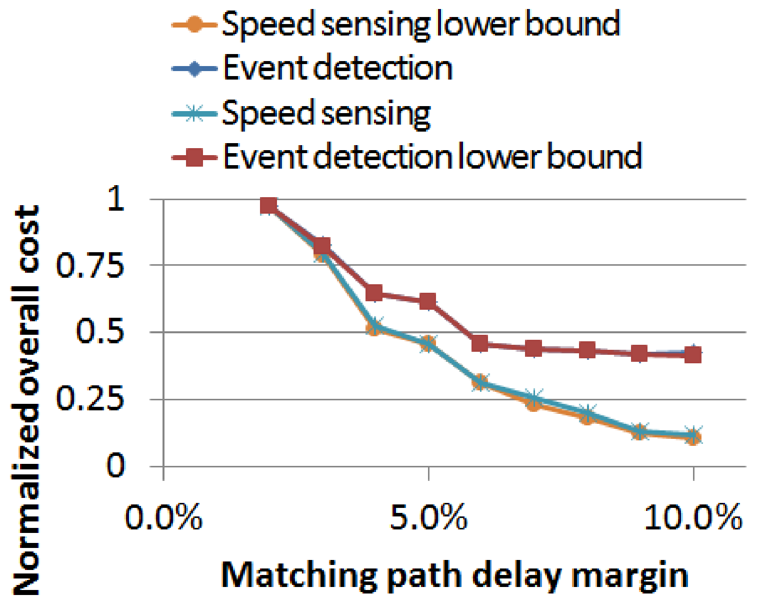
Figure 3.8: Monitor count comparison between *SlackProbe* and baseline. The y-axis is plotted in log scale.

different supply voltages is shown in Fig. 3.10. When the supply voltage decreases from 0.8V to 0.7V, the delay difference between typical corner and slow corner increases by about 2X (approximately from 25% to 50% when normalized with the delay at typical corner).

The magnified variation will scale up the worst-case design margin (see Fig. 3.3) by a similar amount. This has two impacts for *SlackProbe* under the same design setup. First, for the same set of monitored paths, the equivalent opportunism window size will increase with magnified variation, which will increase the potential monitoring benefit and makes the monitoring scheme more viable. Second, the same monitor candidate location will require more delay margin. Therefore, with the same amount of delay margin (ϵ), the set of monitor candidate locations will be less, which results in similar effect as reducing ϵ as discussed in Section 3.6.4 and in Fig. 3.9. This will increase the number of monitors and reduce the corresponding advantage of SlackProbe over conventional endpoint monitoring schemes.



(a) Monitor count



(b) Monitor cost

Figure 3.9: Monitor count and cost vs. delay margin for processor A

3.6.6 Monitoring Efficiency and Observability

In order to evaluate and compare the monitoring efficiency and observability of *SlackProbe* and conventional approaches, we pick processor A and simulate it with Dhrystone [Wei84]

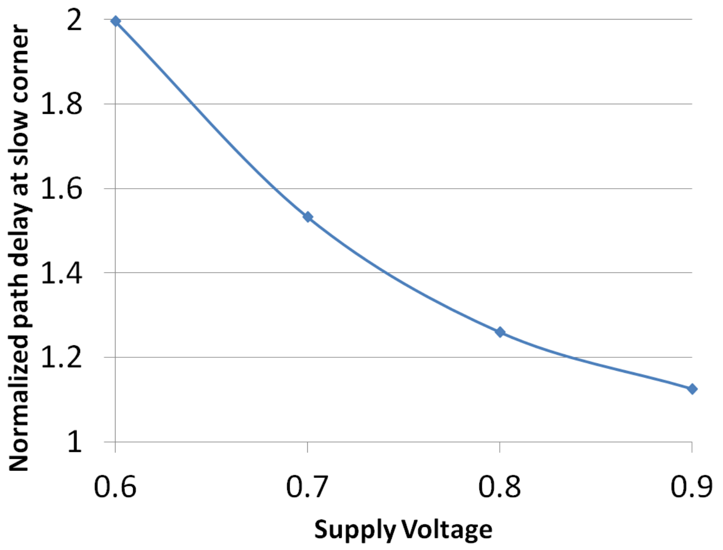


Figure 3.10: Delay of a circuit path at slow corner. All path delay numbers are normalized to the corresponding delay at typical corner. The variation has larger impact at lower supply voltage.

benchmarks. For each simulated cycle, we use the proposed method as in Algorithm 14 to identify the set of nets with signal transitions and the set of nets with timing-critical transitions. λ_i and γ_i are updated accordingly in order to evaluate the monitoring efficiency and to evaluate the results of different cost functions. A counter is used to record the number of nets whose delay is monitored in the current cycle (i.e., a timing-critical transition passes through this net and hits a monitor), which can be used to evaluate the monitoring observability.

Monitoring efficiency and observability results are listed in Table 3.3. The monitor types marked with "(opt)" are the ones using the new cost functions as in Equation (3.10).

Compared to baseline, *SlackProbe* using default cost function in Equation (3.9) selectively inserts monitors at nets that have lower λ_i (thus with lower power overhead). Over all cases, *SlackProbe* has up to 1.5X higher criticality ratio (i.e., $\frac{\gamma_i}{\lambda_i}$) than baseline. SlackProbe using new cost function in (3.10) favors inserting monitors at nets that have both lower λ_i and higher γ_i (thus with higher monitoring efficiency), which further boosts the criticality ratio to up to 1.9X higher than baseline. A larger delay margin (ϵ) will give more flexibility in selecting monitor location, which results in higher criticality ratio.

Because monitors inserted at intermediate nets cannot monitor the nets after the inserted location, *SlackProbe* without observability awareness can only monitor similar or less number of nets compared to baseline. However, with the new cost function, *SlackProbe* is able to monitor up to 32% more nets than baseline. Larger delay margin (ϵ) allows monitors to be inserted further away from path endpoints, which results in less monitored nets (thus lower monitoring observability). Event detection can detect more signal transitions than speed sensing, which makes it achieve higher monitoring observability.

3.6.7 Practical and Implementation Considerations

One major concern with in situ monitoring is the implementation feasibility and potential overhead and disturbance due to the additional instances and wiring. To explore the implementation overhead and find out possible implementation issues of the monitor insertion, we pick processor A and implement the complete monitor insertion on it.

The target application model is shown in Fig. 3.11. The system will start with a safe supply voltage (V_0) and always wait for certain time (t_a) before making the voltage adjustment of ΔV if no monitor flag is generated. In such system, the value of t_a and ΔV will depend on the monitoring observability and delay margin (ϵ). The monitors are designed to give a one bit sticky flag which can be reset externally. For implementation simplicity, we pick the monitor structure that consists of only standard cells as shown in Fig. 3.12.[4]

Monitor flag signals are connected through an OR tree and gives the one bit flag signal as processor primary output. Therefore, for each monitor, there are at least six gates in it

[3]If monitor is inserted at path endpoint, no extra delay margin is required for delay matching.

[4]This monitor is used only for evaluation purpose. It may not work for signal transitions with small glitches. A proper monitor design (e.g., TDTB [Bow09]) can avoid the glitch issues.

Table 3.3: Monitoring efficiency and observability results

Monitor type	Delay margin (ϵ)	Monitor count	Average λ_i	Average γ_i	Criticality ratio $\frac{\gamma_i}{\lambda_i}$	Average monitored nets per cycle
Baseline	N.A.[3]	1256	0.746	0.068	9.11%	317.4
Speed sensing	5%	113	0.311	0.042	13.61%	295.7
Speed sensing(opt)	5%	147	0.443	0.089	20.20%	420.8
Event detection	5%	148	0.303	0.048	15.88%	314.1
Event detection(opt)	5%	216	0.383	0.073	19.02%	376.5
Speed sensing	8%	48	0.329	0.076	22.99%	284.4
Speed sensing(opt)	8%	45	0.568	0.151	26.52%	357.1
Event detection	8%	100	0.459	0.089	19.28%	317.6
Event detection(opt)	8%	100	0.558	0.124	22.21%	360.2

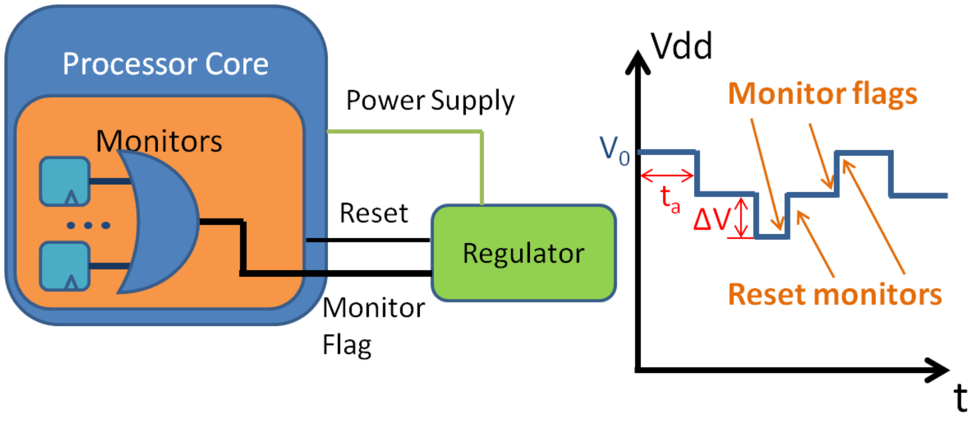


Figure 3.11: Example application: the monitors give a one bit flag to the voltage regulator, the regulator reset the monitors after a corresponding voltage adaptation operation

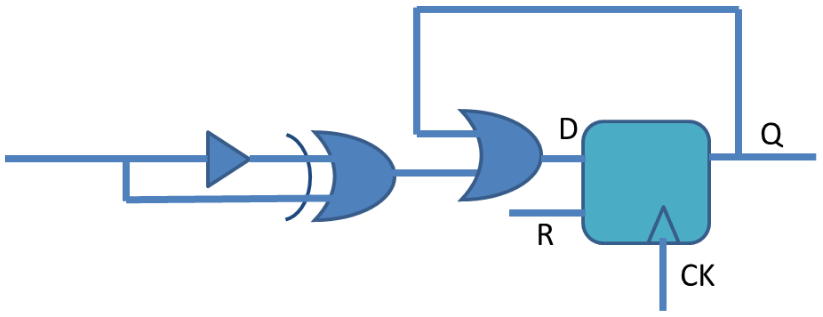


Figure 3.12: Implemented monitor structure: the flag is sticky with an external reset

including the first minimum size inverter, four cells as in Fig. 3.12 and one OR gate in the OR tree.

In this experiment, we implement two versions of the monitor insertion. The implementation results are summarized in Table 3.4. The final layout of implementation II is shown in Fig. 3.14. The overhead may be further reduced by using simpler or customized monitors as in [Das10, Bow09, Kim13].

During monitor insertion, because of the additional load from the monitors and other ECO timing disturbance, the original design is slowed down by about 5%. However, this only happens around the selected nets, which can be recovered through simple optimization like

Table 3.4: Implementation results on processor A

	Implementation I	Implementation II
Target delay margin	5%	8%
Number of monitors	113	48
Additional instances	711	327
Instances per monitor	6.3	6.8
Additional power overhead	13.65%	6.38%

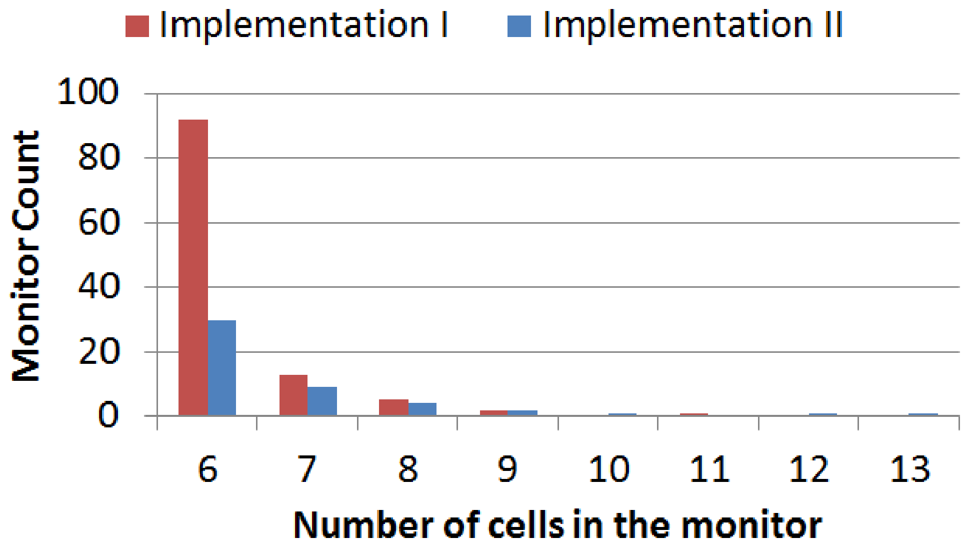


Figure 3.13: Monitor cell count distribution: most monitors are inserted very close to the path endpoints, thus do not need extra cells other than the minimum required 6 cells.

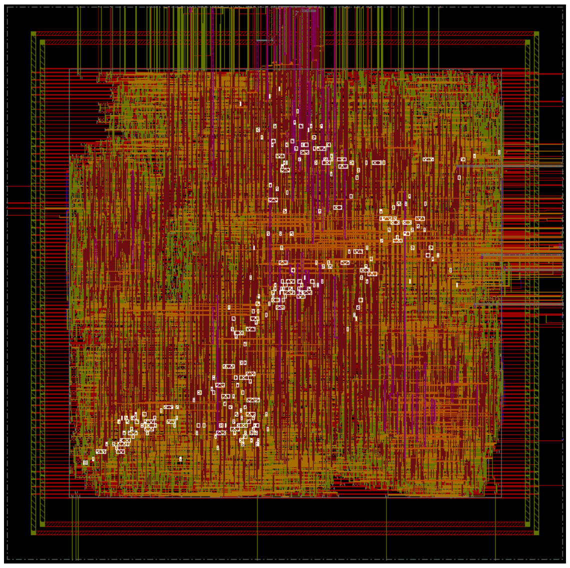
incremental sizing and threshold voltage assignment. In the experiments, we incrementally optimize the design after the monitor insertion so that it still meets the same delay target.

The ECO also affects the clock network. We perform detailed analysis on the clock network before and after ECO for Implementation II. After ECO, the total number of clock sinks increases by 48, which is the number of monitor inserted. The number of clock tree levels remains at 3, but the number of clock buffers increases from 25 to 29. Maximum local clock skew slightly increases from 11.9 ps to 14.9 ps.

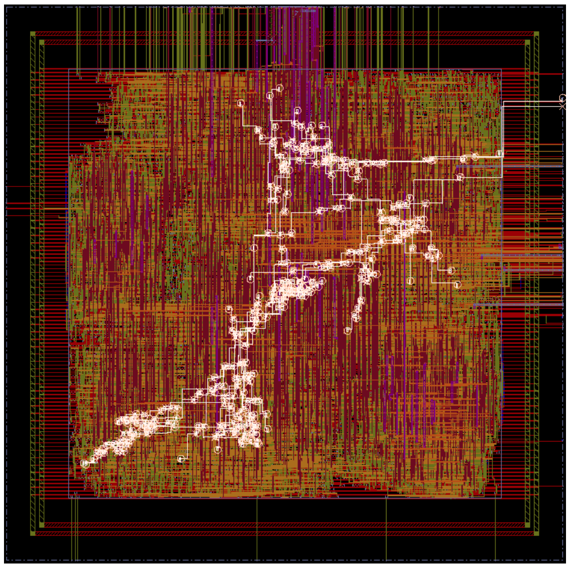
The other side effect of the ECO timing disturbance is that some new nets become timing critical. This may introduce unmonitored critical paths, which will require additional delay margin, pessimism in path selection or further timing optimization. The slow-down due to additional load from the monitors will only affect the nets that are monitored already. However, other timing changes such as ECO routing and clock skew changes may introduce unmonitored critical paths. In this implementation, we found two unmonitored critical paths due to the clock skew changes. A more careful monitor selection and less intrusive insertion can help prevent it.

3.7 *SlackProbe* Conclusions

In this chapter, we have proposed *SlackProbe*, a novel timing slack monitoring methodology of inserting monitors at both path ending nets and path intermediate nets. Experimental results on commercial processors show that with 5% additional timing margin, *SlackProbe* can reduce the number of monitors by 12-16X as compared to the number of monitors inserted at path ending pins. *SlackProbe* can also improve the monitoring efficiency by up to 1.9X and improve the monitoring observability by up to 32%, as compared to approaches that monitor path endpoints only.



(a) Layout with monitor instances highlighted



(b) Layout with both monitor instances and wires highlighted

Figure 3.14: Final layout of processor A with the inserted monitors placed and routed

CHAPTER 4

System-level Dynamic Variation Margining in Presence of Monitoring and Actuation

4.1 Chapter Introduction

Dynamic variations affect the circuit performance and threaten the system resilience. Voltage fluctuations, temperature variations and circuit aging are typical dynamic variation sources that can degrade the circuit performance and cause timing errors. To guarantee correct functionality, increasing amount of margin is applied to account for dynamic variations.

Monitor-and-actuate has been widely adopted as an approach to dynamically mitigate circuit variability and reliability issues [Gup12, R09, B11, Aga, L14a, L13]. Most such systems consist of three components, i.e., monitor, actuator and controller. The variation signatures, including voltage/temperature fluctuations and wear-out aging, are captured by the monitor at runtime. A controller with certain type of management policy uses the monitor readings and makes the adaptation decision. Actuator, which is the mechanism for achieving adaptation, adjusts the operating conditions as the system's reaction to the measured variations.

Previous work has been proposed to improve the quality of the monitors or the actuators. For example, various types of circuit-level monitors are proposed [C14, L14a, L14b, Tsc09, Dra07] with different levels of accuracy, overhead, and measurement latency. Different actuation mechanisms, e.g., clock gating [B11], dynamic frequency scaling (DVFS) [D13, L13], software-driven DVFS [Aga, L14a], task migration [Cue11] are used with different levels of granularity, overhead, and actuation latency. Some of the monitor and actuator design quantifies the margin required for its own inaccuracy [C14, L14a, L14b]. However, for dynamic variations, both inaccuracy and latency in the entire adaptation process has to be accounted for. System-level dynamic variation margining in presence of monitoring and actuation is still understudied and ad-hoc.

In this work, we study the system-level margining problem assuming a simple threshold-triggering policy. Some sophisticated controller policies can reduce the system reaction latency with some prediction capabilities, e.g., through signature-based [R09] or model-based [Cos08] approaches. However, they comes with additional prediction overhead and more importantly imperfect prediction accuracy. Therefore, they may not be able to reduce the system margin, which is typically worst-case driven.

The rest of the chapter is organized as follows: Section 4.2 describes the system margining strategy. Section 4.3 presents the study of system margining for different types of dynamic variation sources. Section 4.3.3 discusses the design implication of dynamic variation margining. Section 4.4 concludes the chapter.

4.2 System Margining Strategy

In this section, we study the margining strategy for dynamic variations in presence of adaptation systems. The system margin breakdown is described in Section 4.2.1. The exact amount of margin required is analyzed in Section 4.2.2.

4.2.1 Margin Breakdown

Ideally, no margin is required if the system can adapt to the variations perfectly. Every source of imperfections during the monitor-and-actuate process, including monitor and actuator inaccuracy, adaptation latency etc., calls for margin. For dynamic variations, system margin can be classified as static margin and dynamic margin. Static margin is the additional design guardband required due to limited monitor coverage/accuracy and actuation granularity. Dynamic margin is the additional design guardband required due to monitor measurement and actuation latency.

As mentioned in Section 4.1, we assume a threshold-triggering policy as illustrated in Fig. 4.1. For certain variation metric (e.g., temperature, voltage droop, delay etc.), the system has a maximum allowed value (marked as the red line in Fig. 4.1) to ensure correct operations. A threshold-triggering policy has a trigger threshold value (dotted line in Fig. 4.1) for the monitor reading, above which the controller will activate the actuation process. Due to various latencies in the entire monitor and actuation process, by the time when actuation becomes effective, the variation metric may have gone worse from what was monitored (blue dotted line in Fig. 4.1). This calls for dynamic margin, which will be a function of the system response latency and variation changing rate. Other than dynamic margin for response latency, certain static margin is required for monitor and actuator inaccuracy. As shown in Fig. 4.1, the total system margin, including both static margin and dynamic margin, is dictated by the difference between the maximum allowed value and the trigger threshold.

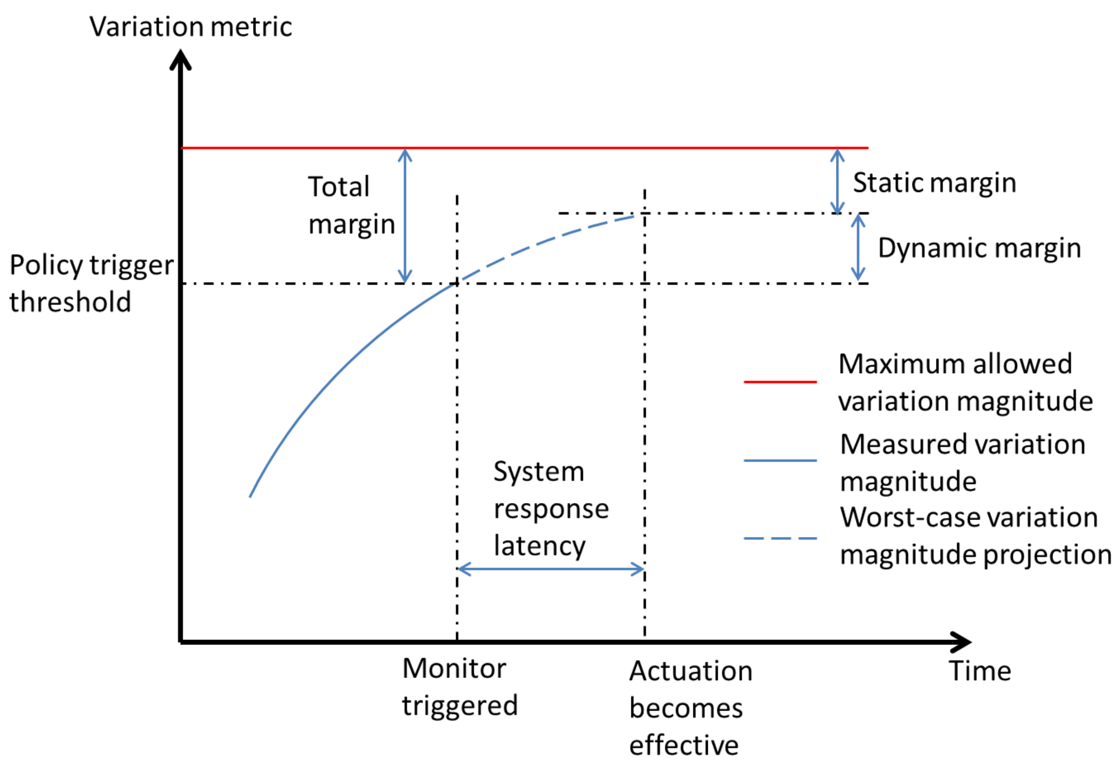
4.2.2 System Margin Analysis

As discussed earlier, the system margin can be classified as static margin and dynamic margin. Static margin is always required, regardless of variation types. Improvement on either monitor accuracy or actuator quality can reduce the required static margin. For example, flexible monitor design methodologies [C14, L14a, L14b] can trade-off its own accuracy with overhead. The static margin M_s can be calculated as:

$$M_s = Err_m + Err_a$$

where Err_m and Err_a are the monitor and actuator errors.

The amount of dynamic margin, however, will depend on the temporal profile of the



Variation metric
Total margin
Static margin
Dynamic margin
Policy trigger threshold
Maximum allowed variation magnitude
Measured variation magnitude
Worst-case variation magnitude projection
System response latency
Monitor triggered
Actuation becomes effective
Time

Figure 4.1: Illustrate of the threshold-triggering policy

specific variation type (i.e., how fast the variation changes) and both the monitor and actuator latencies. For example, process variation or circuit aging (e.g., BTI, HCI etc.) requires little dynamic margin because they are either static or slow changing. But temperature and supply voltage fluctuations will need certain amount of dynamic margin, as the temperature increase or voltage droop can be worsening before the system can monitor and effectively react to them. So based on the variation type, the margin required can be represented as a function $V(t)$ of the latency t. The derivation of this function will be discussed later in Section 4.3. So the dynamic margin M_d can be calculated as

$$M_d = V(l_m + l_t)$$

where l_m and l_d are the latencies for monitor and actuator.

So the total system margin M can be calculated as:

$$M = M_s + M_d = Err_m + Err_a + V(l_m + l_t)$$

4.3 Case Study Experiments

In this section, we studies the system margining problem for various types of dynamic variation sources. The types of dynamic variation sources will be described in Section 4.3.1. Case studies for temperature variations are presented in Section 4.3.2.

4.3.1 Dynamic Variation Sources

In this work, we consider mainly three types of dynamic variation sources, i.e., voltage droops, wear-out aging and temperature variations. Out of these three variation sources, voltage droops is the fastest varying one, and wear-out aging is the slowest. So we expect the dynamic margin to be the largest for voltage droop and smallest for aging.

The behavior of voltage droop depends on both the workload behavior and power delivery network. Some power-management mechanisms like power gating can introduce a sudden step in current and exacerbate the voltage droops. Previous work reports that the step-on current can results in a peak voltage drop resonating at about 100MHz [G07]. In order to make the adaptation system effective, the entire monitor-and-actuate system should operate much faster than 100MHz. Even if we consider clock-gating as the fastest actuator for voltage droop, it is extremely difficult to design the adaption system with short enough latency. Clock gating is a synchronous operation, so the latency for gating the clock can be up to 1 clock cycle. Considering the growing complexity of clock network and massive number of clock sinks, the clock distribution latency from clock source to the end clock pins can be up to a few clock cycles [Res04, Res01]. Therefore, we expect little benefit from dynamic adaptation for voltage droops.

Most of wear-out aging mechanisms, unlike voltage droop, varies slowly along the entire hardware lifetime. Given the typical hardware lifetime of a few years, the monitor and actuator latencies are negligible. The system margin for wear-out aging will be dominated by the static margin. Therefore, in this work, we focus on the case of temperature variations.

4.3.2 Case Studies for Temperature Variations

We demonstrate our strategy with some case study of thermal monitors and corresponding actuators. We assume the threshold-trigger policy in the experiments. For the system to function correctly, a bound is defined for the variation, i.e., maximum allowed temperature. The system will keep reading from the monitor and activate the actuator if the monitor reading is beyond certain threshold. Due to the imperfection of the monitor and actuator, the threshold has to be conservative and tighter than the variation bound. The margin is represented as the difference between the threshold and the bound. For temperature monitoring, there can be two sources of inaccuracy. The monitor itself can be inaccurate and deviate from the actual junction temperature. Meanwhile, the thermal hotspot may not be at the same exact location as the thermal monitor, causing spatial inaccuracy. Therefore, the static margin can be classified as monitor inaccuracy margin and spatial inaccuracy margin.

The problem of margining for temperature fluctuation is illustrated in Figure 4.1. For the experiments, we set the temperature limit as 100 $°C$. By using thermal simulator, HotSpot [Hua06], we can simulate the temperature profile of a given power trace. In this work, we consider a chip with 16X16 homogeneous cores. The baseline power, i.e., Thermal Design Power (TDP), is set so that the highest on-chip temperature reaches the temperature limit 100 $°C$. Different thermal monitors and actuators are summarized in Table 4.2 and Table 4.1, respectively. To be able to cross-compare different type of temperature monitors, the area budget for monitor placement is fixed at 0.4 mm^2. The monitor inaccuracy due to

Table 4.1: List of Actuators

Actuator	Latency
A1: Clock gating	10 ns
A2: DFS(with PLL relock)	100 us
A3: DVFS	10 ms
A4: Task migration	50 ms

limited number of thermal monitors are calculated using the equation [Mem08]:

$$Spatial\ Error = 25 - 4log_2N$$

, where N is the number of monitors.

The dynamic margin is derived based on the maximum power consumption P_{max}. If $P_{max} = TDP$, the problem is not interesting, as the maximum temperature will never be higher than the temperature limit. Some case study results are highlighted in Table 4.3. A few interesting observations can be made from the results:

- Dynamic margin is comparable to or even larger than static margin, depending on P_{max}/TDP ratio.
- The amount of dynamic margin increases dramatically with increased P_{max}/TDP ratio. As we enter the dark silicon era, where power density increases much faster than the packaging technology.
- Within the static margin, spatial error margin is considerably higher than the inaccuracy margin. This can be different if we allocate more resources and increase the number of thermal monitors.

Comparing some cases with the same types of actuator, e.g., Case I and Case IV the same monitor area with different monitor types can result in dramatically different amount of static margin and dynamic margin. Inappropriate combination of monitor and actuator

Table 4.2: List of Thermal Monitors

Monitor	samples/s	Accuracy	Normalized Area (mm^2) at 65nm
M1 [Chu11]	10000	4 $°C$	0.01
M2 [Woo09]	5000	2 $°C$	0.04
M3 [Che05]	1000	1 $°C$	0.01
M4 [Ait09]	10	0.1 $°C$	0.04

types, e.g., unmatched latencies, can results in up to 2.8X increase in system margin from 7.71$°C$ of Case I to 21.33$°C$ of Case VI.

4.3.3 System Design Implication

When designing adaptation systems, both monitors and actuators need to be matched to maximize the benefit. However, some parts of the system, especially the actuators, may also be used for other purposes like power management. The choices of them can be affected by many different considerations.

This work aims at giving some guidelines for selecting and designing the adaptation system, especially for the monitors which usually are dedicated for this purpose and have larger design flexibility. A few design guidelines can be made based on our observations from the case studies for temperature variations:

- Monitor inaccuracy, which is the main metric for monitor design, only constitute a fraction of the entire system margin. Over-optimizing the monitoring accuracy at the cost of excessive area and latency can sometimes result in larger system margin.

- Monitor latency, which may be overlooked during monitor design, can affects the dynamic margin. This makes the design of faster monitor as important as design accurate ones.

- For spatially distributed phenomena like temperature variations, optimizing for smaller

Table 4.3: Case Study on Dynamic and Static Margin

Case	Monitor	Actuator	Total latency	Static Margin (monitor+spatial)	Dynamic Margin ($P = 1.25 * TDP$)	Dynamic Margin ($P = 1.5 * TDP$)	Dynamic Margin ($P = 1.75 * TDP$)
Case I	M1	A1	100 us	$4\ °C + 3.71\ °C$	$0\ °C$	$0\ °C$	$0\ °C$
Case II	M2	A2	0.6 ms	$2\ °C + 11.7\ °C$	$0\ °C$	$0.02\ °C$	$0.05\ °C$
Case III	M3	A3	11 ms	$1\ °C + 3.71\ °C$	$0.11\ °C$	$0.41\ °C$	$0.90\ °C$
Case IV	M4	A4	150 ms	$0.1\ °C + 11.7\ °C$	$1.28\ °C$	$7.07\ °C$	$22.5\ °C$
Case V	M1	A4	50 ms	$4\ °C + 3.71\ °C$	$0.48\ °C$	$1.86\ °C$	$4.13\ °C$
Case VI	M4	A1	100 ms	$0.1\ °C + 11.7\ °C$	$1.01\ °C$	$4.16\ °C$	$9.53\ °C$

monitor area/overhead is also important from the system design perspective. In some cases, trading-off individual monitor accuracy for more efficient monitor design can result in smaller spatial margin and thus better system margin.

We understand that in real design, the choices of adaptation system can be affected by many different considerations. The final decision can be made by trading-off all metrics, e.g., final design margin, design overhead, and design complexity etc.

4.4 Chapter Conclusion

In this work, we first study the system margining problem and analyze different types of margin for monitor-and-actuate systems. Margining strategies are studied for different types of dynamic variation sources. Case studies are performed for temperature variations margining. The experiment results show that incorrect design and selection of monitor and actuator can result in less adaptation benefit. Design guidelines are given for optimizing system margin of monitor-and-actuate adaptations.

CHAPTER 5

Hardware Reliability Margining for the Dark Silicon Era

5.1 Chapter Introduction

With ending of perfect Dennard scaling, integrated circuits have entered the “dark silicon” era [Esm11], where chips are fundamentally power- and thermal-constrained. Multi-core design has been widely adopted as the approach to both improve energy efficiency and reduce power density by operating at lower power state and exploiting the workload parallelism [Isc06, Kum03]. Meanwhile, single-thread performance is still desired by the workload that is not parallelizable. Techniques such as “turbo boost” [Cha09b] are used to temporarily operate one or several processor cores at higher voltage and frequency but still under certain power and thermal constraints. This creates a large dynamic range in performance and power that the hardware is designed to run at but maybe only for part of the circuits or only for a short amount of time. For example, Intel i7-920XM [int, wik] has a base frequency of 2 GHz. It can be boosted to operate at maximum 2.26 GHz with four active cores and to 3.2 GHz with only one core active.

Conventionally, hardware reliability margin is derived from the worst-case aging scenario with possibly pessimistic or pathological assumptions. Although sense-and-adapt approach can be used to reduce parametric (e.g., voltage, delay) margin at runtime [Gup12], physical (e.g., wire width or spacing) margin must be determined during hardware design time. Since most circuit wear-out mechanisms (e.g., Negative- and Positive-Bias Temperature Instability (N/PBTI), Hot-Carrier-Injection (HCI), Electromigration (EM) etc.) degrade faster at higher voltage and temperature, the margin is typically derived when the circuits are operating at peak performance state with the highest operating voltage, power and temperature.

The dark silicon contexts impose constraints on power and temperature, so that it is impossible for all circuits to operate at peak performance state throughout the entire lifetime. Therefore, the reliability margin derived by the conventional approach can be overly pessimistic and is expected to be larger with technology scaling and increasing silicon "darkness". Different terms have been used to describe this "darkness", such as dark silicon fraction [Esm11] and transistor utilization wall [Ven10]. In this work, we define a term d called "dark ratio". The definition of d for power-constrained scenario is described in Equation 5.1, where $d = 0$ means all circuits can be fully powered up. The definition of d for thermal-constrained scenario will be described later in Equation 5.9

$$d = 1 - \frac{\text{maximum allowed power}}{\text{maximum possible power}} \tag{5.1}$$

The over-pessimism can potentially be reduced by correctly accounting for the dark silicon contexts. The main challenge is how to transform the high-level power/thermal (i.e., spatial) constraints into hardware aging (i.e., temporal) profile, especially with mixture of complicated reliability models and system power/thermal management schemes. In this work, we propose a hardware reliability margining methodology (see Fig. 5.1). It uses an accumulation model to represent the aging degradation dependence on different circuit power states. An optimization problem is formulated to search in the space of possible system workload (i.e., power states of each processor cores) patterns that will stress the circuits under the power/thermal constraints. By exploiting the properties of workload and management policy, this spatial distribution workload can be transformed into temporal aging profile, which in conjunction with the accumulation model can be used by the optimization to determine the worst-case aging degradation under the power/thermal constraints. The contribution of our work are:

- To the best of our knowledge, this is the first work on reliability margining considering the high-level power/ thermal constraints imposed by the dark silicon contexts.
- We formulate and solve the margining problem for both power- and thermal-constrained cases. Experimental results show that at 60% dark ratio, conventional margining approach can results in 3-7X and 18% overestimate of aging degradation due to EM and BTI respectively. Our margining method is able to eliminate these over-pessimism and results in about 20% delay margin and 40%-60% metal width margin reduction.

The rest of the chapter is organized as follows: Section 5.2 introduces some background of EM and BTI modeling and margining. Section 5.3 describes how to derive the required data and model for our methodology. Section 5.4 and Section 5.5 present the problem formulation for reliability margining under power constraints and thermal constraints, respectively. Experimental results and some discussions are presented in Section 5.6, and Section 5.7 concludes the chapter.

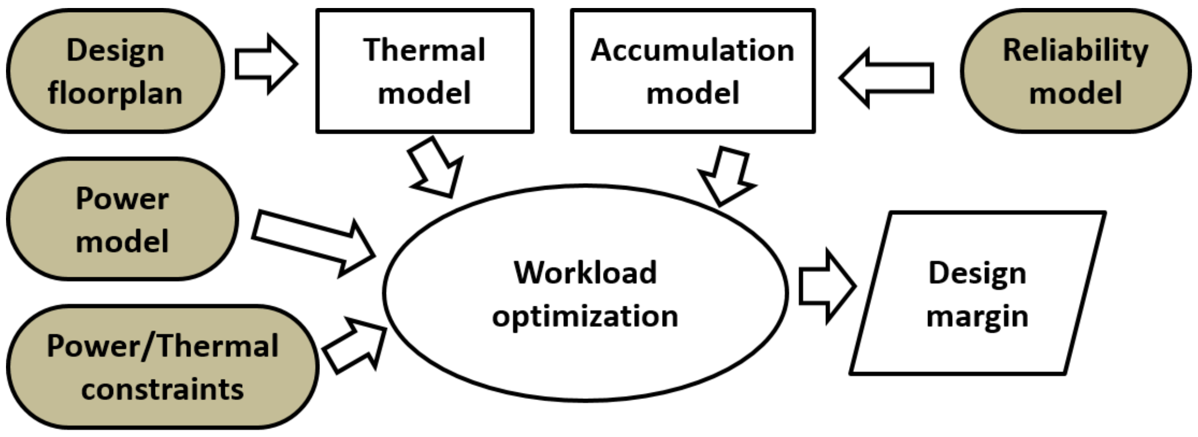


Figure 5.1: Margining Methodology Overview.

5.2 Chapter Background

Various reliability-related phenomena can result in circuit performance degradation or failure. They may require different types of design margin. In this work, we will focus on two reliability phenomena, EM and BTI. The basic modeling for EM and BTI are described in Section 5.2.1 and 5.2.2. Design margining for EM and BTI is introduced in Section 5.2.3.

5.2.1 EM Modeling

Electromigration [Hun97,Ban01] is a wear-out mechanism on interconnects. Void are created on metal wires due to momentum transfer from the electrons moving in the wires. The Mean Time to Failure (MTTF) of EM can be modeled by the famous Black's equation [Bla69]:

$$MTTF = A_0 J^{-2} e^{E_m/kT} \tag{5.2}$$

where A_0 is a constant, J is the current density, E_m is the activation energy for the EM mechanism, k is Boltzmann's constant, and T is the temperature.

For non-DC current, Hunter [Hun97] derived an "average" current density model for arbitrary unipolar current. This model is useful for interconnects with unipolar current, such as power distribution networks. An effective current density can be derived with Equation 5.3.

$$J_{eff} = \frac{1}{T}\int_0^T J(t)dt \tag{5.3}$$

where J_{eff} is the effective current density for EM purpose, T is the period, and $J(t)$ is the current density at time t.

Banerjee etc. [Ban01] derived the model with a recovery factor R for symmetric bipolar (AC) current. So the effective current density J_{eff} is R times the absolute current density. This model is useful for evaluating EM on signal wires, because in digital circuits, signal wires are used to charging up or charging down the same loading capacitance. So the average current density will be proportional to the supply voltage, loading capacitance and switching frequency. This dependence can be derived as Equation 5.4:

$$J_{eff} = R \cdot \frac{\int |J(t)|dt}{T} = R \cdot \frac{\mu C_{load} V f}{w \cdot h} \tag{5.4}$$

where μ is the signal switching probability, C_{load} is the loading capacitance, V is the supply voltage, f is the operating frequency, w is the metal width, and h is the metal height.

5.2.2 BTI Modeling

Bias Temperature Instability is a wear-out mechanism related to the transistor. Depending on the transistor type, BTI effects include NBTI for PMOS and PBTI for NMOS. Due to BTI, the device threshold voltage increases when the transistor is turned on (i.e., positively or negatively stressed). When the stress voltage is removed, part of the degradation can be recovered.

The exact BTI mechanism and model is still under debate within the reliability community. There are two major BTI models, reaction-diffusion (RD) model [Bha06, Wan07] and trapping/detrapping (TD) model [Vel12]. However, most existing analytical BTI models are derived for static operating conditions and may not be directly used for dynamic operating conditions introduced by typical system-level power management mechanisms such as DVFS and power gating. BTI simulators [Cha11] have been used to simulate the degradation under arbitrary stress profile.

5.2.3 Margining for EM and BTI

Unlike typical design optimization, hardware margining must be pessimistic to account for all possible or even pathological worst-case scenarios. In many cases, pessimistic assumptions are unavoidable and necessary. In this work, our approach will strictly follow this rule. The pessimism in the assumptions and approaches will be summarized in Section 5.6.4.

As suggested by Black's equation (see Equation 5.2), EM margining can be done by increasing the interconnect width in order to reduce the current density. Typically, certain design guidelines are given by the foundry as part of the reliability design rules. For example, in a commercial 45nm design manual, the reliability design rules are given as the current-dependent minimum width requirement for each metal layer. The amount of EM margin can also affect other design metrics such as performance, power and area [Kah13].

In a circuit design, there are different types of interconnect that can be affected by

EM. In this work, we consider the EM effects on both local power distribution network for one processor core and the signal wires inside the processor core. We omit the global power distribution network because the margining can be directly derived from the power constraints (i.e., dark ratio and total power).

BTI-induced threshold degradation can affect both logic and memory and result in reduced read/write margin [Kum06] and increased logic delay. Since SRAM operating conditions are usually more stable, in this work, we focus on the BTI-induced delay degradation for logic.

There can be two approaches to margin for BTI-induced delay degradation. One way is to estimate the end-of-lifetime delay degradation for the entire circuit and directly applying it as additional delay margin (i.e., reducing the operating frequency) or voltage margin (i.e., increasing the supply voltage). The other way is to estimate the process corners for the aged devices and performing timing sign-off accordingly (i.e., increase the design efforts such as larger cells). In this work, we will use the former approach to evaluate the BTI margin.

5.3 Power, Thermal and Accumulation Models

An overview of our proposed approach is shown in Fig. 5.1. The shaded blocks are assumed to be known at hardware design time. In this section, we will focus on the modeling of power, thermal and reliability that is required by the workload optimization step. Unless otherwise defined, all notations used in Section 5.3, 5.4 and 5.5 are listed in Table 5.1. Matrix transpose is represented with a prime symbol, i.e., transpose of $\mathbf{x}$ is represented as $\mathbf{x}'$.

In this section, our accumulation model approach will be described in Section 5.3.1. The power and thermal model used in this work is presented in Section 5.3.2.

5.3.1 Accumulation Model

Typical reliability models are derived for static scenarios. However, a dynamic reliability

Table 5.1: Glossary of Terminology

Term	Description
N	Total number of processor cores
n	Index for processor cores
M	Total number of power states
m	Index for power state
P^{max}	Maximum allowed power
T^{max}	Maximum allowed temperature
T^{amb}	Ambient temperature
d	Dark ratio
$\mathbf{x}$	$M \times 1$ vector where x_m is the number of cores at power state m
$\mathbf{P}$	$M \times 1$ vector where P_m is the power consumption of a processor core at power state m
$\mathbf{F}$	$M \times 1$ vector where F_m is the operating frequency of a processor core at power state m
$\mathbf{V}$	$M \times 1$ vector where F_m is the supply voltage of a processor core at power state m
$\mathbf{T}$	$N \times 1$ vector where T_n is the temperature of processor core n
$\mathbf{T}^{bak}$	$N \times 1$ vector where T_n^{bak} is the background temperature of processor core n
$\mathbf{A}$	$N \times N$ matrix where $A(n_1, n_2)$ is the temperature increase of core n_1 due to power consumed by core n_2
$\mathbf{S}$	$N \times M$ matrix where $S(n, m)$ is the proportion of time that core n spends in power state m
$f(\mathbf{x})$	The accumulation model for a processor core spending x_m/M lifetime at each power state m

model is required to derive the worst-case aging scenarios, which can be a mixture of different power states. For example, with the same power budget, the circuits can be operating at 0.8V constantly or alternating between 0.7V and 0.9V. We need to be able to compare these different workload patterns in order to determine the worst case.

Most reliability-related phenomena are temperature-dependent. Intuitively, lower power states are usually associated with lower temperature. So we may be able to use lower temperature for calculating the degradation. But this is not necessarily true considering the following scenario that the processor core has already been heated up by other heavy workload before entering a lower power state. This history dependence makes it difficult to make any assumption about the temperature that is guaranteed to be pessimistic. Therefore, in this work, we always assume the degradation happens under the maximum temperature.

In our proposed methodology, reliability model is used to characterize an accumulation model (see Fig. 5.1). This helps isolate the model-dependence and makes our methodology more general for different wear-out mechanisms. The accumulation model is a function to calculate the pessimistic estimation of end-of-life degradation under a given workload pattern. The input to the accumulation model is the fraction of circuit lifetime spent in different power states (i.e., $\mathbf{x}$). The accumulation model is represented as the function$f(\mathbf{x})$.

Some reliability models themselves suggest the corresponding accumulation model. For example, the current density for local power distribution network is proportional to the power consumption. Therefore the EM accumulation model can be derived based on Equation 5.3 as:

$$f(\mathbf{x}) \propto \mathbf{x}' \cdot \mathbf{P} \tag{5.5}$$

The EM accumulation model for signal wires can be derived based on Equation 5.4 as:

$$f(\mathbf{x}) \propto \mathbf{x}' \cdot (\mathbf{F} \circ \mathbf{V}) \tag{5.6}$$

where $\circ$ represents Hadamard product (i.e., element-wise multiplication).

For other reliability phenomena like BTI, the analytical models may not be directly applied for dynamic cases. Using the model incorrectly can lead to misleading results [Cha11]. In this work, we will derive the accumulation model with a physics-based RD simulator [Cha11].

We use two steps to derive the accumulation model. The first step is to identify the ordering of different power states that will result in the worst degradation. Based on the simulator, the worst BTI degradation happens when power states are applied in increasing order of stress voltages. Therefore, for a given power state distribution ($\mathbf{x}$), there is only one degradation value and it is pessimistic.

The second step is obtaining the accumulation function through interpolation or fitting. In this work, we first pick a set of power state distribution samples $\mathbf{x}$. These samples are generated with fraction of lifetime spent in each power state being multiple of 0.1. To avoid the low activity region where the BTI model shows high non-linearity, we limit the total fraction of idle time to be less than 0.3. Therefore, we will only report the results with $d < 0.7$.

Assuming $g(\mathbf{x})$ is the accumulation function by the simulator, our accumulation function is fitted with a linear function, i.e., $f(\mathbf{x}) = \mathbf{c}' \cdot \mathbf{x} + d$ with $\mathbf{c}$ and d being the fitting coefficients, as the results of the following optimization:

$$\begin{aligned} \text{minimize:} \quad & b \\ \text{subject to:} \quad & \forall \mathbf{x} \;\; 0 \leq f(\mathbf{x}) - g(\mathbf{x}) \leq b \end{aligned} \tag{5.7}$$

We validate this accumulation function against the results from the simulator. The maximum overestimation is less than 0.5mV for about 10mV threshold voltage degradation.

5.3.2 Power and Thermal Model

For chips in the dark silicon era, there can be two types of constraints, i.e., limit on the instantaneous power consumed by the circuits and limit on the highest temperature on-chip. For example, Intel Sandy Bridge [Rot12] has total package power control and responsiveness via dynamic turbo. The former one controls the entire package's total power consumption, while the latter one allows the total power consumption to be temporarily higher (e.g., 1.2-1.3X [Rot12]) than the thermal design power (TDP).

Since the margin is determined at hardware design time, detailed power models may not be available. But the hardware designer should have estimated power for each hardware power states. In this work, we assume that the power consumption at each power state is available for reliability margining purpose. The model should include the power consumption for all possible power states, including different voltage states (i.e., P-state) and idle states (i.e., C-state). These power consumption estimate should be optimistic (i.e., with smaller values) so that the margin will be pessimistic.

The thermal model should give the static state temperature under a given power profile for a fixed floorplan and package. In this work, we will use a thermal simulator, HotSpot [Ska03, Hua06], which takes the floorplan and power profile trace and can calculate both static state and transient temperature. By using the simulator, we can obtain the static state temperature increase of one processor core due to power consumed by the other cores (i.e, **A**). Based on the thermal-electrical duality, the superposition principle can be applied here. Therefore, the temperature can be computed with the values in **A** without invoking the simulation in the optimization. We have also validated this superposition principle and the thermal model using the simulator.

5.4 Reliability Margining with Power Constraints

In this section, we describe the problem formulation for system with power constraints. The interpretation of management policy and power constraints are presented in Section 5.4.1. The optimization formulation is described in Section 5.4.2.

5.4.1 Management Policy and Power Constraints

For circuits in the dark silicon era, certain management mechanisms and policies are essential to enforce the power/thermal constraints. These policies can affect the system behavior and thus the reliability margin. In some case, a bad policy can result in the same degradation as what conventional reliability approach indicates. For example, a policy with a fixed priority in scheduling workload can overload the first core, resulting in spending entire lifetime at the highest power state.

In this work, we assume a fair round-robin policy that iterates the scheduling priority between all processor cores. The priority iterating frequency can range from once per hour to once per week. We argue that this is a reasonable, effective, and possibly pessimistic policy. This policy is an open-loop policy that does not rely on any sensing or monitoring capabilities, and thus is easy to implement. Given a typical hardware life time of multiple years, the number of iteration is sufficient to redress the workload imbalance between different cores. More sophisticated policies, e.g., close-loop sense-and-adapt policies or more advanced management policies [Kar09, Cal09, Sri05], may improve the degradation. This makes our assumed policy pessimistic, which is acceptable for margining purposes.

With such management policy, the workload and the corresponding power states will be iterated among all processor cores. So the power states of all processor cores at a given time will be iterated through each processor core through its lifetime. Therefore, the temporal distribution of power states for one processor core is equivalent to the spatial distribution of power states among all cores. This property will be exploited in our problem formulation.

5.4.2 Problem Formulation

The power constraint limits the total power of all processor cores. In the power model, processor power is represented as its power state. So the total power consumption can be calculated from the number of cores in each power state (i.e., $\mathbf{x}$ should be integer variables).

As discussed earlier, this spatial distribution can be converted into the temporal distribution of power states along the lifetime of one processor core. So the optimization should maximize the resulting degradation with power state distribution of $\mathbf{x}$. Therefore, the problem with power constraints can be formulated as the integer optimization problem described in Equation 5.8.

$$\begin{aligned} \text{maximize:} \quad & f(\mathbf{x}) \\ \text{subject to:} \quad & x_m \geq 0 \\ & \mathbf{x}' \cdot \mathbf{1} \leq N \\ & \mathbf{P}' \cdot \mathbf{x} \leq P^{max} \end{aligned} \tag{5.8}$$

5.5 Reliability Margining with Thermal Constraints

In this section, we describe the problem formulation for system with thermal constraints. The interpretation of thermal constraints is presented in Section 5.5.1. The optimization formulation is described in Section 5.5.2.

5.5.1 Interpreting the Thermal Constraints

Unlike power constraints, thermal constraints do not directly impact the system instantaneous behavior. Circuits can temporarily operate at high power state if the temperature is below the limit. Intuitively, there can be two possible pathological scenarios that may maximize the degradation. The first one stresses the circuits in a two-phase manner which cools down the circuits with lower power state and then heats up with high power state. The second one stresses the circuits in a constant manner so that the temperature is maintained or has small fluctuations right below the temperature limit.

We argue that the second scenario will always result in worse degradation through an example shown in Fig. 5.2. Both cases have the same alternating power states (i.e., power state I and II), but with different alternating frequencies. Since the average temperature of the second case is higher than the first case, by the thermal model, the heat dissipated from the second case will be more than the first case. Therefore, the second case can spend more time in higher power state and thus results in more wear-out degradation.

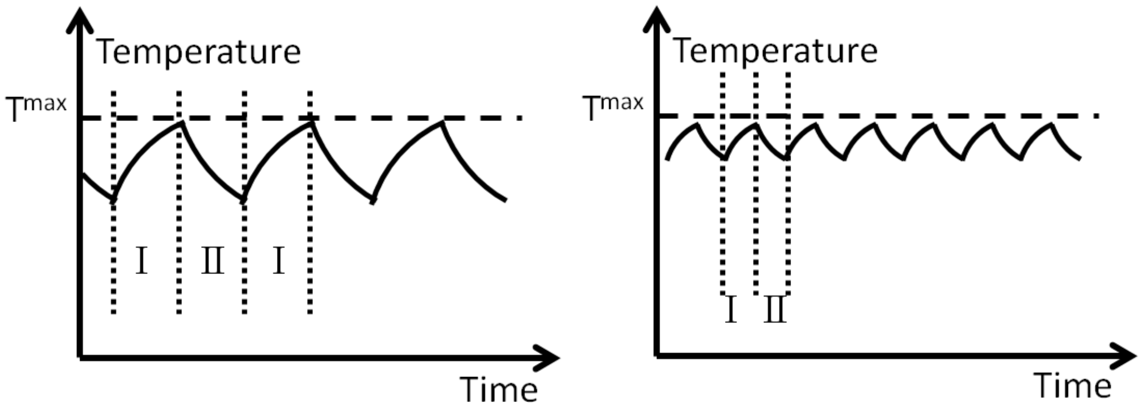


Figure 5.2: Example of two workload cases. Both cases are alternating their power states between I and II while keeping the temperature under the thermal limit.

So with the thermal constraints, a "constant" stress will result in more degradation. We also define the dark ratio d for this thermal-constrained scenario in Equation 5.9. The term $T^{max} - T^{amb}$ is the maximum temperature increase allowed for a given ambient temperature. The term $max(\mathbf{A} \cdot \mathbf{1} \cdot max(\mathbf{P}))$ is the temperature increase when all processor cores are fully powered up.

$$d = 1 - \frac{T^{max} - T^{amb}}{max(\mathbf{A} \cdot \mathbf{1} \cdot max(\mathbf{P}))} \tag{5.9}$$

5.5.2 Formulation

As discussed earlier, the thermal constraints can be represented as the limit on the average power, which can be represented as the fraction of time spent in each power states for all processor cores (i.e., continuous variables **S**). The static temperature is calculated based on this average power consumption and the pre-characterized temperature-sensitivity matrix **A**. The problem with thermal constraints can be formulated as an optimization problem described in Equation 5.10.

$$\begin{aligned} \text{maximize:} \quad & f(\mathbf{S}' \cdot \mathbf{1}) \\ \text{subject to:} \quad & \forall m, n \quad S(m,n) \geq 0 \\ & \mathbf{S} \cdot \mathbf{1} \leq \mathbf{1} \\ & \mathbf{A} \cdot \mathbf{S} \cdot \mathbf{1} \leq T^{max} \times \mathbf{1} - \mathbf{T}^{bak} \end{aligned} \tag{5.10}$$

5.6 Experimental Results and Discussion

We will demonstrate our proposed architecture-assisted reliability margining methodology for both power- and thermal-constrained scenarios. The experiment setup is described in Section 5.6.1. The results for EM margining and BTI margining are presented in Section 5.6.2 and 5.6.3 respectively. Some of the results and the pessimism in our approaches are discussed in Section 5.6.4.

5.6.1 Experiment Setup

The power model is based on a commercial processor benchmark and a commercial sub-32nm process technology and libraries. The processor benchmark has 32KB L1 instruction cache, 32KB L1 data cache and 256KB L2 cache. The power values are derived from the synthesized, placed and routed results of the processor benchmark. Libraries are characterized at different supply voltages to calculate the operating frequency and power. Five different voltage states are used with supply voltage ranging from 0.6V to 0.9V. So the total number of available states is six including the state when the core is shut down or power gated. The SRAM power is derived from the memory compiler results.

The thermal model (i.e., the values in **A**) is pre-characterized using HotSpot [Ska03, Hua06]. The chip floorplan size is configured as 15mm x 15mm. The ambient temperature is set as 300 K. Other parameters are set the same as default HotSpot settings. Accumulation model is derived using the BTI simulator [Cha11]. The BTI simulator results are scaled to match the built-in NBTI and PBTI models in the process library. We report the BTI degradation (i.e., ΔVth) as the sum of NBTI and PBTI degradation (i.e., $\Delta Vthn + \Delta Vthp$).

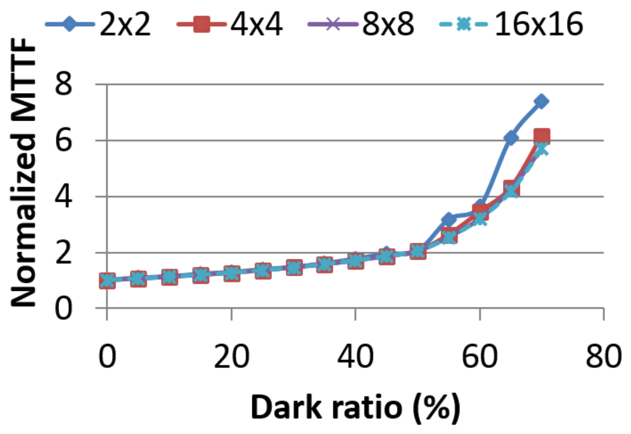
In the experiments, we will compare against conventional margining approaches, which derives the margin assuming the core is operating at its highest power state along the entire lifetime (i.e., equivalent to the cases when $d = 0$). In order to test the applicability and scalability of our proposed approaches, we consider the case of 4, 16, 64 and 256 cores which are placed in 2x2, 4x4, 8x8 and 16x16 arrays respectively. The power/thermal constraints are determined by the dark ratio d defined in Equation 5.1 and 5.9.

5.6.2 EM Results

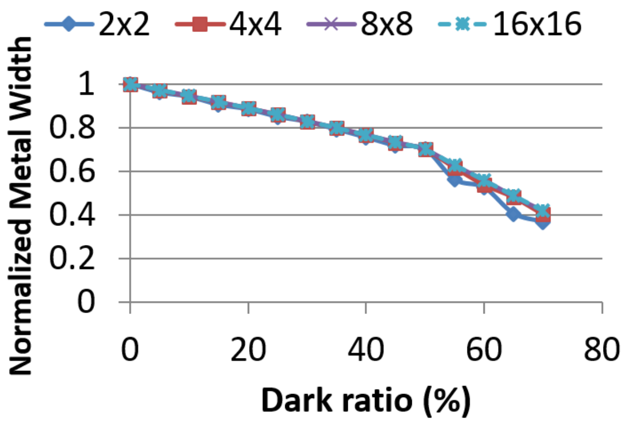
As mentioned earlier in Section 5.2.1, we will apply our margining methodology for both local power distribution network and signal wires. The accumulation function for unipolar and bipolar current is applied as discussed in Section 5.3.1. For both the power-constrained and thermal-constrained scenarios, we obtain the value of P^{max} and T^{max} by varying the value of dark ratio d. The optimization results of Equation 5.8 and 5.10 are reported and normalized with respect to the conventional margining approaches (i.e., the case when $d = 0$).

The results of local power distribution network under power constraints are plotted in Fig. 5.3. Compared to our margining approach, conventional approach gives about 3X underestimate of the MTTF at 60% dark ratio, which is equivalent to about unnecessary 40% metal width margin. The results of signal wires under power constraints are plotted in Fig. 5.4. The MTTF underestimate by conventional margining approach is about 7X, which is equivalent to 60% redundant metal width margin.

We also record the worst-case power state results generated by our methodology. For local power distribution network, the worst-case happens when some of the cores are at the highest power state and other cores are idle. For signal wires, the worst-case happens when all cores are active and at some intermediate power states. This also suggests different preferences in power management schemes for EM purposes.

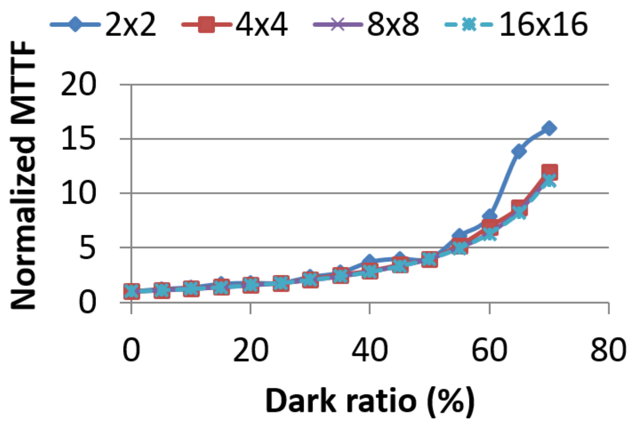


(a) MTTF

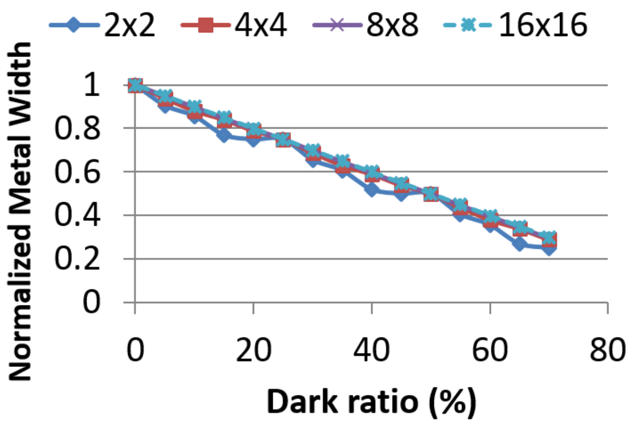


(b) Metal Width

Figure 5.3: EM results with power constraints for local power distribution network.



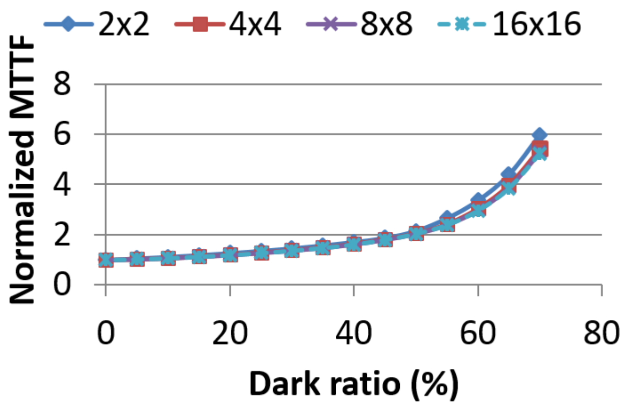
(a) MTTF



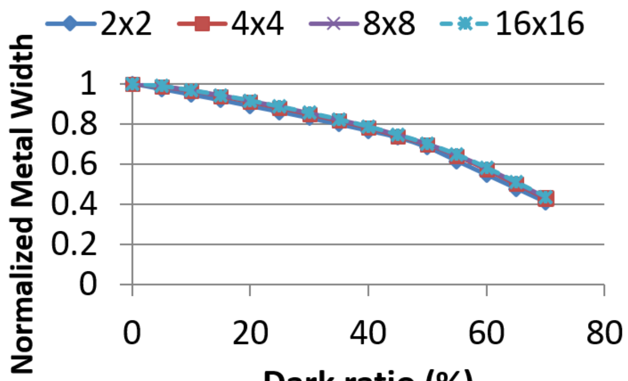
(b) Metal Width

Figure 5.4: EM results with power constraints for signal wires.

The results of same scenarios under thermal constraints are plotted in Fig. 5.5 and Fig. 5.6. Compared to the results under power constraints, the curves are smoother. This is because the power-constrained problem is an integer problem, while the thermal-constrained problem is continuous. The over-pessimism of conventional margining approach is similar to the cases with power constraints. At 60% dark ratio, our approaches can results in 3-6X less MTTF overestimation and 40%-60% metal width margin reduction.

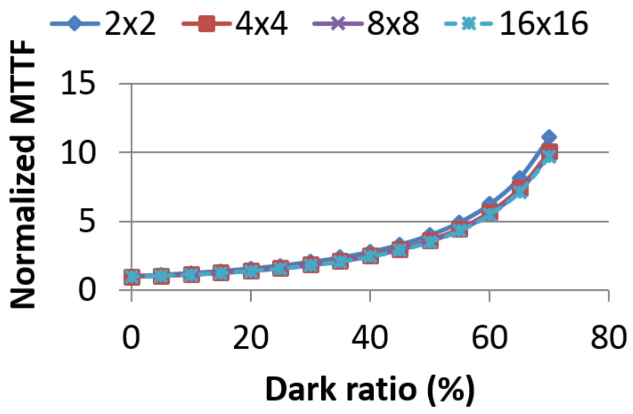


(a) MTTF

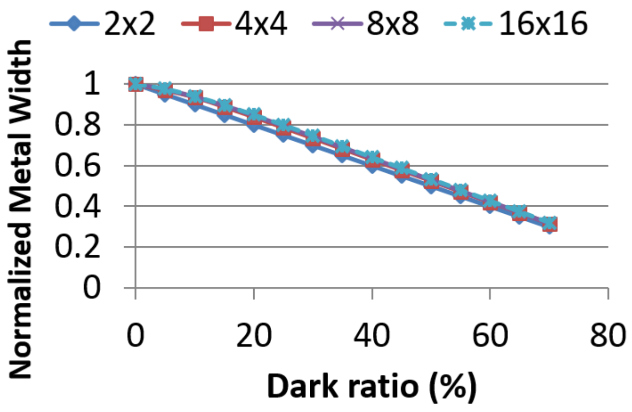


(b) Metal Width

Figure 5.5: EM results with thermal constraints for local power network.



(a) MTTF



(b) Metal Width

Figure 5.6: EM results with thermal constraints for signal wires.

5.6.3 BTI Results

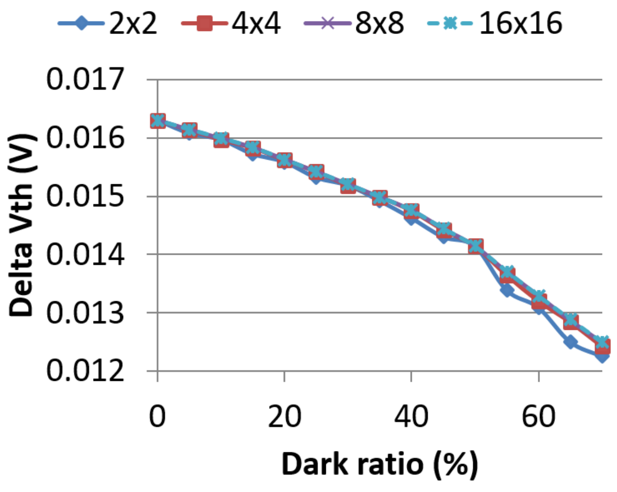
We also apply our methodology for BTI margining. A linear accumulation function is fitted by using the RD simulator as described in Section 5.3.1. The results for Vth degradation are plotted in Fig. 5.7. At 60% dark ratio, conventional margining approaches result in about 3mV overestimate of the threshold voltage degradation, which is about 18% of the total degradation. We also apply this degradation difference on a buffer chain and use SPICE to study the delay impacts. The around 16 mV threshold degradation requires up to 14% margin on the total delay at the lowest supply voltage (i.e., 0.6V). At 60% dark ratio, our margining approach is able to reduce these delay margin values by about 20%, i.e., about 3% total delay margin reduction. The margin reduction can imply even more saving in the final design if we consider the interaction between aging margin and design flow [Cha13c].

Among all the results, the number of processor cores does not significantly affect the results. The only exception is the power-constrained cases with high dark ratio values. This is caused by the discrete power states and integer variables in power-constrained problem.

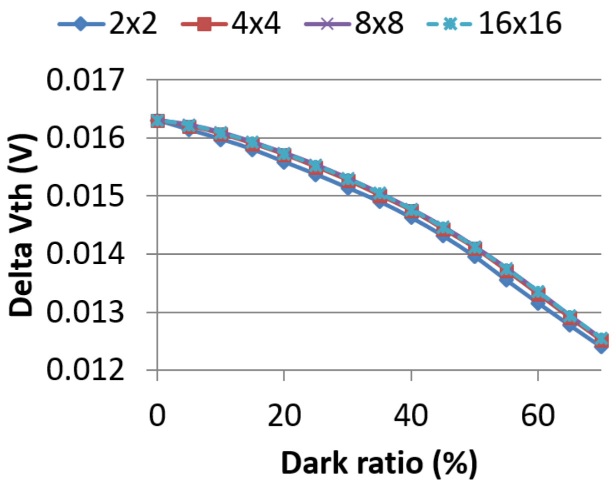
5.6.4 Pessimism in Our Approach

Due to the nature of design margining, the approach has to guarantee its pessimism against all possible pathological cases. Throughout our margining methodology, possible pessimism is introduced. We summarize and discuss some of them here.

- As discussed in Section 5.3.1, due to the history effects of temperature, we assume the worst-case temperature for all power states.
- As discussed in Section 5.3.1, the accumulation model, if derived from the simulator, can results in pessimism. For example, the BTI accumulation model always assumes the worst-case ordering of power states.
- As discussed in Section 5.3.2, the power and thermal models used in this approach



(a) Power-Constrained



(b) Thermal-Constrained

Figure 5.7: BTI results of Vth degradation with power constraints and thermal constraints.

are pessimistic, e.g., the use of best-case power values under process, temperature and workload variations. This will result in optimistic estimation of the number of active cores and therefore result in pessimistic estimation of the margin.

5.7 Chapter Conclusion

In this work, we propose an architecture-assisted hardware reliability margining methodology for chips in the dark silicon era. By exploiting the properties of workload and system management policies, we formulate the margining as an optimization problem. Experimental results show that at 60% dark ratio, our method can reduce the over-pessimism in conventional margining approach for EM and BTI by 3-7X and 18% respectively, which is equivalent to 40%-60% reduction in metal width margin and 20% reduction in delay margin. Our ongoing work is looking at the margining problem for heterogeneous multi-core systems.

CHAPTER 6

BTI-Gater Aging Resilient Clock Gating Methodology

6.1 *BTI-Gater* Introduction

Negative- and Positive-Biased Temperature Instability (N/PBTI) have become a major reliability concern for modern semiconductor technology [Sch03, Hic08]. NBTI manifests itself as an increase in PMOS threshold voltage V_{th} when PMOS is negatively biased, while PBTI manifests as an increase in NMOS V_{th} if NMOS is positively biased. When the bias voltages are removed, the devices enter a recovery phase, where part of the degradation can be recovered.

Due to this stress-recovery behavior, N/PBTI-induced degradation depends heavily on the workload. Mintarno et. al. [Min13] report that N/PBTI-induced timing degradation on data path varies from 2% to 11%, depending on the workload. Compared to data path, clock distribution network usually has balanced structure and invariant signal pattern, which makes it more robust against N/PBTI-induced degradation. But this may not hold for clock distribution networks with clock gating features [Liu12]. Conventionally, clock signal on the gated branch is frozen at one state (i.e., logic low or logic high), which will introduce imbalanced degradation and cause additional clock skew.

Previous work [Cha13a, Hua13, Che13, Cha09a] has attempted to address this issue. However, most of existing work did not consider the clock gating control signal generation and the synchronous nature of the clock gating operations, which may lead to incorrect assumptions and circuit malfunctions. We will review each of them in Chapter 6.6.

In this work, we first analyze the effects of N/PBTI on clock distribution network and N/PBTI-induced clock skew under different clock gating use cases. Then cross-layer solutions are proposed to address this issue of N/PBTI-induced clock skew. Since the clock skew is mainly due to the imbalanced clock signal duty ratio (D), two ICG cell circuits are proposed to alternate clock idle state between logic high and logic low for each clock gating operation, so that the clock signal duty ratio (D) can be balanced to close to 50%. A skew mitigation methodology is also proposed to apply the appropriate ICG cells for different design, based on the architecture and micro-architecture context. To avoid certain pathological aging scenarios, an example of sleep scheduling is also described as a simple software-level technique that can be used in conjunction with *BTI-Gater*.

The rest of this chapter is organized as follows: Chapter 6.2 introduces some background information and discusses the effects of N/PBTI on clock distribution network. Chapter 6.3 analyzes the N/PBTI-induced clock skew for clock distribution network with clock gating. Chapter 6.4 presents our cross-layer solutions, including two *BTI-Gater* cells and a skew mitigation methodology and an example of software sleep scheduling. Experimental results are described in Chapter 6.5. Related work is reviewed in Chapter 6.6, and Chapter 6.7 concludes this chapter.

6.2 Effects of N/PBTI on Clock Distribution Network

In this section, we present background information and discuss how N/PBTI affects clock distribution network. We first introduce the N/PBTI mechanism and modeling of the device in Chapter 6.2.1. Then we analyze the effect of NBTI and PBTI on clock path delay in Chapter 6.2.2.

6.2.1 N/PBTI Mechanism and Modeling

The exact mechanisms of N/PBTI are still under debate within the reliability community. There are two major NBTI aging models: the reaction-diffusion (RD) model [Bha06, Wan07] and the trapping/detrapping (TD) model [Vel12].

Regardless of the argument of N/PBTI mechanisms and models, it is widely-accepted and well-understood that stress duty ratio is a strong determining factor for the degradation due to its stress-recovery behavior. This work aims at reducing N/PBTI-induced clock skew by balancing the stress-recovery ratio between different clock branches. The purpose of this work is not comparing different aging models. In this work, we will evaluate our methodology using different aging models.

We use the RD model in [Bha06, Wan07]. The time dependence of V_{th} degradation in this model follows a power-law model as:

$$\Delta V_{th}(t) = Kt^n + M \tag{6.1}$$

where the time exponent n$\sim$0.16.

We use the TD model in [Vel12]. The time dependence of V_{th} degradation in this model follows a logarithm timing dependence as:

$$\Delta V_{th}(t) = L \cdot [A + B \cdot log(1 + Ct)] \tag{6.2}$$

To get the realistic amount of N/PBTI degradation,in the experiments, we use a commercial sub-32nm process technology. A calibrated industry aging model (including both NBTI and PBTI, without recovery effect) is used as baseline. The RD model as in Equation (6.1) and the TD model as in Equation (6.2) are fitted and scaled to match the degradation under close to DC aging ($D = 99\%$). Based on the simulation results, the amount of PBTI degradation is significant and cannot be ignored, comparing to NBTI.

6.2.2 N/PBTI on Clock Path Delay

A clock path example is shown in Fig. 6.1. Here we assume that the design uses Flip-Flops (FF) that are triggered by clock rising-edge. So clock skew depends only on the clock path delay associated to this signal transition pattern, i.e., transition from logic state A to logic state B as in Fig. 6.1.

As shown in the table in Fig. 6.1, at logic state A, device P1 and N2 are stressed while N1 and P2 are relaxed. This degradation pattern will result in weaker (i.e., with larger V_{th}) P1 and N2, and stronger (i.e., with smaller V_{th}) P2 and N1, all of which will help the signal transition (i.e., reduce the delay) from logic state A to logic state B. The degradation pattern at logic state B is reversed, i.e., stronger P1, N2 and weaker P2 and N1, all of which will slow down the signal transition from logic state A to logic state B.

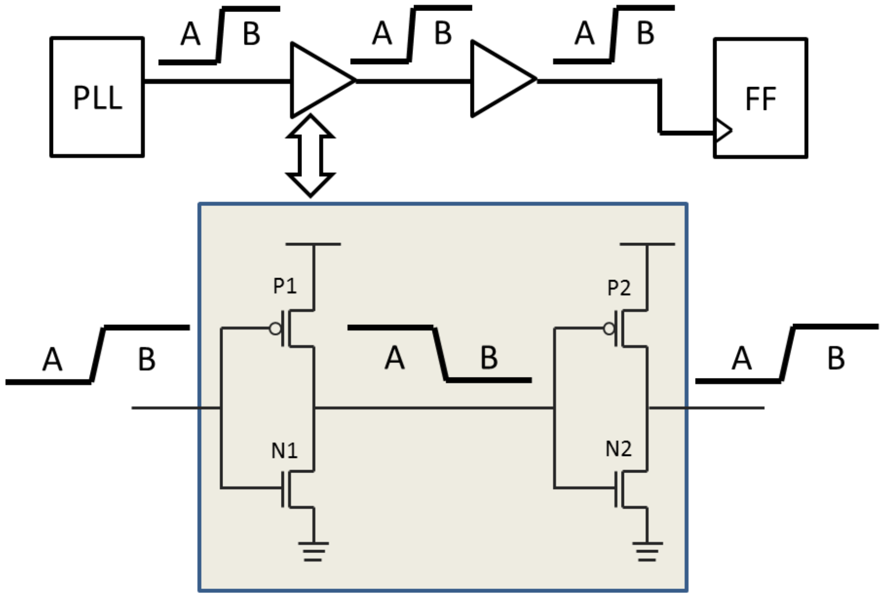
Therefore, we have two conclusions here:

(i) Both NBTI and PBTI will result in the same type of clock path delay change. In this work, we will consider both effects of NBTI and PBTI.

(ii) The amount of N/PBTI-induced delay change on the clock path depends on the time it spends on each logic state. In this work, we define the clock signal duty ratio (D) as the probability of clock path staying at logic state B as in Fig. 6.1.

6.3 N/PBTI-induced Clock Skew under Clock Gating

We have discussed the effects of N/PBTI on clock path delay. In this section, we will explain why N/PBTI-induced delay degradation can result in clock skew if clock gating is implemented on the clock distribution network. We first discuss different clock gating use cases in Chapter 6.3.1. Then we describe the implementation of clock gating using conventional ICG cells in Chapter 6.3.2. We present the analysis of N/PBTI-induced skew and the vulnerability of different clock gating use cases in Chapter 6.3.3. Finally, we discuss



Logic State	N1 status	P1 status	N2 status	P2 status
A	relaxed	stressed	stressed	relaxed
B	stressed	relaxed	relaxed	stressed

Figure 6.1: Effects of NBTI and PBTI on clock paths

the impacts of N/PBTI-induced clock skew and clock uncertainty on design timing margining in Chapter 6.3.4.

6.3.1 Clock Gating Use Cases

Clock gating is one of the most popular power management techniques, due to its low reaction latency and small implementation overhead. It can be implemented at various levels of granularity, depending on the context and use cases. Based on the implementation granularity and software visibility, different clock gating use cases can be classified as follows:

- Architecture-level clock gating: The clock signal of the entire processor core or specific module (e.g., co-processor, hardware accelerators etc.) is gated upon the request of the software. This is typically used when turning the processor/module into sleep mode or

Wait-For-Interrupt (WFI) mode. The clock gating probability has strong dependency on the actual software workload. An example of architecture-level clock gating is C states in ACPI [acp].

- Microarchitecture-level clock gating: Many operations in a processor, especially superscalar processor, may be mutually exclusive and utilize different parts of the circuit. Computer architect can identify the idle modules and corresponding clock gating conditions during RTL design phase. Examples include dynamic high level clock gating [cora] and deterministic clock gating [Li03].

- Circuit-level clock gating: During the circuit design phase, data dependency information can be derived from the RTL. Based on the data dependency information, clock gating can be inserted automatically [Con10, Tel95, Ben96] without affecting the architectural behavior. This feature is also supported by some commercial tools.

6.3.2 Clock Gating Implementation with ICG Cell

Clock is one of the most important and critical signals in VLSI circuit. Any imperfection in the clock signal may cause circuit malfunction or timing violations. For example, a glitch in clock signal may cause the FF to sample incorrect signal and corrupt the architectural state. Clock skew between launching FF and destination FF can cause timing violations.

Since clock signal integrity is crucial for correct functionality, clock gating is usually implemented using a special ICG cell, which is carefully designed to avoid introducing any suspicious timing behavior or glitches. An example of clock gating implementation using a conventional ICG cell [Kea07] is shown in Fig. 6.2. The ICG cell consists of one latch and one AND gate. An enable signal (EN) is used to specify whether the output clock (CKOUT) is enabled. If the ICG cell samples the EN signal as logic low, it will gate the output clock and omit the immediate clock pulse (the dotted pulse in Fig. 6.2) with no clock gating latency.

Although the ICG cell in Fig. 6.2 uses a latch, its timing behavior is similar to a regular

FF, i.e., the signal EN should be and only need to be ready around the incoming clock rising edge with corresponding setup and hold time constraints. This is extremely important for the general applicability of the ICG cell as it specifies the timing requirements of the enable signal EN. In a typical clock gating implementation, the EN signal is generated in the same way as other regular data signals, i.e., driven by the same set of FFs and share the same combinational logic. Therefore, the FF-like timing behavior makes the implementation flow and timing verification consistent with the rest of the design.

Depending on the clock gating granularity, multiple or even thousands of clock sinks can share the same clock gating control. For example, for the architecture-level clock gating use cases, all FFs in the processor core can be gated or ungated together, depending on its sleep state. Usually, only one or a few number of ICG cells will be used at the clock distribution network root. The reasons for this are: 1) this can reduce the number of ICG cells and thus reduce the implementation overhead; 2) this can help saving the clock power dissipation along the clock path. A potential pitfall of this type of implementation is the large clock skew between the ICG cell and its sink FFs(see Fig. 6.2). For large design, the clock path delay can be comparable or even larger than one clock cycle [Res01, Res04]. In this case, even though the ICG cell can cut off the root clock signal with zero cycle latency, it can still take one or more clock cycles to stop the clock signals at the leaf sinks. Special pipeline arrangement should be used to handle this extra latency, e.g., Power Control Register [cora].

6.3.3 Clock Skew Induced by N/PBTI

In a clock distribution network without clock gating, clock signal duty ratio (D), i.e., the probabilities of staying at logic state B, is balanced at 50% between different clock path. This results in balanced delay degradation and no additional clock skew. However, if clock gating is implemented, the signal duty ratio D will be imbalanced between the regular clock branch and gated clock branch, as shown in Fig. 6.3. In this work, we define the clock

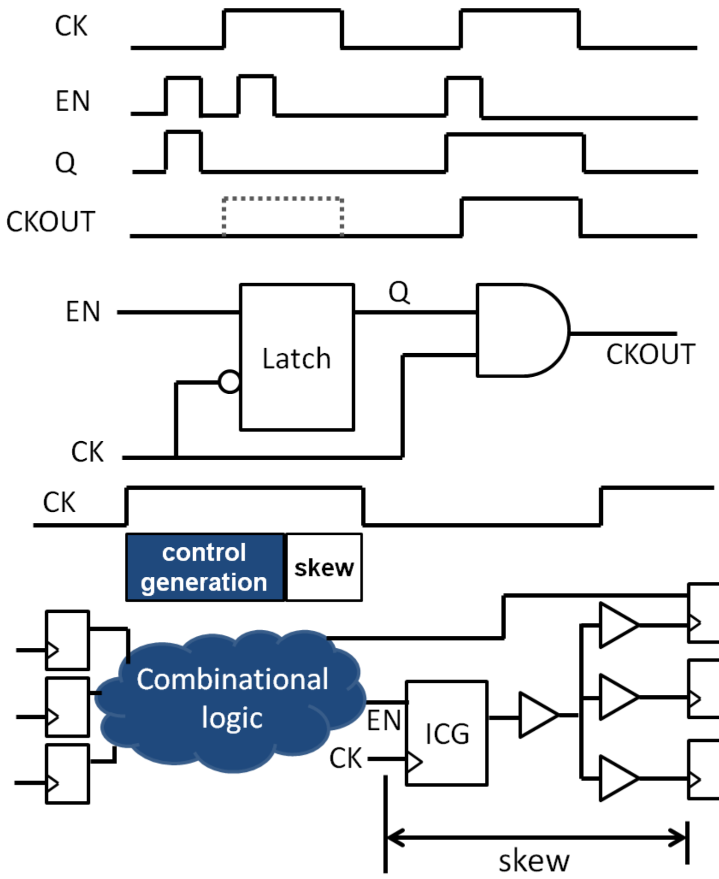


Figure 6.2: Conventional ICG cell and its usage in circuits. The timing behavior of the ICG cell is similar to an edge-triggered element.

activity ratio μ as:

$$\mu = 1 - \frac{\text{total clock gated cycles}}{\text{total cycles}} \tag{6.3}$$

So if conventional ICG cell is used, we have $D = \mu/2$.

As shown in earlier analysis in Chapter 6.2.2, lower D, or equivalently staying more at logic state A, will result in less degradation, i.e., faster clock paths. Therefore, the gated clock path is expected to become faster than regular clock path after aging if conventional ICG cell is used. Both NBTI and PBTI will result in the same type of clock skew, thus aggravating this effect. The clock skew will be mostly determined by the clock path delay after the ICG cell because the clock signal duty ratio D is different only for this part. Therefore, we expect architecture-level and microarchitecture-level clock gating to be more vulnerable to N/PBTI-induced clock skew. We will focus on these two types of clock gating use cases in this work. As shown in Table 6.2 in Chapter 6.5.2, the clock skew becomes significant only when the number of gated clock sinks is large enough.

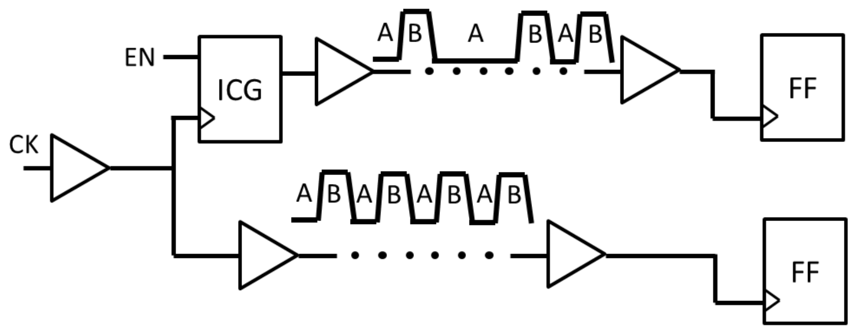


Figure 6.3: Imbalanced signal duty ratio (D) for gated clock branch and regular clock branch

6.3.4 Clock Uncertainty and Design Margining

The exact amount of N/PBTI-induced clock skew depends on the degradation contexts, e.g., lifetime, clock gating activities etc., which are unknown during design time. So it should be considered as additional clock uncertainty on top of other existing static clock skew. This clock uncertainty can either increase or decrease the clock skew and affect the setup-time or hold-time margin, depending on the relative location of FFs. For example, as shown in Fig. 6.4, $skew_1$ will increase the hold-time margin of stage 1 and the setup-time margin of stage 2, while $skew_2$ will increase the setup-time margin of stage 1 and the hold-time margin of stage 2. Therefore, as long as the total clock uncertainty, i.e., $skew_1 + skew_2$, keeps the same, the total design margin can not be reduced.

Unless we can reduce the total delay uncertainty of gated clock path, it is difficult to reduce the design margin by using static design methods. For example, even though we know that the gated clock path is likely to become faster comparing to regular clock path, a static delay shift for gated clock path at $t = 0$, i.e., shifting the shaded part upwards, can only change the split between $skew_1$ and $skew_2$ rather than reducing the total clock uncertainty.

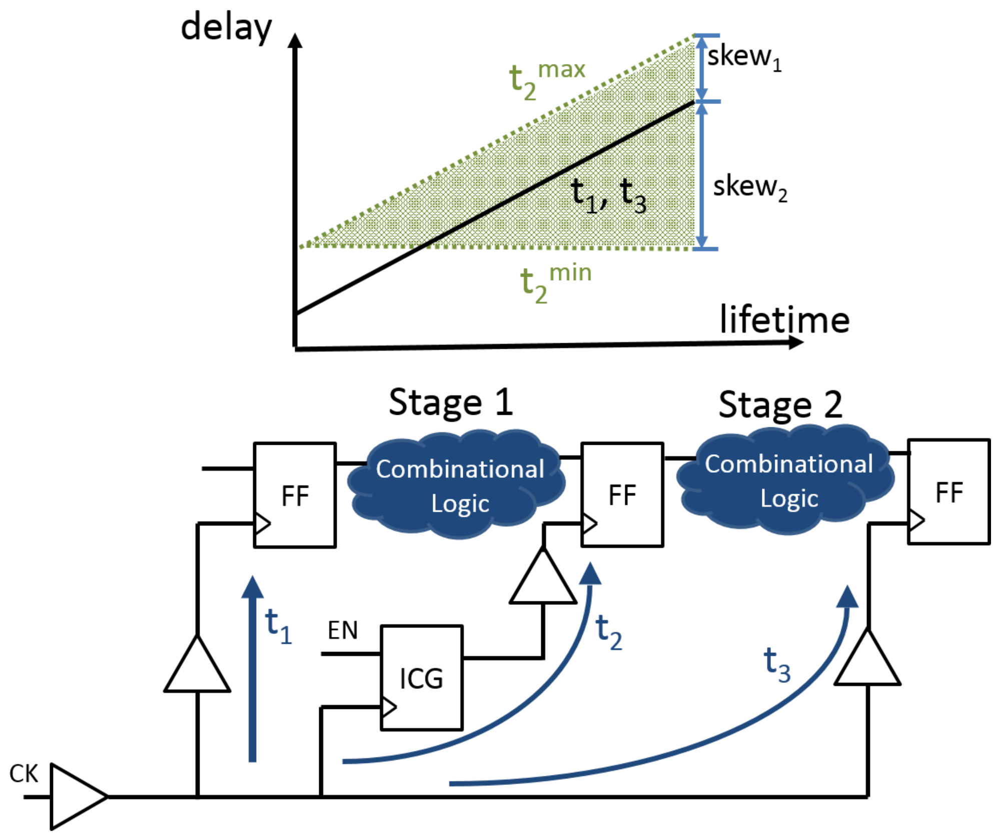


Figure 6.4: Sample illustration of N/PBTI-induced clock skew and clock uncertainty.

So clock gating can change the clock signal duty ratio D of the gated clock path, resulting in additional clock skew and thus design margin. If the ICG cell can alternate clock idle state between logic state A and logic state B, the clock signal ratio D can be balanced to close to 50%, which can eliminate the clock skew introduced by N/PBTI. This motivates our proposed clock gating methodology.

6.4 N/PBTI-Resilient Clock Gating Methodology

In this section, we first discuss the challenges and issues in designing ICG cells with controllable idle state in Chapter 6.4.1. Then we describe *BTI-Gater* cells with half cycle and one cycle clock gating latency in Chapter 6.4.2 and 6.4.3. A skew mitigation methodology is presented in Chapter 6.4.4. Finally, software scheduling techniques are described in Chapter 6.4.5 to avoid certain pathological scenarios.

6.4.1 Clock Gating Latency

Theoretically, it is impossible to implement an idle high ICG cell with zero cycle latency. As shown in the example in Fig. 6.5, at the time when correct clock gating control signal arrives at ICG cell, the output clock is already at logic low state (see the end of cycle 1 as in Fig. 6.5). This makes the AND-type gate, i.e., with logic low idle state, the only option for an ICG cell. If we use an OR-type gate, i.e., making the idle state at logic high, the output clock will inevitably generate an undesired clock rising edge (the red rising edge at the end of cycle 1 as in Fig 6.5), which can corrupt the architecture states. The same issue happens when enabling the clock, where an additional clock falling edge (the red falling edge at the end of cycle 2) is needed.

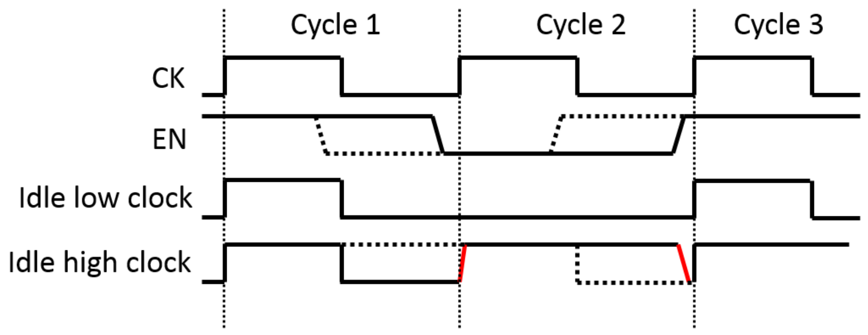


Figure 6.5: Illustration of the clock gating latency issue

The same argument holds if we apply the ICG cell to an inverted clock stage in an inverter-based clock distribution network. The clock control signal, in this case, is coming at the clock falling edge, which makes the OR-type gate the only possible ICG cell with idle state being logic high. Note that at the inverted clock stage, logic high state is equivalent to the logic low state at non-inverted stage. If we use an AND-type gate, the output clock will generate an undesired clock falling edge, which corresponds to a clock rising edge at the FFs.

Therefore, it is impossible to use both AND-type and OR-type gates in ICG cell without changing the clock gating latency. The minimum clock gating latency for an ICG cell with controllable idle state is half clock cycle, i.e., the enable signal arrives before the clock falling edge.

6.4.2 Proposed ***BTI-Gater*** Cell of Half Cycle Latency

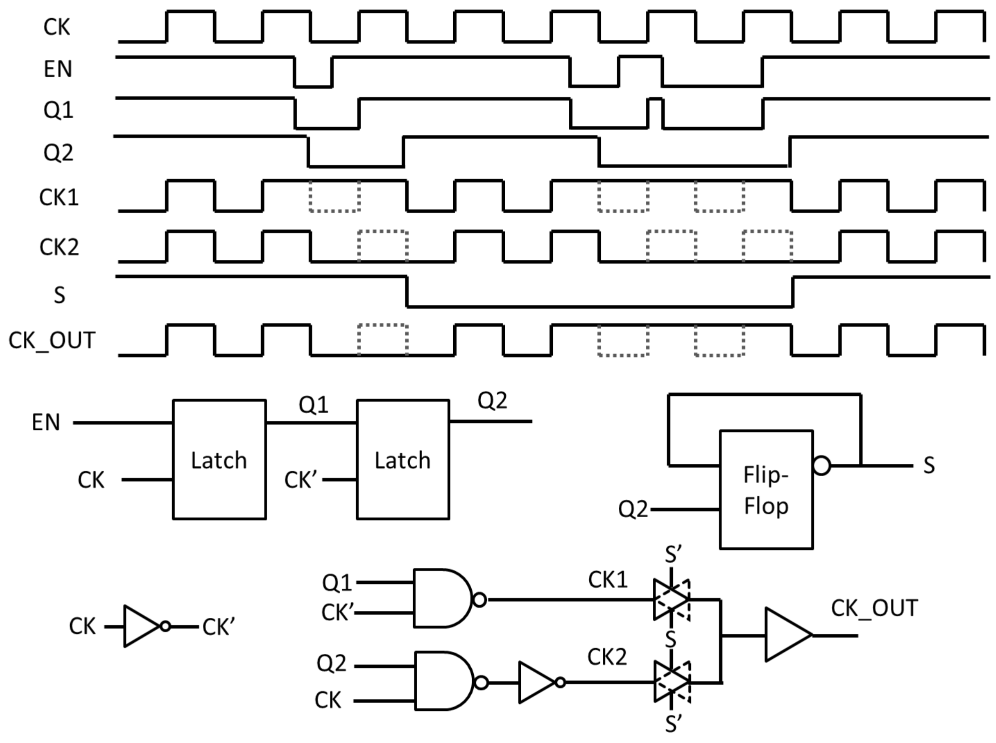


Figure 6.6: *BTI-Gater* cell with half cycle clock gating latency

For an ICG cell with half cycle latency, the enable signal is sampled at clock falling edge. Our proposed ICG cell generates two copies of the gated clock signal with different idle states and alternatively select the appropriate one.

The complete design of *BTI-Gater* cell with half cycle latency is shown in Fig. 6.6. The signal CK1 is the clock signal with logic high idle state. The signal CK2 is the clock signal with logic low idle state. An internally generated signal S is used to select between the two clock signals CK1 and CK2. The value of S is supposed to change each time the circuit enters clock gating state. This *BTI-Gater* cell requires the EN signal to be ready at clock falling edge, which leaves about half clock cycle to generate the control signal.

There are several key implementation points here:

- First, we use one master-slave FF to generate both of signal Q1 and signal Q2. This saves the area and reduces the power overhead.
- S is flipped on Q2's rising edge. Since the rising edge of Q2 is synchronous to CK and happens when both CK1 and CK2 are logic low, this implementation makes sure that no additional glitch is introduced when alternating the state.
- The delays in generating CK1 and CK2 are matched to have exactly one NAND gate and one inverter. This can reduce the clock skew mismatch between the idle high and idle low clock output under variations.

6.4.3 Proposed *BTI-Gater* Cell of One Cycle Latency

As discussed earlier in Chapter 6.3.2, there are cases where clock gating action can not finish within one clock cycle. Special pipeline arrangement is used to handle this additional latency, e.g., Power Control Register [cora]. For this case, an ICG cell with one cycle latency can be accepted and even preferred, because its timing verification is consistent with the rest of the circuits.

We propose *BTI-Gater* cell with one cycle latency as shown in Fig. 6.7. The design is similar to *BTI-Gater* cell with half cycle latency. One more latch is inserted to buffer the enable signal. The same key implementation consideration is applied as well. As shown in the timing diagram in Fig. 6.7, *BTI-Gater* cell will block the second rising edge, i.e., with one cycle latency, after detecting the enable signal.

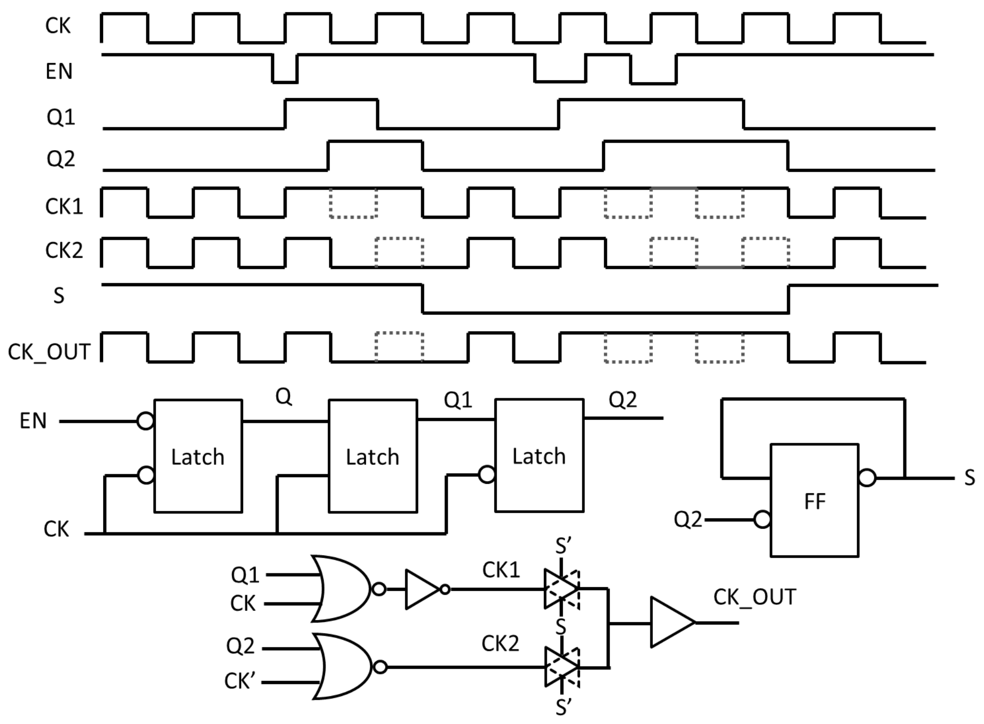


Figure 6.7: *BTI-Gater* cell with one cycle clock gating latency

6.4.4 Mitigation Methodology

Depending on the architectural context and latency requirements, different ICG cells may be used. For example, some microarchitecture-level clock gating cases may specifically require sub-cycle latency. So only *BTI-Gater* cell of half cycle latency can be used. If sub-cycle latency is not required, *BTI-Gater* cell of one cycle latency may be preferred because of its simpler and more consistent delay constraints (with respect to the clock rising edge only).

Based on the properties of the two *BTI-Gater* cells, we propose a skew mitigation methodology to select the appropriate ICG cell, depending on the architectural context and clock gating latency requirements. The methodology is described in Fig. 6.8. First, the clock gating use cases and corresponding latency requirements are identified. Corresponding candidate *BTI-Gater* cells are selected. For clock gating cases with sub-cycle latency and more than half cycle control signal generation time, design guardbanding is the only option.

For other cases, the cost efficiency can be calculated and used to determine if it is beneficial to use *BTI-Gater* cells. The power benefit of using *BTI-Gater* cells can be modeled by a sample cost function as Equation (6.4):

$$\Delta P = \Delta t_{skew}(\alpha \cdot p_s + \beta \cdot p_h) - P_{cell} \tag{6.4}$$

where Δt_{skew} is the skew saving of using *BTI-Gater* cells, p_s is the power cost to margin for per unit setup time uncertainty, p_h is the power cost to margin for per unit hold time uncertainty, P_{cell} is the additional power cost of using *BTI-Gater* cells over conventional ICG cells, α is the percentage of circuits being setup time constrained, β is the percentage of circuits being hold time constrained. In practice, the value of P_{cell} can be pre-characterized. Δt_{skew} can be derived through circuit simulation on the clock path using typical and aged libraries. Direct derivation of p_s, p_h, α, and β is difficult. But the entire term $\Delta t_{skew}(\alpha \cdot p_s + \beta \cdot p_h)$ can be estimated through design optimization tools by adding/removing Δt_{skew} to/from the total clock uncertainty[1]. Design intuition based on the same or similar design/technology can also be utilized to estimate the cost of additional setup and hold margin (i.e., p_s and p_h). $\Delta t_{skew} \cdot \alpha$ and $\Delta t_{skew} \cdot \beta$ can be derived using Total Negative Slack (TNS), which is supported by most design tools.

6.4.5 Software Sleep Scheduling Techniques

N/PBTI degradation on datapath will result in slower path delay and lead to setup-time violations. System adaptation schemes [Min11] can be applied to reduce the design guardband. However, N/PBTI degradation on clock path can result in hold-time violations, which are difficult to resolve through system adaptation schemes. Therefore, from hardware's point of view, design margin can not be reduced as long as there are certain pathological scenarios that can lead to imbalanced degradation, especially for architecture-level clock gating, where the software can have very regular behavior.

[1]In this work, we apply Vt-assignment using commercial tools to estimate the power saving (see Chapter 6.5.5 and Fig. 6.12).

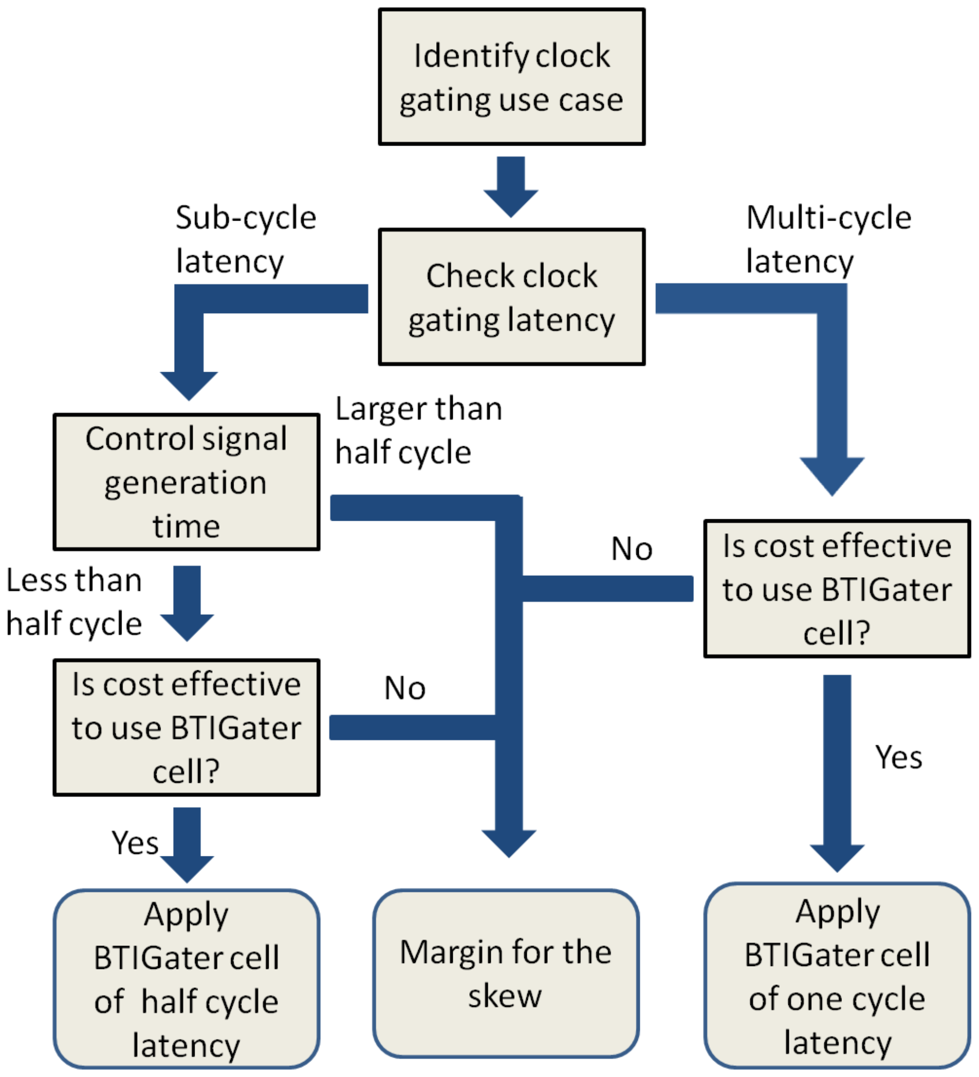

Identify clock gating use case
Check clock gating latency
Sub-cycle latency
Multi-cycle latency
Control signal generation time
Larger than half cycle
Less than half cycle
Is cost effective to use BTIGater cell?
No
No
Is cost effective to use BTIGater cell?
Yes
Yes
Apply BTIGater cell of half cycle latency
Margin for the skew
Apply BTIGater cell of one cycle latency

Figure 6.8: N/PBTI-induced clock skew mitigation methodology

For all typical software workload experiments we ran (see Table 6.3 in Chapter 6.5), *BTI-Gater* can balance the clock signal duty ratio (D) to close to 50%. But one may still be able to construct certain pathological cases that can result in highly imbalanced degradation even with *BTI-Gater*. For example, software may have two program phases where the first phase always results in a very short period of clock gating time while the second one always results in a very long period of clock gating time.

To avoid these pathological scenarios, software sleep scheduling techniques in conjunction with *BTI-Gater* can be applied for architecture-level clock gating cases to balance the clock degradation. A software wrapper can be implemented on the sleep function. A sample software wrapper pseudo code is shown in Algorithm 2. A software bit *idle_state* is set to keep track of the previous clock gating status. Even status and odd status correspond to the idle state low and idle state high cases of *BTI-Gater* cells. Based on *idle_state*, the wrapper uses a software counter *counter* to keep track of the difference between the sleep time spent in even and odd clock gating operations. Upon receiving the sleep request, the wrapper will first determine the desired sleep state based on the value of *counter* and *idle_state*. The sleep operation is broken into two parts ($time/3$ and $time * 2/3$) and scheduled accordingly to achieve a total of $time/3$ spent in the desired sleep state. This makes sure that the absolute value of *counter* is bounded by one third of the maximum value of *time*, if this sleep function is the only mechanism for clock gating. This is tiny and negligible compared to typical hardware lifetime of several years.

If clock-gating is invoked when the processor receiving a WFI request, slightly modified software scheduling techniques can be applied. Instead of breaking the sleep time into two parts, the wrapper can enforce a short period of sleep before turning into WFI state in order to balance the value of *counter*. In this case, the absolute value of *counter* is bounded by the maximum sleep time. If the system expected sleep time can be large and non-negligible compared to its lifetime, a periodic wake-up mechanism can also be used to limit the maximum time of a single sleep operation.

6.5 Experimental Results

In this section, we first describe our experiment setup in Chapter 6.5.1. The clock skew analysis results are presented in Chapter 6.5.2, which verify our earlier analysis in Chapter 6.3.3. Chapter6.5.3 reports the results on *BTI-Gater* cells, including functionality verification, area and power overhead, and clock jitter evaluation. Two different case studies using *BTI-Gater*

Algorithm 2 Software wrapper for sleep scheduling

```
Require: idle_state is previous clock gating status. counter is the software counters to record the difference
  between the sleep time spent in even and odd clock gating operations.
  function mySleep(time)
      if counter > 0 then
          //More time spent in even clock gating status
          if idle_state == even then
              Sleep(time*2/3)
              Sleep(time/3)
              counter −= time/3
              idle_state = even
          else
              //Force the clock idle state
              Sleep(time/3)
              Sleep(time*2/3)
              counter −= time/3
              idle_state = odd
          end if
      else
          //More time spent in odd clock gating status
          if idle_state == odd then
              Sleep(time*2/3)
              Sleep(time/3)
              counter += time/3
              idle_state = odd
          else
              //Force the clock idle state
              Sleep(time/3)
              Sleep(time*2/3)
              counter += time/3
              idle_state = even
          end if
      end if
  end function
```

Case	Benchmark	Gated Module	Clock sinks	Clock levels
I	Processor A	Entire core pipeline	$\sim$ 30k	9
II	Processor A	Part of core pipeline	$\sim$ 10k	8
III	Processor B	Entire core pipeline	$\sim$ 1k	4
IV	Processor B	Part of peripherals	$\sim$ 100	2
V	Processor B	Part of core pipeline	$\sim$ 350	2

Table 6.1: Summary of different clock gating cases

cells are performed in Chapter 6.5.4 and Chapter 6.5.5.

6.5.1 Experiment Setup

In this work, we perform clock gating case study on two commercial processors. Both processors have built-in micro-architecture-level clock gating options. Processor B has built-in architecture-level clock gating to gate part of its peripheral when idle. We also consider the architecture-level clock gating cases when the entire processor core pipeline is clock gated when receiving a sleep request. The different clock gating cases are summarized in Table. 6.1.

To analyze N/PBTI-induced clock skew, we use a commercial sub-32nm process technology and libraries. An industry calibrated aging model (including both NBTI and PBTI, without recovery effect) is used as the baseline. To include the recovery effect, both RD and TD models are fitted and scaled to match the technology node, desired operating conditions and lifetime requirements. We see about 10% lifetime delay degradation for clock buffer chain under 100% stress duty ratio.

6.5.2 N/PBTI-Induced Skew Analysis

In this analysis, we evaluate the amount of N/PBTI-induced clock skew for each of the clock gating cases, assuming only conventional ICG cells are used. For each of the clock gating cases, we first extract the corresponding clock path after the ICG cells. Then we

	Fresh (at t=0)	$\mu = 100\%$ (i.e., ungated)			$\mu = 2\%$			Skew when $\mu = 2\%$		
		RD	TD	Industry	RD	TD	Industry	RD	TD	Industry
Case I	311.4	328.9	328.9	342.7	320.7	312.1	311.2	8.1	16.8	31.5
Case II	229.1	240.8	239.6	250.7	234.8	228.7	228.0	6.0	10.9	22.7
Case III	90.1	93.7	93.4	96.6	91.9	90.2	90.0	1.8	3.2	6.6
Case IV	47.2	47.2	47.2	50.6	47.2	47.2	47.2	0.0	0.0	3.4
Case V	41.5	41.5	41.5	44.2	41.4	41.4	41.5	0.1	0.1	2.7

Table 6.2: Clock path delay and skew (in ps) under different aging models and activities using conventional ICG cells

perform circuit simulation on these paths with degradation based on different aging models and different clock activity ratios (μ). The results are highlighted in Table 6.2. The 100% clock activity ratio represents the scenario that the clock path is never gated. The heavily gated scenario, i.e., with smaller A, has relatively smaller clock path delay, which matches our earlier analysis in Chapter 6.3.3.

The N/PBTI-induced clock skew equals the delta between ungated cases ($A = 100\%$) and corresponding entries in Table 6.2. The skew when $A = 2\%$ is also listed in Table 6.2. The amount of N/PBTI-induced skew depends heavily on the clock path delay. For the longest path, i.e., Case I, the maximum N/PBTI-induced skew can be as much as 17ps based on TD model. While for short paths, e.g., Case III, IV and V, the N/PBTI-induced skew is very small.

6.5.3 ***BTI-Gater*** Cell Results

To validate *BTI-Gater* cell design, we implement both cells using the same sub-32nm technology and libraries. Their functionality is verified by SPICE simulation and is consistent with the timing diagrams in Fig. 6.6 and Fig. 6.7. The area of half cycle latency *BTI-Gater* cell is equivalent to about 25 standard size inverters or 4 standard size FF. The area of one cycle latency *BTI-Gater* cell is equivalent to about 31 standard size inverters or 5 standard size FF. The area of conventional ICG cell is equivalent to about 5 standard size inverters. The power consumption of our proposed ICG cells is about 4X larger than conventional ICG cell. However, in all studied cases, the processor requires at most four ICG cells to implement the clock gating functions. Compared to the size of the entire design and its clock distribution network (see Table 6.1) , the power and area overhead of using *BTI-Gater* is almost negligible. For cases with small clock network, e.g., Case IV and V, the proposed methodology as shown in Fig. 6.8 can be applied to decide whether it is beneficial to use *BTI-Gater* cells.

SPICE simulations are also used to evaluate the potential clock jitter introduced by *BTI-Gater* cell (see Fig. 6.9). The jitter is within 1ps across supply voltage ranging from 0.8V to 1V, which is small compared to the clock uncertainty introduced by local process variations.

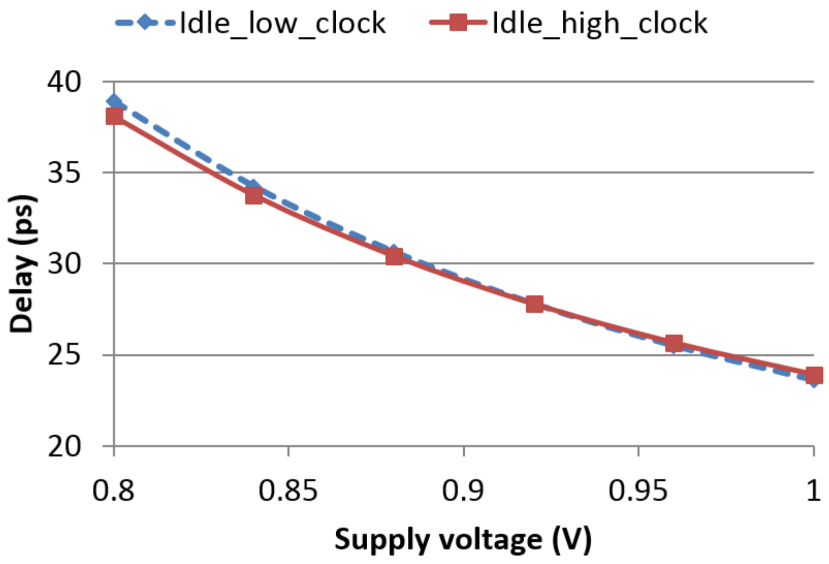


Figure 6.9: Clock delay under different supply voltages. The clock jitter introduced by *BTI-Gater* is very small.

6.5.4 Case Study for *BTI-Gater* Applicability

To verify and demonstrate the applicability of *BTI-Gater*, we apply *BTI-Gater* methodology for Case V. Since Case V specifically requires sub-cycle clock gating latency, *BTI-Gater* cell of half cycle latency is considered. The processor is synthesized, placed and routed with the clock gating implemented using conventional ICG cell. After verifying the clock gating control signal generation time, we replace it with *BTI-Gater* cell of half cycle latency through Engineering Change Order (ECO). The design is verified again after the ECO to make sure that the clock gating control signal meets the timing constraints with respect to the clock falling edge. The layout with clock distribution networks highlighted is shown in Fig. 6.10.

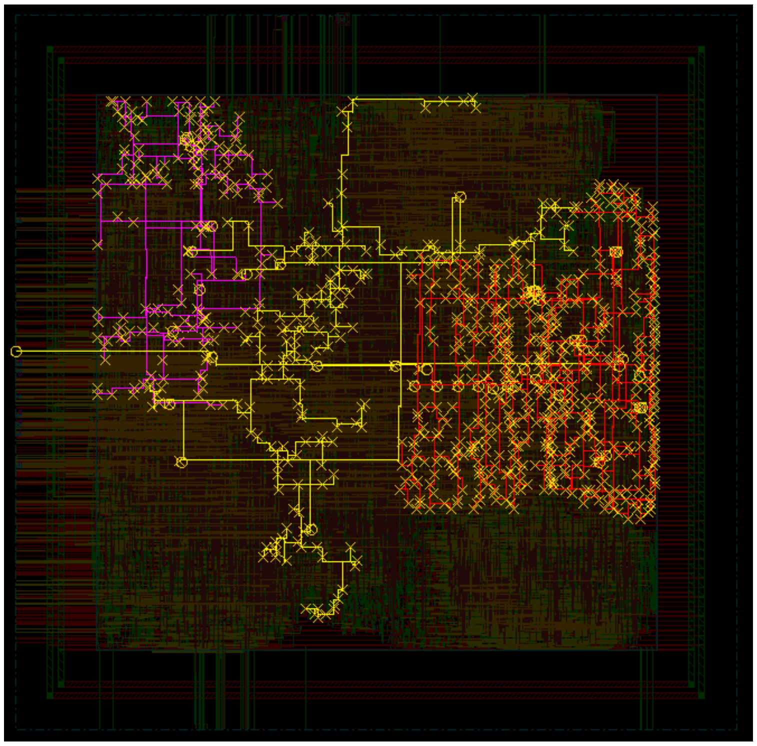


Figure 6.10: Layout for processor B with clock distribution networks highlighted (ungated part in yellow, Case IV in purple and Case V in red)

We also perform functional simulation of processor A by running Dhrystone [Wei84] benchmark. The clock activity ratio μ in Case V is less than 9%, which implies a clock

Benchmark	Clock activity ratio (μ)	Clock signal duty ratio (D)	
		Conventional	*BTI-Gater*
mp3	2.14%	1.07%	49.98%
Dhrystone	39.26%	19.63%	56.53%
3D rendering	23.25%	11.62%	46.73%
Web browsing	6.78%	3.39%	52.66%

Table 6.3: Clock activity ratio(μ) and signal duty ratio(D) results for different software benchmarks

signal duty ratio (D) of less than 5% for the gated clock path if conventional ICG cell is used. Our *BTI-Gater* cell is able to generate a clock signal with $D = 52.68\%$, which is very close to standard clock signal duty ratio of 50%.

6.5.5 Case Study for Power Saving

To evaluate the benefit and potential power saving of our skew mitigation methodology, we apply *BTI-Gater* methodology on Case I. Since Case I is architecture-level clock gating, *BTI-Gater* cell of one cycle latency is used. The corresponding clock skew with conventional ICG cell under different clock activity ratio (μ) is shown in Fig 6.11. The maximum skew is about 17 ps with TD model and about 10 ps for other aging models. We also expect the clock gating probability to be a strong function of the software workload. We perform simulation of various software benchmarks and generate the active-sleep patterns. The benchmark active-sleep patterns are used to simulate the clock signal duty ratio D for both conventional ICG cell and *BTI-Gater* cells. As shown in Table 6.3, *BTI-Gater* cells can balance the clock signal duty ratio D to the range between 46.73% and 56.53%, which will reduce the corresponding clock skew to less than 1ps. This improvement is expected to be larger if more software iterations are running on one processor or the software scheduling techniques as in Chapter 6.4.5 is used.

To evaluate the power saving of the clock skew reduction, we make the following assump-

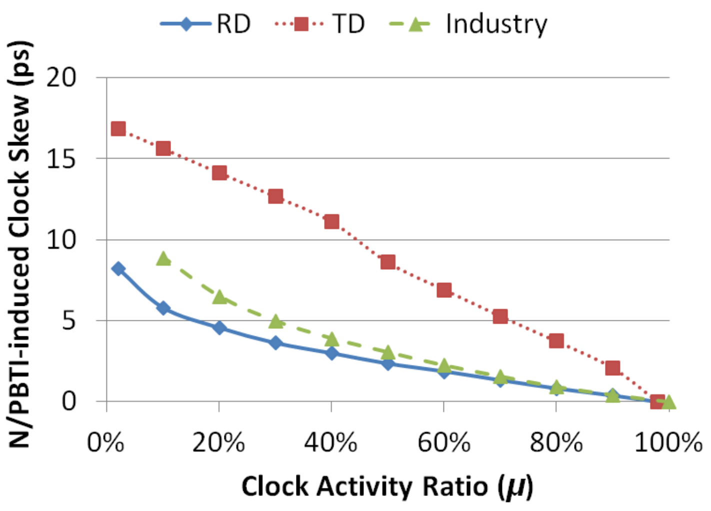


Figure 6.11: Clock skew with conventional ICG cell for Case I

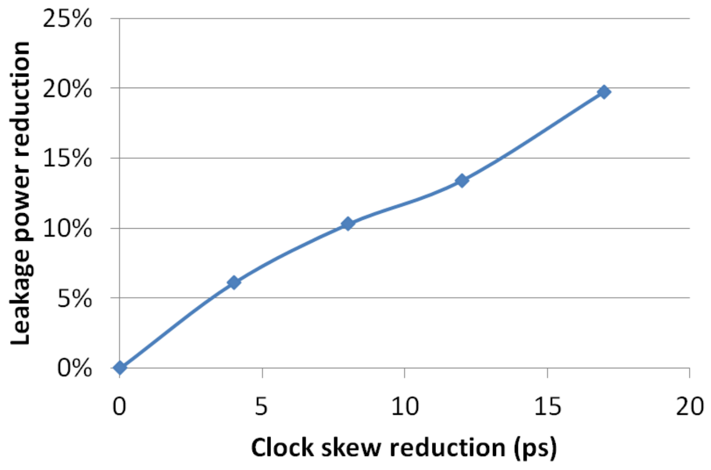


Figure 6.12: Leakage power saving by Vt-assignment with different clock reduction for Case I

tions: 1) the designer will guardband for the worst-case P/NBTI-induced skew if conventional ICG cell is used; 2) *BTI-Gater* cell can balance the clock signal duty ratio D to around 50%. In the experiment, this processor design is first synthesized, placed, routed, and signed-off with the maximum N/PBTI-induced skew (i.e., 17ps) as additional clock uncertainty margin. Then we apply *BTI-Gater* methodology and remove the corresponding amount of clock uncertainty. The additional timing slack is reclaimed by Vt-assignment using commercial tools. The leakage power results for different amounts of skew reduction are plotted in Fig. 6.12. The results show that our method can save up to 19.7% leakage power from the 17ps relaxation in clock uncertainty. Since Vt-assignment will only swap cells with the same gate size but different Vt options, the dynamic power remains the same.

6.6 Related Work

There is several related work aiming at reducing N/PBTI-induced clock skew issue by different approaches.

Chakraborty et. al. [Cha13a] proposed a scheme to statically select AND-type or OR-type cells as the output stage of the ICG cell. There are three potential drawbacks in this method. First, the method relies on knowing the clock gating probability at design time. With incorrectly assumed clock gating probability, the method may aggravate rather than reduce N/PBTI-induced skew. Second, the method did not consider the generation of clock control signal, which leads to the inappropriate assumption that both AND-type and OR-type ICG cells can be used at the same time. As discussed in Chapter 6.4.1, it is non-trivial to implement both AND-type and OR-type ICG cells. Last, the method works only if clock gating is implemented at multiple locations in the clock distribution network, which limits its applicability.

Huang et. al. [Hua13] proposed a scheme to identify critical PMOS in the clock distribution network and selectively use AND-type or NAND-type ICG cells. This method focuses on reducing the aging effect within the ICG cell rather than the clock buffers after it. Therefore, this method can only offer limited benefit in reducing N/PBTI-induced clock skew for clock gating with coarse granularity, which is more vulnerable to N/PBTI-induced clock skew.

Chen et. al. [Che13] proposed a scheme to selectively replace cells on clock path with higher Vt cells to compensate both aging and process variation. This method applies Vt-assignment on the clock distribution network so that the clock path with larger degradation has smaller initial delay at t=0. However, as discussed in Chapter 6.3.4, this method may not reduce the clock uncertainty due to N/PBTI-induced skew, thus may not reduce the design margin.

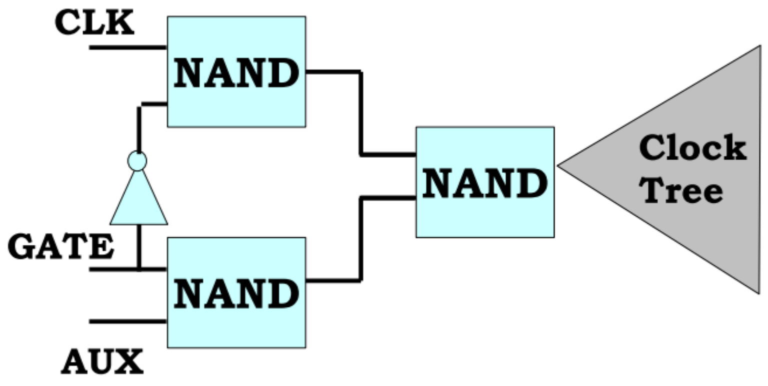


Figure 6.13: The ICG cell proposed in [Cha09a].

Chakraborty et. al. [Cha09a] proposed an ICG cell design (see Fig. 6.13) that can alternate the clock state based on an external signal AUX. This is the most closely related work. However, this method does not consider the synchronization of the control and state alternating signals. This leads to the same assumption that AND-type and OR-type gating cells can be used at the same time, which is non-trivial to implement.

6.7 Chapter Conclusions

In this chapter, we first analyzed the effect of N/PBTI on clock distribution network with clock gating features. Then we proposed two *BTI-Gater* cells that can be used to balance delay degradation on gated clock branch and regular clock branch. Last we proposed an N/PBTI-induced skew mitigation methodology. Software sleep scheduling techniques are described in conjunction with *BTI-Gater* to avoid certain pathological aging scenarios.

Experimental results show that *BTI-Gater* cells can be used to reduce N/PBTI-induced clock skew by up to 17ps. By using our skew mitigation methodology, we can save up to 19.7% leakage power compared to pure design guardbanding. Our future work includes a silicon prototype of the proposed approach and its validation under accelerated aging scenarios.

CHAPTER 7

Evaluating and Exploiting Impacts of Dynamic Power Management Schemes on System Reliability

7.1 Chapter Introduction

Hardware reliability has been a major concern for nano-scale computing systems. Different design-time hardware implementation choices and run-time software management schemes, though may not be specifically designed for reliability purposes, can significantly affect the system's resilience. Therefore, it is necessary to analyze their effects under different application workloads and even exploit them to retain system resilience.

There can be two possible ways to implement system power management schemes to use system performance slack. Run-fast-then-stop (RFTS) completes the workload with nominal performance and then goes to certain sleep states for the remaining slack to reduce power. Just-in-time (JIT) tries to adjust the peak performance to elongate the runtime with lower power consumption. Both of them can be used to achieve power saving but can have different impacts on reliability.

There are two main questions to be answered:

- How are reliability-related phenomena affected by different power management mechanisms and hardware implementation choices?
- Can the system exploit different power management mechanisms to achieve its reliability target under varying ambient conditions and runtime workload?

In this work, we try to address the two issues in the context of radiation-induced soft error, as the hardware design choices and system power management schemes can jointly affect soft error rate (SER). For example, power gating, as one of RFTS schemes, can be implemented with or without state retention flip-flops (FFs) [Kea07]. Power gating without retention FFs will dump its state into memory, which helps eliminate logic SER during gating. However, power gating with retention FFs will increase SER as FFs in retention mode are more vulnerable to soft errors.

Radiation-induced soft error also poses challenges of how to estimate SER, especially for mobile devices. It is difficult to monitor by hardware itself due to the unique ambient (i.e., location/altitude) dependence [Gor04] and rare occurrence nature. The same type of device can be used by users living at different geographic locations. Even the same mobile device can be used at different locations over time. We observe that most mobile devices are typically equipped with various hardware sensors (barometers, GPS etc.) and Internet connection capabilities. Therefore, system-level or even cloud-based SER monitors can be implemented to capture the device location information and to assist the reliability management.

The key contributions of our work are the following:

- We implement a hardware evaluation platform based on BeagleBone Black development board and standard Linux kernel. We demonstrate the use of our platform for studying the system power and SER under different hardware design choices, application workloads and software management schemes.

- We propose a system/cloud-based virtual sensing approach to capture the varying location/altitude for SER estimate. This enables the reliability adaptation and management.

- We propose new reliability management policies to exploit the flexibility in existing Linux power management schemes. We show that our forbidden-state based approaches are effective in achieving system reliability target with minimal power overhead compared to the optimal schemes characterized offline.

Previous work addressed either the hardware reliability modeling challenges [Ale11,Ebr14] or proposed new sophisticated power management algorithms [Pop07,Zhu06,RMD07]. They focused on a specific hardware model and did not consider the power and reliability impacts of wide varieties of hardware support for power management. Our evaluation platform focuses on studying the interactions between different hardware design choices and representative power management schemes in generic system software (i.e., Linux kernel). Our evaluation platform will be using the reliability models in some of the existing work. Our reliability management policies are proposed to exploit flexibility in different existing power management schemes rather than proposing new power management algorithms. Previous work [Mer14,Kar06,RMD07] addressed the reliability management problem in the context of other reliability-related issues, including Negative-/Positive-biased Temperature Instability (N/PBTI), Electromigration (EM), etc. Our framework can be extended to consider these reliability issues as well.

The rest of the chapter is organized as follows: Section 7.2 introduces some background information about radiation-induced soft error, hardware support for power management and power management schemes in Linux kernel. Section 7.3 describes the development of our evaluation platform and some results of using the platform to assess the system power and SER. Section 7.4 presents our approaches and results of power state management policies for retaining system reliability targets. Section 7.5 concludes the chapter.

7.2 Chapter Background

7.2.1 Radiation-Induced Soft Error

Radiation-induced soft error is caused by alpha particles or neutrons hitting silicon and flipping circuit logic states. The actual failure rate or failure-in-time (FIT) rate depends on the technology [Shi02, Mah10], circuit structure (e.g., logic gate structure, size) [Ebr14], operating points (e.g., supply voltage) [CA08] and ambient conditions (e.g., altitude) [Gor04].

A model for calculating altitude-dependent SER is proposed in [Gor04] as:

$$SER = \int \sigma(E) \cdot (\frac{d\phi}{dE}) dE \tag{7.1}$$

where $\sigma(E)$ is the bit fail cross section for particle energy E, $d\phi/dE$ is the fluence rate per unit energy, or differential flux. The main altitude dependence of $d\phi/dE$ is exponential attenuation based on the atmospheric depth d. This dependence can be represented as a factor $F_{alt}(d)$:

$$F_{alt}(d) = exp(\frac{d_{SL} - d}{L_n}) \tag{7.2}$$

where d_{SL} is the atmospheric depth at sea level, and L_n is the effective attenuation length. This altitude-dependence model is reported with less than 5% error compared to measured data across different geographic locations [Gor04]. In this work, we will use pre-characterized FIT rate data as baseline and apply the altitude dependence in Equation 7.2.

Various techniques have been proposed to detect and correct soft errors [Wan06, Mit07, RKS14, RCK15]. Different architecture components can have different vulnerability to soft error [Muk03], and the error propagation can become very complicated [Cho13]. It is difficult to combine the SER of different components into one system failure rate. In this work, we assume that the system has two separate SER targets for the processor core and memory, since the factors we studied, such as supply voltage and location, affect the SER of either or both components. For core SER, only FF soft errors are considered, as combinational soft errors contribute a relatively small fraction [Ebr14]. The FFs include both architectural visible and non-visible (i.e., pipeline FFs, control FFs etc.) ones. For memory SER, we consider only soft errors in SRAM without ECC, as SRAM can be well-protected with ECC and physical interleaving.

7.2.2 Hardware Support for Power Management

Most power management mechanisms are supported by some special hardware design techniques or configurations. Different design choices can result in different management efficiency (e.g., power saving and switching latency) as well as hardware reliability.

One example is the design of on-chip power delivery network in support for voltage scaling. In a typical design, memory other than L1 cache is designed to operate at a constant voltage and thus on a separate memory power rail. There are two possible ways to connect the power supply of L1 cache. One common way is to connect it to the same power rail as the processor core. This design can avoid the voltage level converter between the processor core and L1, but will also limit the lowest supply voltage which is usually determined by the SRAM V_{min}. The other way is to connect L1 to the separate power rail. This allows more aggressive voltage scaling but requires level converter and results in longer delay and higher design overhead.

Another example is the implementation of state retention for power gating. As mentioned earlier, the state retention can be realized by either retention FFs or dumping architecture states into memory. An example of retention FF is plotted in Fig. 7.1 [Kea07]. The retention part is usually implemented by smaller transistors and powered by lower supply voltage than the master/slave latches on the main path. This makes the FF under retention mode more vulnerable to soft errors.

There can be a lot of different consideration for making these hardware design choices. The purpose of our work is studying the interactions between these design choices and system-level power management schemes rather than comparing them from reliability or power perspective. Therefore, we will make our implementation flexible for targeting these different hardware configurations.

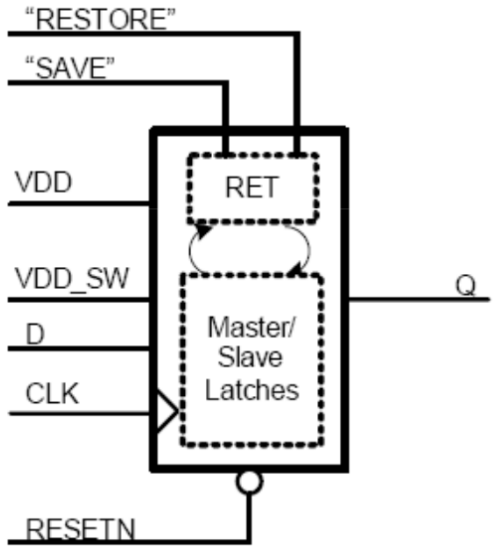


Figure 7.1: An example of state retention Flip-Flop [Kea07]. The retention part is powered by a different supply rail than the main master/slave latches.

7.2.3 Power Management in Linux Kernel

Different power management schemes can have substantially different impacts on SER. Previous work has proposed sophisticated scheduling and voltage scaling algorithms in the context of soft error for real-time systems [Pop07, Zhu06, QZA11, FHL15, SM10]. However, most existing work does not consider the interactions between software management schemes and hardware implementation choices.

For evaluating hardware design choices, it is important to correctly account for the generic software system and workload behavior. In this work, we build our platform based on Linux kernel of version 3.12. The two power management schemes, JIT and RFTS, are implemented in Linux kernel as CPUFreq and CPUIdle modules respectively.

CPUFreq [cpua] (see left top of Fig. 7.4) is the module inside Linux kernel for CPU frequency scaling. Depending on the hardware driver and implementation, the supply voltage may or may not scale with the frequency, as voltage scaling does not need to be visible to the software. The CPUFreq policy is a data structure used to record information, such as current governor, operating frequency etc. The module is scheduled to run at a fixed period. At the beginning of each period, CPUFreq governor retrieves system utilization information from the kernel. Based on CPUFreq policy data and the frequency table, the governor will set a frequency target or range. An example state diagram of CPUFreq governor is shown in Fig. 7.2. The utilization threshold for state switching is configurable in Linux kernel. In this work, we use the default values of ondemand governor. Driver, which is the actual code talking to the hardware, will try to scale the frequency/voltage (i.e., P-states) to match the target specified by the governor. [1] The actual change done by the driver will be updated to CPUFreq policy and broadcasted through a system notifier.

CPUIdle [cpub] (see Fig. 7.3) is the module to support multiple CPU idle levels inside Linux kernel. Different available idle states (i.e., C-states) are pre-characterized with their corresponding power consumption and exit latency. The idle state selection is based on target residency, which is the minimum idle time to achieve a net energy saving compared to a lighter idle state. At runtime, CPUIdle module is called when the system completes the tasks and enters the idle loop. The module generates an estimate of the expected residency (i.e., sleep time) based on next scheduled event and other history information like past interrupts. The governor will select the idle state whose target residency is the largest but is still not larger than the expected residency. Corresponding driver function is used to prepare and enter the selected idle state. Upon wake up, the module will recover the system state and record the actual residency time.

7.3 Reliability Evaluation of Power Management Schemes

An evaluation platform is essential for studying system reliability under system-level power management with various hardware configurations and software applications. Some key

[1]Latest version of the kernel deprecates this and requires the driver to set to the exact target frequency as specified.

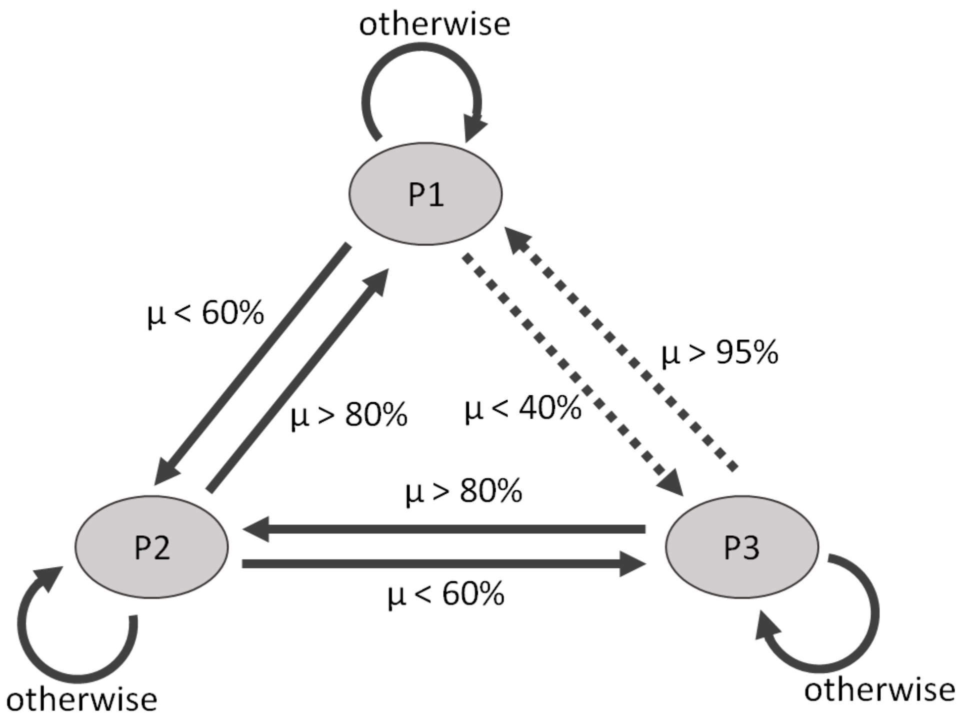


Figure 7.2: State transition diagram example based on CPU utilization μ for CPUFreq governors. Ondemand governor can switch between any two levels while conservative governor can only switch between neighboring levels. The switching threshold values are configurable in Linux kernel, and we use the default values for ondemand governor.

questions can be answered by experiments using such an evaluation platform, e.g., how will software management schemes in conjunction with hardware design choices affect system reliability, what is the worst-case workload for system reliability. In this section, we describe the implementation of the evaluation platform. The base platform is introduced in Section 7.3.1. The targeting hardware configurations and corresponding power and SER models are explained in Section 7.3.2. The platform customization is described in Section 7.3.3. Some experimental results on system reliability evaluation are presented in Section 7.3.4.

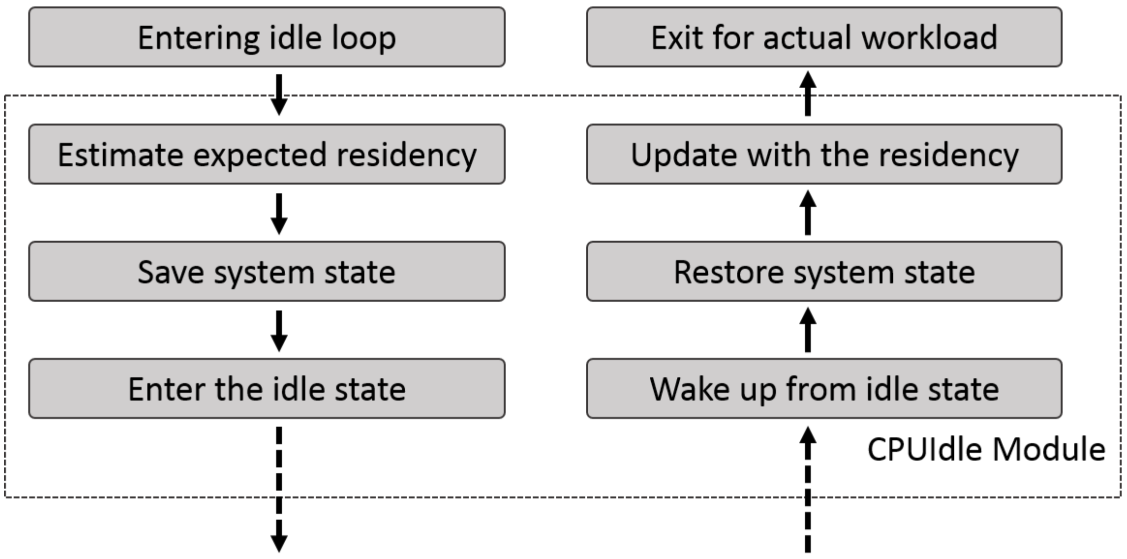


Figure 7.3: CPUIdle module flow chart

7.3.1 Base Platform for Reliability Evaluation

The hardware evaluation platform has to be general enough for targeting different hardware configurations as well as fast enough for running system software. Therefore, implementing actual hardware for each of the configuration is impractical and simulation-based approaches are not feasible. Emulation-based tools [Wan13b] can meet both the speed and flexibility requirements, but lack accurate time accounting for frequency scaling and entering/leaving idle states.

In this work, we build the platform based on BeagleBone Black [BBB] board, on which the kernel and software are running. Power, reliability and different power management latencies are emulated online with a customized kernel module or by inserting dummy busy loops. The power and reliability models are pre-characterized and treated as input for the kernel module.

BeagleBone Black board is based on AM335x ARM Cortex-A8 processor [am3a] with 45nm process technology. The processor has separate L1 32KB data cache and 32KB in-

Table 7.1: Table of different hardware implementations

	Retention FF	L1 Rail	L1 ECC
Case I	No	Same as core	No
Case II	No	Separate	No
Case III	Yes	Same as core	No
Case IV	No	Same as core	Yes

struction cache. L2 cache is of 256KB. The software stack is based on TI SDK7 [AM3b]. Linux kernel is of version 3.12 with support for both CPUIdle and CPUFreq modules. Software benchmarks are used to represent different workloads, including web browsing, MPEG 4 decoding, FFPlay [ffm] and 3D image rendering. We also implement a synthetic benchmark in order to study the system behaviors under different workload intensity.

7.3.2 Hardware Configuration and Modeling

As discussed in Section 7.2.2, different hardware configurations can have different impacts on reliability. The hardware evaluation platform will be used to emulate systems with these configurations. The design configurations to be explored include the use of retention FFs for power gating, whether L1 cache is under the same power rail as the processor core, and whether L1 cache is protected with ECC. The hardware design configuration cases studied in this work are listed in Table 7.1.

Power and SER models for these configurations are essential for our evaluation platform. In this work, we derive the power and SER models based on a 45nm technology. We also use projected power and SER model for 28nm technology to study the impacts of technology scaling in Section 7.3.4.

BeagleBone Black is based on Cortex-A8 processor. Since we do not have the detailed design information, building the power and delay models at different supply voltages and idle states for the exact processor is not possible. In this work, the power and delay model is derived based on the synthesis, placement and routing results of a similar processor using commercial tools [enc] with commercial 45nm process technology and libraries. The power and delay models should reflect the voltage-dependence of this process technology and libraries for mobile-class processors. As will be discussed later, the SER model is based on the FIT rate of individual elements (i.e., FFs and SRAM cells), which is independent of the processor model after normalization.

The power values are calculated using the same commercial tools [enc] with the libraries. To model the power difference under voltage scaling, we also re-characterize the libraries at the supply voltages for all P-states in Table 7.2. The libraries include the retention FF cells, which are used to model the power when retention FFs are used. The power saving for clock gating (i.e., C2 state in Table 7.3) is derived by stripping off clock power. The memory power model is based on the memory compiler results for the same process technology. The memory compiler is configurable with options to include memory ECC and/or retention modes. The power difference is taken into account with respect to the corresponding targeting design configuration cases in Table 7.1. The power model projection for 28nm is done by repeat the same process with commercial 28nm technology and libraries, including memory compiler results.

As mentioned earlier in Section 7.2.1, it is difficult to model or compute the system failure rate based on SER of each individual component. So the processor core SER and memory SER are considered separately. In fact, even for individual component like processor core, the failure rate is extremely difficult to model due to masking effects at various levels, different visibility and vulnerability. However, the factors we are considering in this work, i.e., voltage and altitude, have similar effects on FFs or SRAM cells. Moreover, none of these factors will change the error propagation or behaviors other than increasing or decreasing the probability of getting bit-flips. In this work, we model SER of processor core or memory by the FIT rates of FFs or SRAM cells.

Sea-level processor core SER values are based on FIT rates in [Ebr14] for 45nm technology and only sequential elements (i.e., FFs) are considered. The core soft error rate is calculated based on the total number of FFs in the synthesized processor netlist. The SRAM SER values are also derived based on the results in [Ebr14, Ale11] and voltage dependence in [CA08]. The 28nm SER values are scaled based on the model in [CA08]. Altitude dependence of SER is modeled by Equation 7.1 based on the work in [Gor04]. The parameters in Equation 7.1 are derived based on the design manual of the same 45nm technology.

7.3.3 Platform System Customization

The platform supports both CPUFreq and CPUIdle modules with five frequency levels and two idle states, Wait-For-Interrupt (WFI) and clock gating. To be able to evaluate more variety of power management schemes, two more idle states, including core power gating with and without memory retention mode, are added. Extra latency are emulated by inserting additional dummy busy loops. In this work, we use the ondemand CPUFreq governor.

A customized power/reliability bookkeeping module is developed for taking the system trace and keeping track of the system power and SER at runtime. The module structure is illustrated on the left bottom of Fig. 7.4. The module reads in the pre-characterized power and nominal SER model values for the processor core and cache. Based on these values and device location information, the module updates the system's current power consumption and SER for normal running and each C-state. Any frequency change notifier or location update will trigger the module and recalculate the power/SER values. A logging thread is running at a fixed period to calculate and record the accumulated power and SER, based on the corresponding time spent in normal running and each C-state.

The information for supported P-states and C-states are summarized in Table 7.2 and Table 7.3. The frequency switching delay shown in the driver is 0.3ms, which makes the CPUFreq module run every 300ms. But our actual measurement shows much longer latency for frequency level switching, which is mainly caused by the low bandwidth I2C communi-

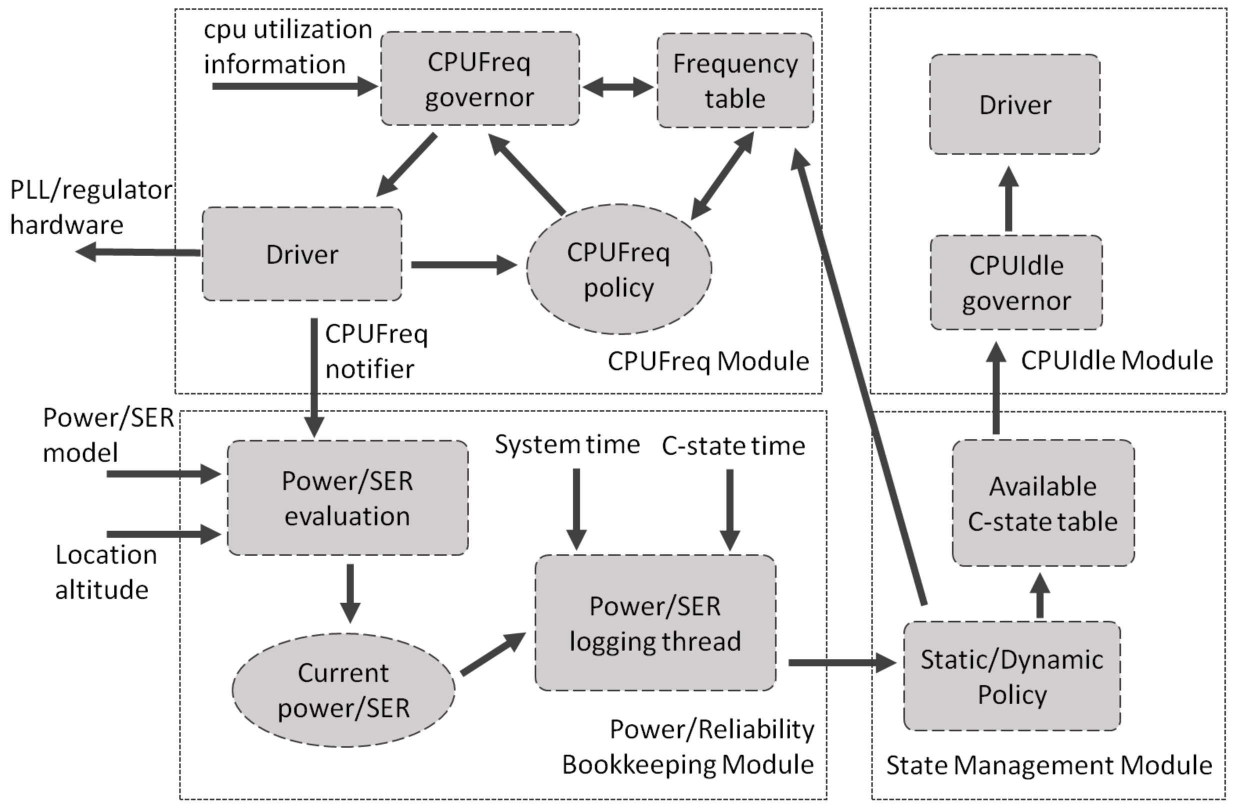


Figure 7.4: Overall block diagram

cation between the processor and the off-chip regulator. Since only frequency change need to be visible to software stack, we also implement a customized version of the driver that bypasses the voltage changes and only does the frequency scaling. This gives our platform the capabilities to emulate systems with fast frequency switching latencies.

7.3.4 Results of Reliability Evaluation

The experiments are done with the hardware evaluation platform based on BeagleBone Black board (see Fig. 7.5). The device is assumed to be operated at sea level. The reliability bookkeeping thread is configured to generate a log entry every 100ms to limit the interference and have enough resolution for tracking the states.

Table 7.2: Table of different CPUFreq states spec used

Frequency	Voltage	Latency
1000 MHz	0.90V	2.1ms
800 MHz	0.82V	2.1ms
720 MHz	0.80V	2.2ms
600 MHz	0.76V	2.2ms
300 MHz	0.72V	3.2ms

Table 7.3: Table of different CPUIdle states spec used

State	Description	Exit Latency	Target Residency
C1	WFI	68us	150us
C2	clock gating	130us	200us
C3	core power gating	530us/1060us[2]	800us/ 1450us
C4	core power gating with SRAM retention	650us/1180us[2]	1000us/ 1550us

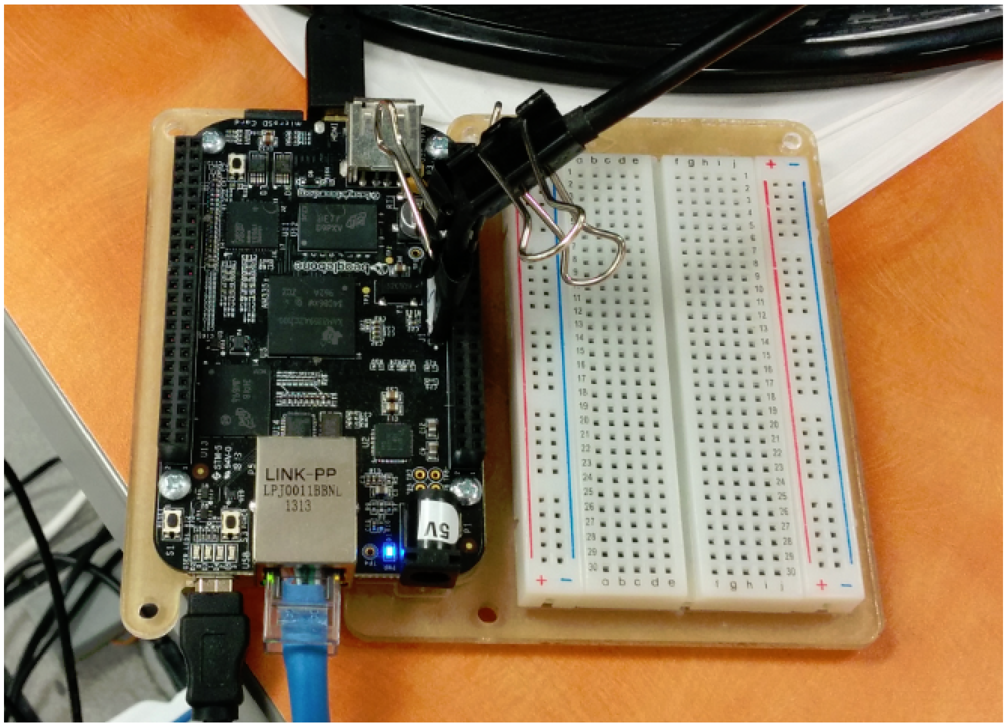


Figure 7.5: Photo of the hardware evaluation platform setup based on BeagleBone Black development board.

The results of different hardware configuration cases with our synthetic workload are plotted in Fig. 7.6. The synthetic workload is implemented as a busy-loop that is scheduled every 20ms. The busy-loop is configurable with a tunable number of iterations to load the system with different workload intensity. 100% workload intensity means the number of iterations in the busy-loop is configured to fully load the system with no performance slack. 40% workload intensity means the number of iterations in the busy-loop is 40% of that for the 100% workload intensity case. For the ease of cross-comparison, all results in this section are normalized with respect to the maximum power/SER values from Case I study, which is about or equivalent to 0.26W power, 200 FF FIT and 500 SRAM FIT for per mega cells.

[2]Exit latency and target residency for system with/without retention FFs

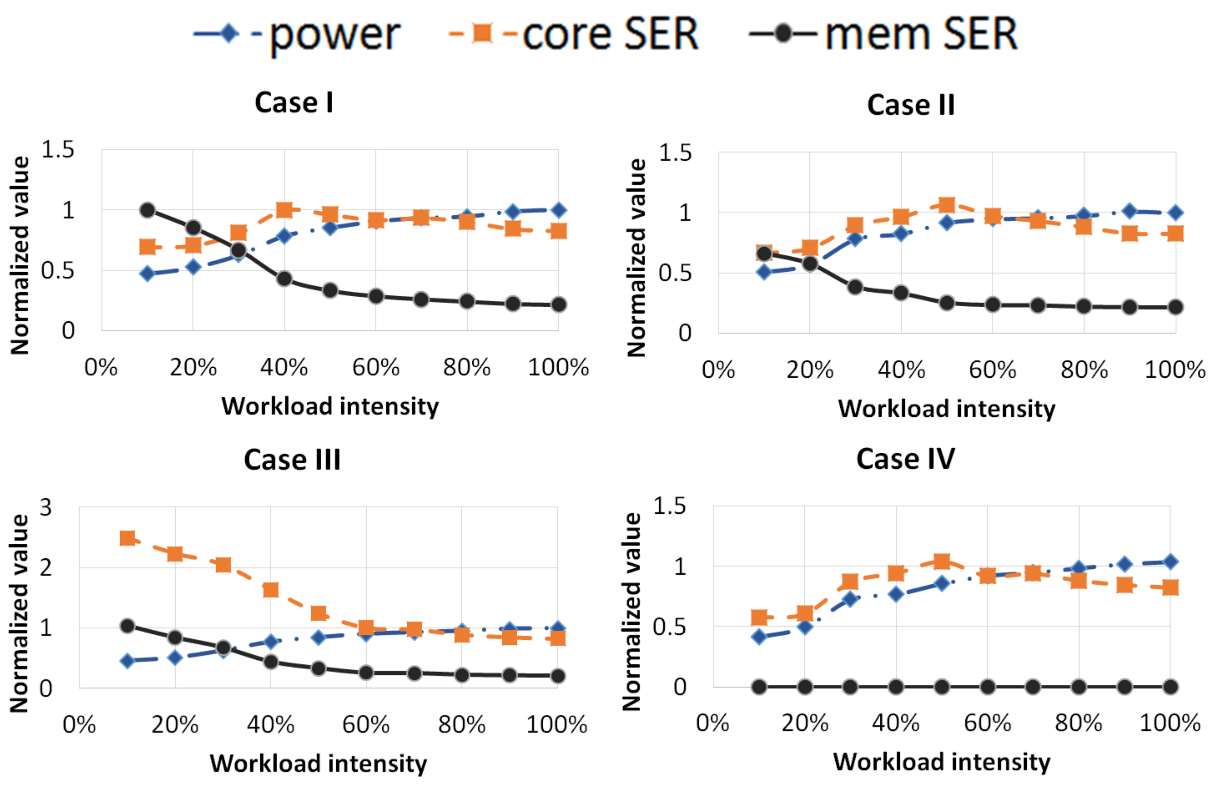


Figure 7.6: Power/SER Results of different hardware configuration cases with different software workload intensity at sea level. 100% workload intensity means the system is fully loaded and has no performance slack.

In all cases, the power increases with increased workload intensity. This is expected as higher workload intensity imposes higher performance demand on the system, which results in higher voltage state and less idle time. For systems without L1 memory ECC, i.e., case I, II and III, memory SER decreases with increased workload intensity, due to less time spent in retention mode and lowered supply voltage levels.

An interesting observation is that for systems without retention FFs, i.e., case I, II and IV, core SER is non-monotonic with respect to the workload intensity. Peak core SER occurs at medium (around 40% - 50%) workload intensity, where system is operating at low voltage level and with little slack for going into deep idle states. This suggests the potential

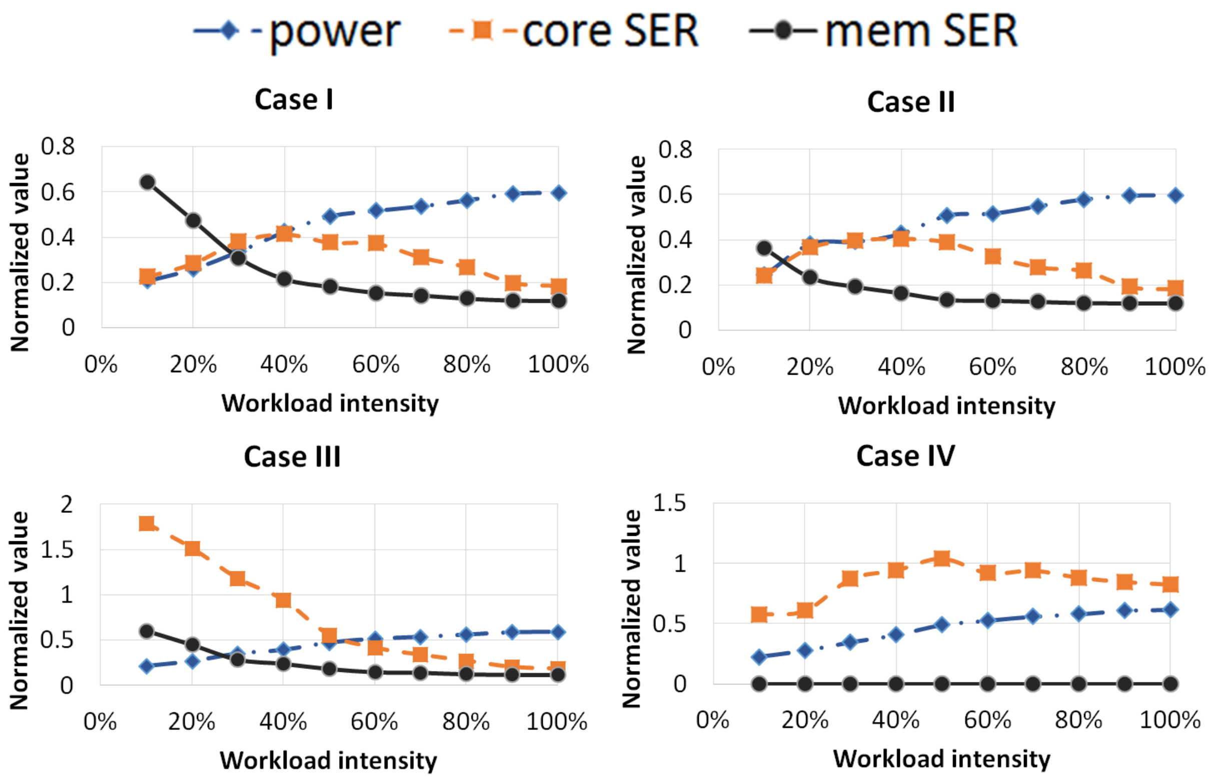


Figure 7.7: Power/SER Results of different hardware configuration cases with different software workload intensity for 28nm technology.

of preferring an RFTS power management approach to reduce SER. This motivates the reliability management policies proposed in Section 7.4. For system with retention FF, i.e., case III, core SER is significantly higher than other cases and peaks at low workload intensity regions, as the FFs in retention mode are more vulnerable to soft errors. A potential software solution to this can be implementing a virtual idle state with all or part of CPU states dumped to RAM even in presence of retention FFs. This idle state can be used for low workload intensity scenarios, where sleep/wake-up latency requirements may be relaxed but core SER is the highest.

To study the impact of technology, the experiments with synthetic workload is repeated with the projected 28nm power and SER models. The results are plotted in Fig. 7.7. The

Table 7.4: Normalized power/core SER/memory SER results of different software benchmarks with different configuration cases at sea level

	Web browsing	MPEG 4 decoding	FFPlay	3D image rendering
Case I	0.35/0.57/1.05	0.58/0.83/0.77	0.36/0.80/0.51	0.54/0.85/1.01
Case II	0.39/0.59/0.65	0.57/0.77/0.45	0.37/0.80/0.39	0.58/0.85/0.62
Case III	0.34/2.30/1.04	0.61/1.93/0.70	0.35/1.57/0.52	0.52/2.42/1.01
Case IV	0.34/0.55/0	0.55/0.86/0	0.33/0.76/0	0.52/0.83/0

trend of power and SER changes is very similar to the results in Fig. 7.6. Similar to the results reported in [Shi02], we observe reduced SER of sequential elements at smaller technology. This reduced SER is primarily due to the smaller FIT rate for each individual sequential element, as a result of shrinked diffusion area and increased driving strength per diffusion area. In reality, this result may be different considering that the design at smaller technology is likely to have more logic gates and larger memory capacity.

We also demonstrate the use of our platform for evaluating power/SER with real life benchmarks. Four software benchmarks are used, including web browsing, MPEG 4 decoding, FFPlay and 3D image rendering. We launch one benchmark at a time on the platform and record the average power and SER over a fixed period. The period starts 3 seconds before the beginning of the benchmark and lasts long enough for the benchmark to finish. This matches the running environments of typical embedded/mobile devices and includes the warm-up and cool-down period of the system, which is important for modeling the system-level power management schemes.

The power and SER results of running different software benchmarks are highlighted in Table 7.4. The numbers are in line with the results in Fig. 7.6. MPEG 4 decoding is the most demanding workload, therefore have the highest power consumption and lowest memory SER. The core SER is highest for 3D image rendering, which has slightly smaller workload intensity than MPEG 4 decoding. Comparing the results of web browsing and FFPlay, they have very similar power results. However, both core SER and memory SER are very different for these two benchmarks. Average core SER is much higher for FFPlay, and memory SER is much higher for web browsing. This is because the two benchmark have very different workload patterns. Workload for FFPlay is more steady, while web browsing behaves more like bursts of workload. As the results of these workload patterns, the power management behaves more like RFTS for web browsing and JIT for FFPlay. This further motivate our proposed reliability management policies, which will be described in Section 7.4.

7.4 Exploiting Power Management Schemes for Reliability

Since different power management schemes can all achieve power saving but have different impacts on system reliability, they can potentially be used to retain the system reliability target if ambient condition (e.g., location) changes. As prerequisite of such adaptation, a system/cloud-based virtual SER sensing is proposed in Section 7.4.1.

As mentioned earlier in Section 7.3.2, we consider the processor core SER and memory SER separately. In this work, we assume the system has specific SER target for the processor core and memory in order to retain its mean-time-between-failure (MTBF). For a real system, error injection studies can be performed to derive the relationship between soft errors (i.e., bit flips) and system failures. Based on the desired system MTBF, the SER target can be calculated.

The SER target can be a static constraint, i.e., specification sheet target, which means that the instantaneous SER at any time should be kept smaller than it. We propose a forbidden-state based policy for such static constraints in Section 7.4.2. Since typical soft error MTBF can be of days to months, restricting the instantaneous SER may be too pessimistic and unnecessary. We also consider the case when the constraint is dynamic, i.e., the system's overall SER within certain period should be kept smaller than the SER target.

This period should be much shorter than the soft error MTBF (in days to months) but still significant longer than the power management operation time (in milliseconds). We propose a dynamic state enabling/disabling policy for such dynamic constraints in Section 7.4.3. Experiment results are described in Section 7.4.4. The effectiveness of our dynamic policy is evaluated in Section 7.4.5.

7.4.1 SER Estimation Mechanism

Soft error itself can result in different software/hardware symptoms and be detected by corresponding mechanism [Wan06]. SER, however, is extremely difficult to measure or estimate due to soft error's rare occurrence nature and unique ambient dependence. Direct measurement of the error occurrence such as measuring memory ECC [Zha14] errors or checkpoint recovery events may be possible. But it will require huge memory size or extremely long measurement time, as the soft error rate are typically less than one per mega devices per month.

Our proposed mechanism is motivated by the fact that most modern mobile devices are equipped with various types of sensors (e.g., GPS, barometer) and network connection capabilities. A system-level virtual altitude/location sensor can be implemented based on these sensors. SER estimation of current location can be made with altitude-dependence model [Gor04] or a locally-stored look-up table. A cloud-based service can also be used to answer incoming queries of the measured flux rate data, solar activities and SER estimate from mobile devices with given location information.

In this work, we assume the system is equipped with the virtual altitude sensor. Whenever the virtual altitude sensor updates the altitude value, our reliability bookkeeping module (see Fig. 7.4) will update the SER estimate for both core SER and memory SER based on the altitude-dependence model. The SER estimate is based on a pre-characterized reference SER value at sea level and scales with an exponential function of altitude dependence from Equation 7.2. The overhead of SER calculation is negligible. Alternatively, a local look-up table can be used to store the pre-characterized SER values at different altitudes.

7.4.2 Forbidden-State Based Policy

Different approaches can be used to affect the system power management choices and account for ambient condition changes. A good approach should be both effective and non-intrusive so that the impacts on the system behavior and performance is minimal and mostly predictable. Based on these considerations, we decide to preserve existing Linux power management modules and perform the reliability management by manipulating the available power states. In this way, our reliability management policy is isolated and independent from the power management schemes. This also improves the compatibility of our approaches with arbitrary power management governors.

Considering the structure of kernel power management schemes (see Fig. 7.4), we propose a forbidden-state based policy for enabling/disabling certain P-states and C-states, depending on system's current SER. The policy is implemented as a stand-alone module (see Fig. 7.4). For P-states, the module will change the frequency table directly so that the driver that decides the final frequency level will select only the available states. For C-states, a C-state table is maintained for CPUIdle governor to enter only the available idle states.

A static policy can be used for the case of static SER target. Whenever the reliability bookkeeping module updates the system SER, the state management module compares the SER values to the target SER and disables all states whose SER values are higher. This guarantees that the instantaneous SER at any possible power state can meet the reliability target.

7.4.3 Dynamic State Enabling/Disabling Policy

For dynamic SER target, the static policy may be too conservative and unnecessary. Therefore, we propose a policy which limits the average SER, SER_{avg}, within a given period of T

through dynamically enabling/disabling the available P-states and C-states.

The policy is built based on our reliability bookkeeping module described in Section 7.3.3. The module keeps track of the accumulated SER at any time t as $SER_{acc}(t)$. So at the beginning of each period t_{start}, the accumulated SER target for the end of this period, SER_{end}, can calculated as:

$$SER_{end} = SER_{acc}(t_{start}) + SER_{avg} \cdot T \tag{7.3}$$

At any time t, the remaining SER budget, SER_{budget}, until the end of this period t_{end} can be calculated as:

$$SER_{budget} = \frac{SER_{end} - SER_{acc}(t)}{t_{end} - t} \tag{7.4}$$

The policy will use SER_{budget} instead of the actual SER target to determine which states should be enabled or disabled.

The overall structure of the state management module is kept the same with this dynamic policy. Only an update request mechanism is added to calculate SER_{budget} on demand. Whenever the CPUFreq or CPUIdle module access the state table, a request is initiated to update current power management states. The state management module can also initiate a request for reliability bookkeeping module to update the SER values. Though we are working on a single-core platform, the module and policies are compatible for multi-core systems. For multi-core system, scheduling can also be a strong knob with techniques like workload aggregation or distribution in trading-off JIT vs. RFTS.

7.4.4 Experimental Evaluation of Reliability Management Policies

All experiments are run with the hardware evaluation platform described in Section 7.3. The dynamic policy is configured to account for overall SER over a period of one second, i.e., $T = 1s$ for Equation 7.3. The altitude values are fed to the system through a script. Corresponding SER values for processor core and memory are calculated based on the model in [Gor04]. The reliability target is calculated based on each hardware configuration's worst

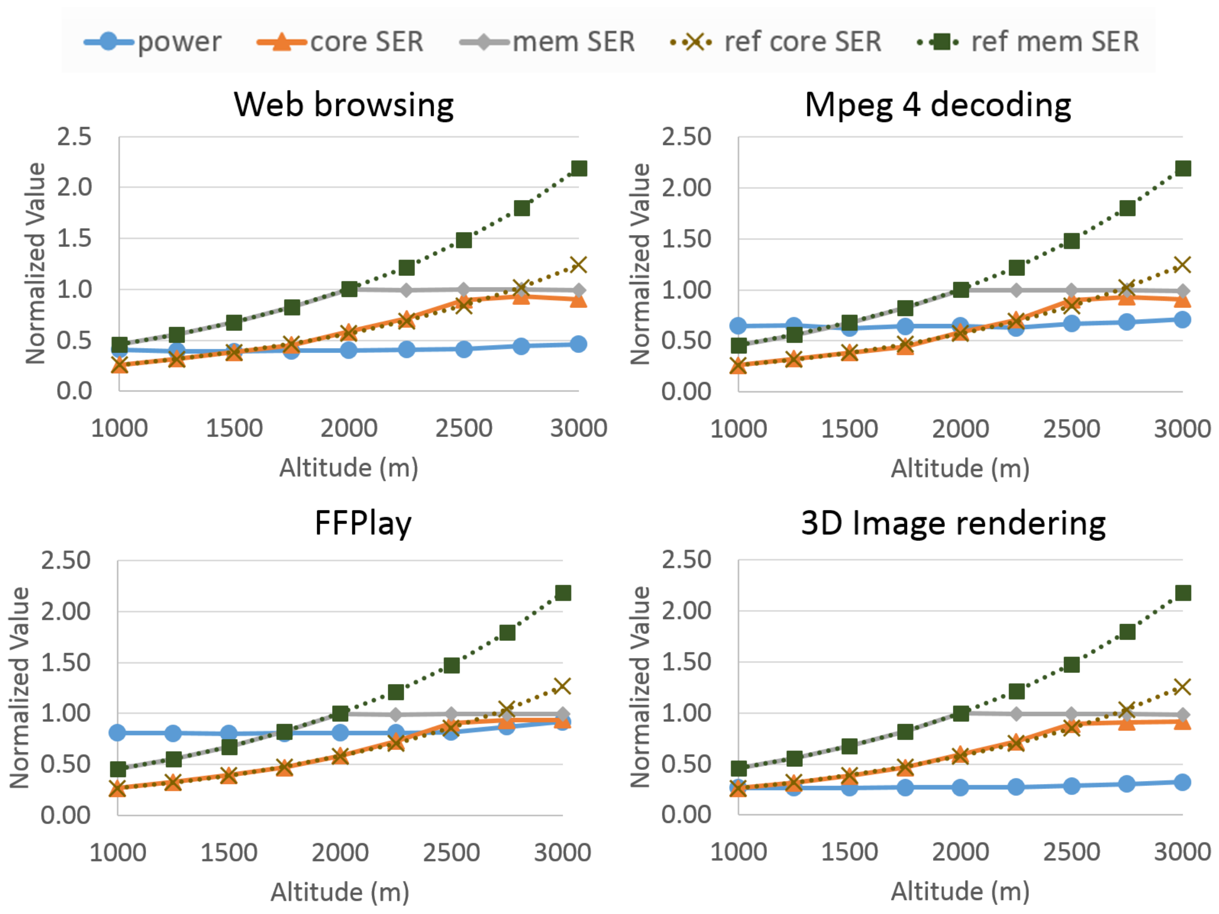


Figure 7.8: Power/SER Results of different software benchmarks on case I with dynamic policy.

SER states at an altitude of 2000m. In this section, if not otherwise mentioned, we report the maximum accumulative SER value over the period of one second and normalize it with respect to the SER targets, i.e., an SER value 1 means we are right at the SER target.

The implementation of the policies is verified by the reliability bookkeeping module. Both the static policy and dynamic policies can effectively control the SER with increasing altitude value. Some of the results with dynamic policies are plotted in Fig. 7.8, 7.9, 7.10, 7.11. For all benchmarks, our dynamic policy is able to retain both core SER and memory SER under the targets. In Fig. 7.8, the core SER reaches the target at around 2000m altitude, while

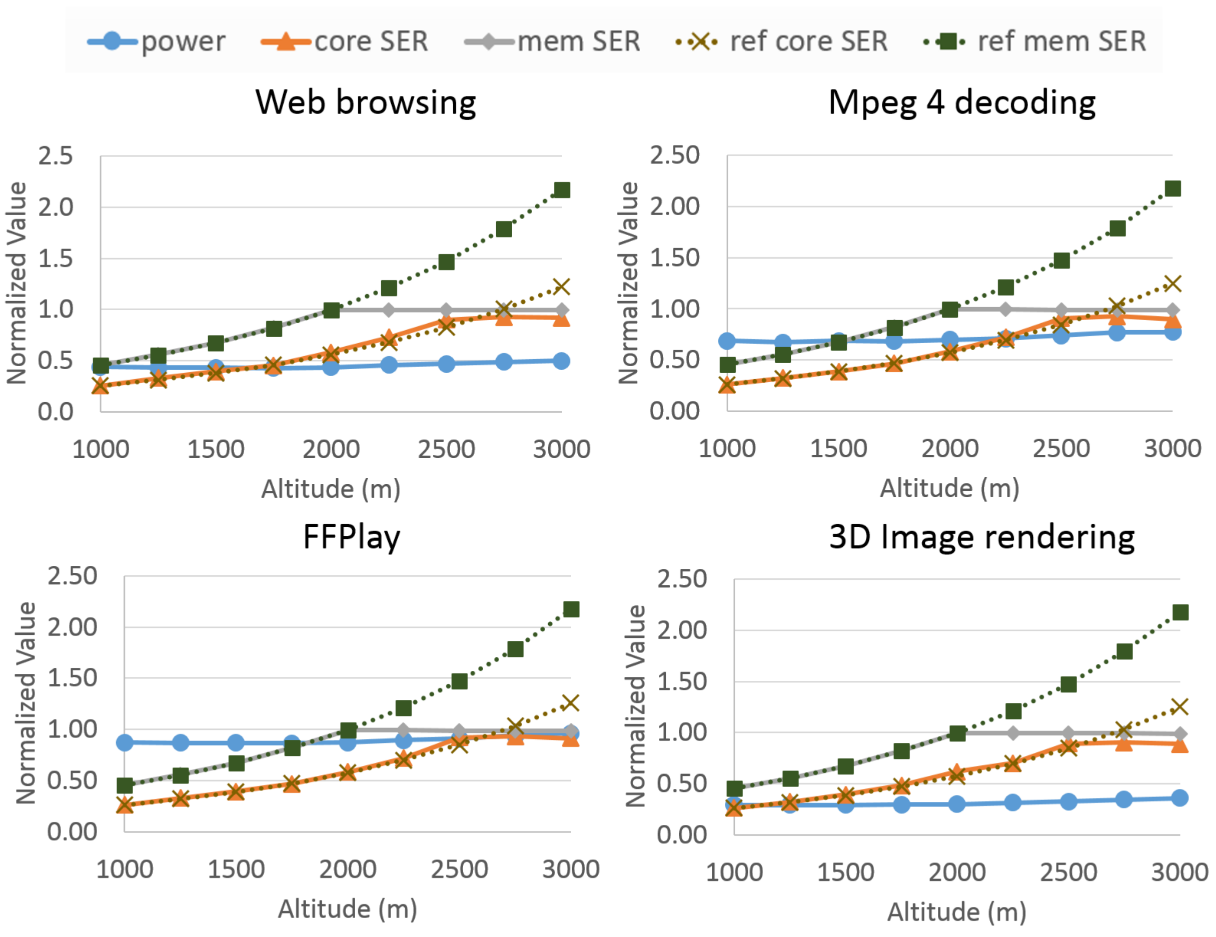


Figure 7.9: Power/SER Results of different software benchmarks on case II with dynamic policy.

the memory reach the target at 2500m altitude. The reason is that the worst-case workload patterns for core SER and memory SER are different (as suggested in Fig. 7.6). In Fig. 7.10, the core SER and memory SER are in the same pace because their dependence on workload is similar (see Fig. 7.6).

For all cases, the power overhead is small except for some large altitude cases. This is because extensive use of higher voltage states and shallower idle states are required once the gap between reference SER values and SER targets becomes large. This can also be

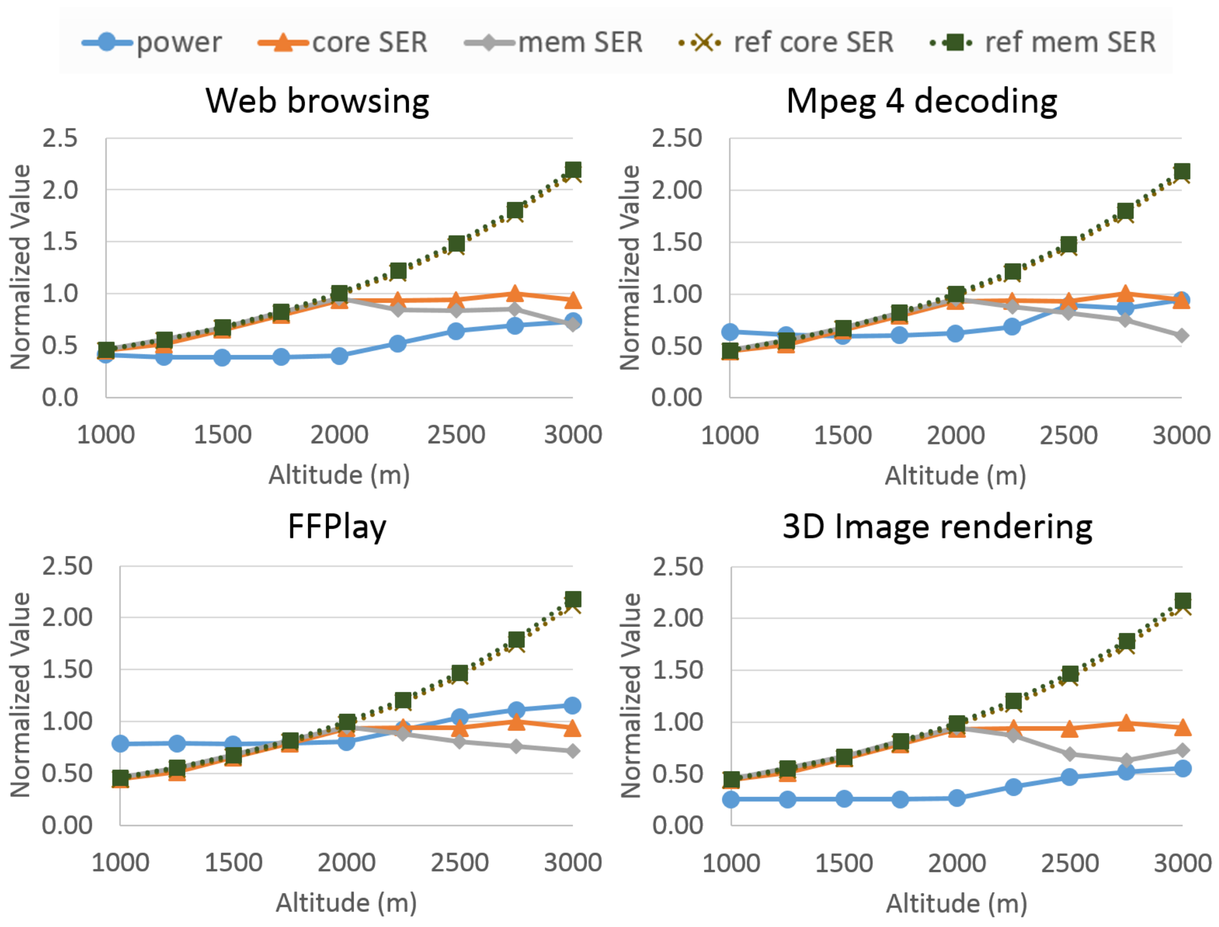


Figure 7.10: Power/SER Results of different software benchmarks on case III with dynamic policy.

explained with the limited dynamic range of SER in all possible power states. The study on the effectiveness of our dynamic policy against optimal power state selection schemes will be described in Section 7.4.5.

The other way to study the policy behavior is to analyze how different power states are enabled/disabled. An example of web browsing benchmark is shown in Fig. 7.12. A free-running trace at sea level (see Fig. 7.12(a)) is used as a baseline for comparison. For case I at 2500m altitude (see Fig. 7.12(b)), where memory SER dominates, the policy retain SER target by disabling low voltage states, resulting in entering higher frequency states at

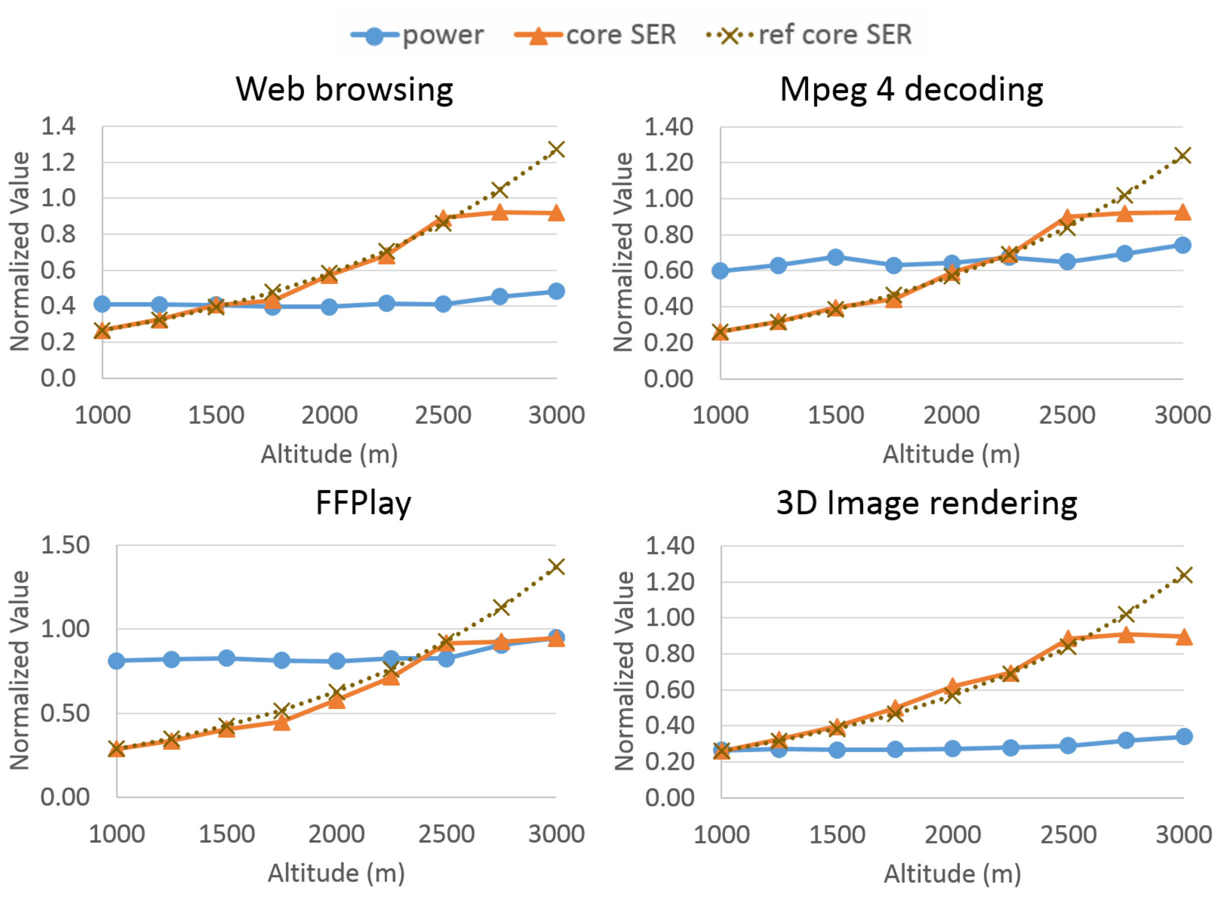


Figure 7.11: Power/SER Results of different software benchmarks on case IV with dynamic policy.

around $t = 5s$ in Fig. 7.12(b). For case III at 2500 altitude (see Fig. 7.12(c)), where core SER dominates, the policy retain SER target by disabling deep sleep state so that the FFs spend less time in retention mode. This results in static frequency but varying SER (due to enabling/disabling the deepest idle state) at around $t = 5s$ in Fig. 7.12(c).

The system performance is also evaluated through quality metrics such as runtime and frame miss rate. Since our policies only disable lower frequency levels or long exit latency idle states, system performance with our policies is always the same or better. The overhead of our reliability bookkeeping module and state management module is negligible compared to the latency involved in changing power states.

The small power overhead seen in the results is partly due to the fact that voltage scaling is not a strong knob for this technology. We also observe this from the generated libraries at different supply voltages. For other high sub-threshold swing devices like FinFET, the circuits may scale better with voltage and thus have different results.

7.4.5 Policy Effectiveness Evaluation

Our dynamic policy adopts a pessimistic approach by disabling all possible power states that will violate the reliability target if the states last for the entire remaining time, i.e., $t_{end} - t$ in Equation 7.4. This can result in left-over SER budget and unnecessary power overhead. To study the effectiveness of our policies, we also implement a trace-recording module, which can be enabled to store the all power state switching information. With these traces, we can characterize the optimal power state selection scheme offline and compare with our policies. Since our purpose is evaluating the effectiveness of our policies rather than evaluating the built-in Linux governors, we will use the power states suggested by the Linux governors as baseline. Only the same or higher power states, i.e., higher voltage states or shallower idle states are considered so that we won't have any performance impacts.

Considering the fact that P-state selection will change the idle time length and corresponding C-state selection contexts, we will implement two separate schemes for P-state and C-state selection.

To find the optimal C-state selection, we formulate the problem as an integer linear programming (ILP) problem as in Equation 7.5:

$$
\begin{aligned}
\text{minimize:} \quad & \sum_{n=1}^{N} (T_n \times \mathbf{P}_n{}^{t} \cdot \mathbf{S}_n) \\
\text{subject to:} \quad & \forall\, n : \mathbf{S}_n{}^{t} \cdot \mathbf{1} \geq 1 \\
& \sum_{n=1}^{N} (T_n \times \mathbf{R}_n{}^{t} \cdot \mathbf{S}_n) \leq K \\
& \forall\, n : \mathbf{S}_n{}^{t} \cdot \mathbf{L}_n \leq 0 \\
& \forall\, n : \mathbf{S}_n \geq \mathbf{0}
\end{aligned}
\tag{7.5}
$$

The optimization objective is to minimize the energy overhead, where n is the index for each C-state selection, N is the total number of C-state selections. T_n is the actual resident time for the C-state selection. Since we have four idle states here, $\mathbf{P}_n$ is a 4x1 vector corresponding to the power of each C-state. The power can be different for different n because the corresponding P-state can be different. $\mathbf{S}_n$ is selection vector, i.e., 4x1 integer vector for each C-state selection. The i-th entry in $\mathbf{C}_n$ equals 1 means the i-th C-state is selected. The first constraint ensures that one state is selected[3]. K is the SER budget for these idle time selection. The second constraint is to make sure that SER budget will not be exceeded. $\mathbf{R}_n$ is a 4x1 vector corresponding to the SER of each C-state. This SER can also be different for different n because of the corresponding P-state. $\mathbf{L}_n$ is a 4x1 vector representing the selection of built-in Linux governor. If the i-th C-state is selected by Linux governor, all entries including and after the i-th entry are 1 and 0 otherwise. So the third and fourth constraints are used to limit the available state selection.

For P-state selection, we will first calculate the total active time until next P-state switching time according to the operating frequency, then scale the idle time spent in each C-state accordingly. This actually is likely to be optimistic because the optimal scheme will prefer a

[3]An upper bound here is unnecessary due to the minimization and the fact that $\mathbf{P}_n$ and T_n are non-negative

lower voltage state compared to our policies, which can result in less active time within the scheduling ticks and thus entering more into the shallower idle states.

For each P-state, we can first compute the new active time and idle time of each C-state. The we can characterize the energy and SER impacts for all P-states. To find the optimal P-state selection, we can also formulate the problem as an Integer Linear Programming (ILP) problem in Equation 7.6.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{m=1}^{M} (\mathbf{E}_m{}^t \cdot \mathbf{S}_m) \\
\text{subject to:} \quad & \forall\, m : \mathbf{S}_m{}^t \cdot \mathbf{1} \geq 1 \\
& \sum_{m=1}^{M} (\mathbf{R}_m{}^t \cdot \mathbf{S}_m) \leq K \\
& \forall\, m : \mathbf{S}_m{}^t \cdot \mathbf{L}_m \leq 0 \\
& \forall\, m : \mathbf{S}_m \geq \mathbf{0}
\end{aligned} \tag{7.6}$$

where m is the index for each P-state selection, M is the total number of P-state selections. Since we have five P-states, $\mathbf{E}_m$ and $\mathbf{R}_m$ are 5x1 vectors and corresponding to the pre-characterized energy and SER difference. Similar to C-state formulation, S_m is the 5x1 selection vector for P-states. K is the SER budget.

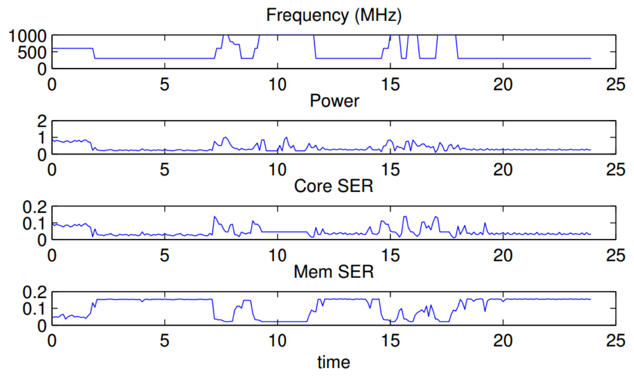
The formulated ILP problems are solved by LP_solve [lp]. The comparison results for software benchmarks running at altitude of 3000m are summarized in Table 7.5. Over all cases, our dynamic policy is within 1.5% with the optimal P-state selection scheme and within 3% with the optimal C-state selection scheme. The difference is smaller for P-state because the overhead of selecting higher voltage state can be compensated by spending more time in idle state, i.e., JIT vs. RFTS. But selecting shallower idle state will not change the utilization and thus the P-state selection. This also supports our claim that different power management schemes (i.e., JIT and RFTS) can be exploited for achieving system reliability targets.

Table 7.5: The additional power of running the benchmarks with our policies compared to the optimal P-state and C-state selection schemes

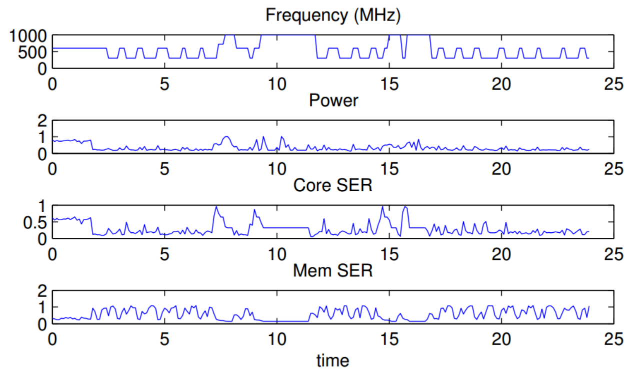
Benchmark	P-state optimal		C-state optimal	
	Mean	Max	Mean	Max
Web browsing	0.08%	0.42%	0.72%	2.18%
MPEG 4 decoding	0.14%	0.73%	0.63%	2.47%
FFPlay	0.18%	1.31%	0.58%	2.51%
3D Image rendering	0.14%	1.17%	0.53%	1.89%

7.5 Chapter Conclusion

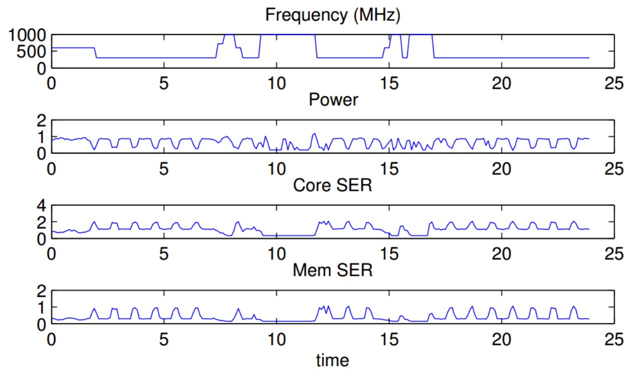
In this chapter, we developed a hardware evaluation platform to assess system reliability under different system power management schemes. We also propose a system/cloud-based virtual sensing for reliability evaluation. We propose two reliability management policies based on the already existing power management schemes. Experimental results with real life benchmarks show that our policy is effective and achieves system reliability target with minimal power overhead, compared to the optimal schemes characterized offline. Future work will try to include more reliability issues. The kernel module code is available for download at https://github.com/nanocad-lab/JIT-RFTS.



(a) sea level



(b) 2500m altitude Case I



(c) 2500m altitude Case III

Figure 7.12: An example of web browsing benchmark execution with different altitude and hardware cases. For case I, P-states are exploited by the policy while C-states are used for case III. Note some y-axis scale for the figures are different.

CHAPTER 8

Conclusion

In this dissertation, we have proposed cross-layer methodologies aiming at reducing the system design margin. The studied approaches include monitoring, margining and mitigation of circuit variability. Two circuit performance monitors are proposed. A complete monitor-and-actuate system for soft error is presented. Margining methodologies for dynamic variation and the "dark silicon" era are described. A clock-gating methodology is proposed to mitigate aging-induced clock skew.

The proposed methodologies reduce the design margin required for different sources of variability. Estimates of the performance margin and the reduction from our approaches based on 45nm process technology are shown in Fig. 8.1. The margin and reduction numbers are derived based on the following assumptions:

- **Wear-out Aging**: Wear-out aging is mainly considering the delay degradation due to BTI on the logic datapath. The amount of wear-out aging (i.e., 10%) is derived based on the BTI model from the design manual of a commercial 45nm process technology. The reduced margin is derived assuming the use of *SlackProbe* with 5% delay margin. The dark silicon margining work can reduce the margin by another 20% at 60% dark ratio.

- **Ambient Fluctuations**: Ambient fluctuations is mainly considering the IR drop in power distribution network. The amount of ambient fluctuations (i.e., 10%) is derived based on the typical sign-off voltage corners from 45nm commercial libraries. The reduced margin is derived assuming the use of *SlackProbe* with 5% delay margin. The dark silicon margining work can help reduce 40-60% of the metal width margin, but it is not included in the reduction of delay margin.

- **Process Variation**: The amount of process variation (i.e., 15%) is derived based on the measurement of our 45nm testchips. We assumed the use of *DDRO*, which can reduce the mean delay overestimation to 3% based on the simulation results.

- **Clock Uncertainty**: The clock uncertainty is mainly considering the local variation and BTI-induced clock skew in the clock distribution networks. The amount of clock uncertainty (i.e., 5%) is based on the amount of local variation of clock buffer and the amount of BTI-induced clock skew from *BTI-Gater*. The margin is reduced to 3% if *BTI-Gater* is used to mitigate the aging-induced clock skew.

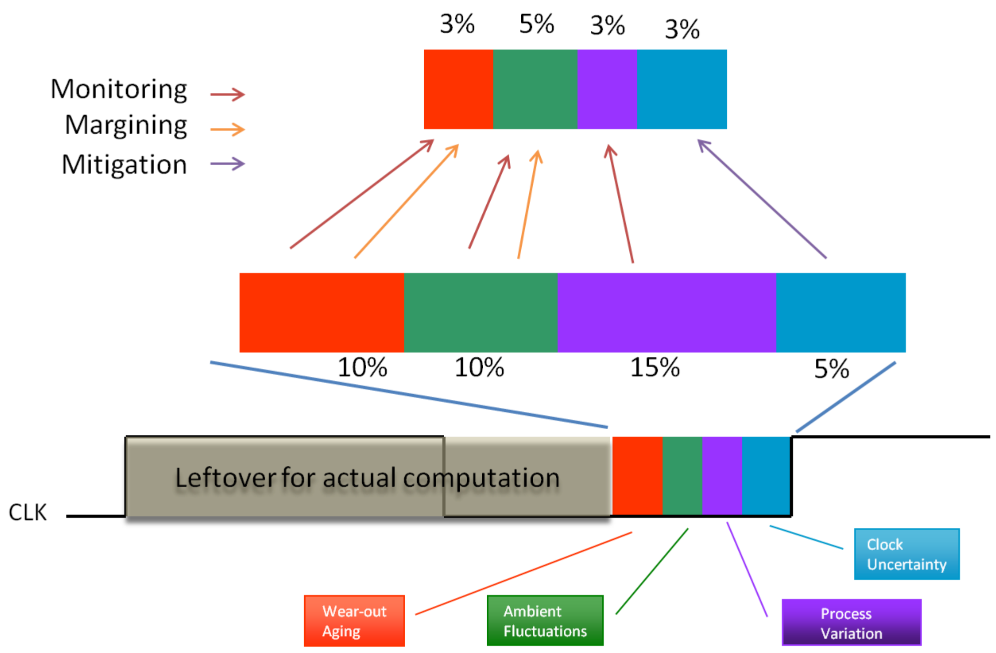


Figure 8.1: Estimate of the margin of different variability sources and projected margin saving from our approaches for 45nm process technology

References

[acp] "Advanced Configuration and Power Interface Specification." http://acpi.info/spec.htm.

[Aga] Yuvraj Agarwal et al. "RedCooper: Hardware Sensor Enabled Variability Software Testbed for Lifetime Energy Constrained Application." Technical report. http://escholarship.org/uc/item/1c21g217.

[Aga07] M. Agarwal et al. "Circuit Failure Prediction and Its Application to Transistor Aging." In *IEEE VLSI Test Symposium*, may 2007.

[Ait09] André Luiz Aita et al. "A CMOS smart temperature sensor with a batch-calibrated inaccuracy of±0.25 C (3σ) from- 70 C to 130 C." In *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pp. 342–343. IEEE, 2009.

[Ale11] Dan Alexandrescu. "A comprehensive soft error analysis methodology for SoCs/ASICs memory instances." In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pp. 175–176. IEEE, 2011.

[am3a] "am335x." http://www.ti.com/product/am3358.

[AM3b] "AM335xSDK." http://softwaredl.ti.com/sitara_linux/esd/AM335xSDK/-latest/index _FDShtml.

[B11] Keith A Bowman, , et al. "All-digital circuit-level dynamic variation monitor for silicon debug and adaptive clock control." *Circuits and Systems I: Regular Papers, IEEE Transactions on*, **58**(9):2017–2025, 2011.

[Ban01] Kaustav Banerjee et al. "Coupled analysis of electromigration reliability and performance in ULSI signal nets." In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pp. 158–164. IEEE Press, 2001.

[BBB] "BeagleBone Black." http://beagleboard.org/Products/BeagleBone+Black.

[Ben96] L. Benini et al. "Automatic synthesis of low-power gated-clock finite-state machines." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **15**(6):630–643, 1996.

[Bha06] S. Bhardwaj et al. "Predictive modeling of the NBTI effect for reliable design." In *IEEE Custom Integrated Circuits Conference*, pp. 189–192. IEEE, 2006.

[Bhu06] M. Bhushan et al. "Ring oscillators for CMOS process tuning and variability control." *IEEE Transactions on Semiconductor Manufacturing*, **19**(1):10–18, 2006.

[Bla69] James R Black. "ElectromigrationA brief survey and some recent results." *Electron Devices, IEEE Transactions on*, **16**(4):338–347, 1969.

[Bow09] K.A. Bowman et al. "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance." *IEEE Journal of Solid State Circuits*, jan. 2009.

[C14] Tuck-Boon Chan, , et al. "Synthesis and Analysis of Design-Dependent Ring Oscillator (DDRO) Performance Monitors." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, **22**(10):2117–2130, Oct 2014.

[CA08] Vikas Chandra and Robert Aitken. "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS." In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pp. 114–122. IEEE, 2008.

[Cad]

[Cal09] Andrea Calimera et al. "NBTI-aware power gating for concurrent leakage and aging optimization." In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pp. 127–132. ACM, 2009.

[Cha09a] A. Chakraborty et al. "Analysis and optimization of NBTI induced clock skew in gated clock trees." In *IEEE/ACM Design, Automation and Test in Europe*, pp. 296–299, 2009.

[Cha09b] James Charles et al. "Evaluation of the Intel Core i7 Turbo Boost feature." In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 188–197. IEEE, 2009.

[Cha11] Tuck-Boon Chan et al. "On the efficacy of NBTI mitigation techniques." In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pp. 1–6. IEEE, 2011.

[Cha12a] T. Chan et al. "DDRO: A novel performance monitoring methodology based on design-dependent ring oscillators." In *IEEE International Symposium on Quality Electronic Design*, march 2012.

[Cha12b] T. Chan et al. "Tunable sensors for process-aware voltage scaling." In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 7–14. ACM, 2012.

[Cha13a] A. Chakraborty et al. "Skew management of NBTI impacted gated clock trees." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **32**(6):918–927, 2013.

[Cha13b] T. Chan et al. "Synthesis and Analysis of Design-Dependent Ring Oscillator (DDRO) Performance Monitors." *IEEE Transactions on Very Large Scale Integration Systems*, 2013. Accpted for publication.

[Cha13c] Tuck-Boon Chan et al. "Impact of adaptive voltage scaling on aging-aware signoff." In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, pp. 1683–1688. IEEE, 2013.

[Che05] Poki Chen et al. "A time-to-digital-converter-based CMOS smart temperature sensor." *Solid-State Circuits, IEEE Journal of*, **40**(8):1642–1648, 2005.

[Che13] J. Chen et al. "A novel flow for reducing clock skew considering NBTI effect and process variations." In *IEEE International Symposium on Quality Electronic Design*, pp. 327–334, 2013.

[Cho13] Hyungmin Cho et al. "Quantitative evaluation of soft error injection techniques for robust system design." In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–10. IEEE, 2013.

[Cho14] M. Choudhury et al. "Time-Borrowing Circuit Designs and Hardware Prototyping for Timing Error Resilience." *Computers, IEEE Transactions on*, **63**(2):497–509, 2014.

[Chu11] Ching-Che Chung et al. "An autocalibrated all-digital temperature sensor for on-chip thermal monitoring." *Circuits and Systems II: Express Briefs, IEEE Transactions on*, **58**(2):105–109, 2011.

[Con10] J. Cong et al. "Behavior-level observability analysis for operation gating in low-power behavioral synthesis." *ACM Transactions on Design Automation of Electronic Systems*, **16**(1):4, 2010.

[CoO]

[cora] "ARM Cortex-A9 Technical Reference Manual." http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[Corb] "Cortex-M0." http://www.arm.com/products/processors/cortex-m/cortex-m0.php.

[Corc] "Cortex-M3." http://www.arm.com/products/processors/cortex-m/cortex-m3.php.

[Cos08] Ayse Kivilcim Coskun et al. "Proactive temperature balancing for low cost thermal management in MPSoCs." In *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pp. 250–257. IEEE, 2008.

[cpua] "cpufreq." https://www.kernel.org/doc/Documentation/cpu-freq/.

[cpub] "cpuidle." https://www.kernel.org/doc/Documentation/cpuidle/.

[Cue11] David Cuesta et al. "Adaptive task migration policies for thermal control in mpsocs." In *VLSI 2010 Annual Symposium*, pp. 83–115. Springer, 2011.

[D13] Alan J Drake, , et al. "Single-cycle, pulse-shaped critical path monitor in the POWER7+ microprocessor." In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pp. 193–198. IEEE, 2013.

[Das09] S. Das et al. "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance." *IEEE Journal of Solid State Circuits*, jan. 2009.

[Das10] B.P. Das et al. "Warning Prediction Sequential for Transient Error Prevention." In *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, oct. 2010.

[Dra07] A. Drake et al. "A Distributed Critical-Path Timing Monitor for a 65nm High-Performance Microprocessor." In *Proc. IEEE International Solid State Circuits Conference*, feb. 2007.

[Ebr14] Mojtaba Ebrahimi et al. "Comprehensive analysis of alpha and neutron particle-induced soft errors in an embedded processor at nanoscales." In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–6. IEEE, 2014.

[Eir07] M. Eireiner et al. "In-Situ Delay Characterization and Local Supply Voltage Adjustment for Compensation of Local Parametric Variations." *IEEE Journal of Solid State Circuits*, july 2007.

[enc] "Cadence Encounter Digital Implementation System." http://www.cadence.com/products/di/edi_system/pages /default.aspx.

[Ern03] D. Ernst et al. "Razor: a low-power pipeline based on circuit-level timing speculation." In *IEEE/ACM International Symposium on Microarchitecture*, dec. 2003.

[Esm11] Hadi Esmaeilzadeh et al. "Dark silicon and the end of multicore scaling." In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pp. 365–376. IEEE, 2011.

[ffm] "ffmpeg." https://www.ffmpeg.org/.

[FHL15] Ming Fan, Qiushi Han, Shuo Liu, and Gang Quan. "On-line reliability-aware dynamic power management for real-time systems." In *Quality Electronic Design (ISQED), 2015 16th International Symposium on*, pp. 361–365. IEEE, 2015.

[Fic10] D. Fick et al. “In situ delay-slack monitor for high-performance processors using an all-digital self-calibrating 5ps resolution time-to-digital converter.” In *Proc. IEEE International Solid State Circuits Conference*, feb. 2010.

[Foj13] M. Fojtik et al. “Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction.” *Solid-State Circuits, IEEE Journal of*, **48**(1):66–81, 2013.

[Fuk09] H. Fuketa et al. “Adaptive performance compensation with in-situ timing error prediction for subthreshold circuits.” In *IEEE Custom Integrated Circuits Conference*, sept. 2009.

[G07] Meeta S Gupta, , et al. “Understanding voltage variations in chip multiprocessors using a distributed power-delivery network.” In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE’07*, pp. 1–6. IEEE, 2007.

[Gor04] MS Gordon et al. “Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground.” *Nuclear Science, IEEE Transactions on*, **51**(6):3427–3434, 2004.

[Gup12] P. Gupta et al. “Underdesigned and opportunistic computing in presence of hardware variability.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012. Keynote Paper.

[Hic08] J. Hicks et al. “45nm transistor reliability.” *Intel Technology Journal*, **12**(2):131–144, 2008.

[Hir12] K. Hirairi et al. “13% Power reduction in 16b integer unit in 40nm CMOS by adaptive power supply voltage control with parity-based error prediction and detection (PEPD) and fully integrated digital LDO.” In *Proc. IEEE International Solid State Circuits Conference*, feb. 2012.

[Hua06] Wei Huang et al. “HotSpot: A compact thermal modeling methodology for early-stage VLSI design.” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, **14**(5):501–513, 2006.

[Hua13] S. Huang et al. “Low-power anti-aging zero skew clock gating.” *ACM Transactions on Design Automation of Electronic Systems*, **18**(2):27, 2013.

[Hun97] William R Hunter. “Self-consistent solutions for allowed interconnect current density. II. Application to design guidelines.” *Electron Devices, IEEE Transactions on*, **44**(2):310–316, 1997.

[int] “Intel Core i7-920XM Processor Extreme Edition.” http://ark.intel.com/products/43126.

[Isc06] Canturk Isci et al. “An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget.” In *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, pp. 347–358. IEEE Computer Society, 2006.

[itr09] “International Technology Roadmap for Semiconductors(ITRS).” http://www.itrs.net/, 2009.

[Kah13] Andrew B Kahng et al. “On potential design impacts of electromigration awareness.” In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pp. 527–532. IEEE, 2013.

[Kar06] Eric Karl et al. “Reliability modeling and management in dynamic microprocessor-based systems.” In *Proceedings of the 43rd annual Design Automation Conference*, pp. 1057–1060. ACM, 2006.

[Kar09] Ulya R Karpuzcu et al. “The BubbleWrap many-core: popping cores for sequential acceleration.” In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 447–458. IEEE, 2009.

[Kea07] M. Keating et al. *Low Power Methodology Manual: For System on Chip Design.* Springer, 2007.

[Kim13] S. Kim et al. “Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm SOI CMOS.” In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pp. 264–265. IEEE, 2013.

[Kum03] Rakesh Kumar et al. “Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction.” In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp. 81–92. IEEE, 2003.

[Kum06] Sanjay V Kumar et al. “Impact of NBTI on SRAM read stability and design for reliability.” In *Quality Electronic Design, 2006. ISQED'06. 7th International Symposium on*, pp. 6–pp. IEEE, 2006.

[Kur10] M. Kurimoto et al. “Phase-adjustable error detection flip-flops with 2-stage hold-driven optimization, slack-based grouping scheme and slack distribution control for dynamic voltage scaling.” *ACM Trans. Des. Autom. Electron. Syst.*, 2010.

[L13] Charles R Lefurgy, , et al. “Active guardband management in power7+ to save energy and maintain reliability.” *Micro, IEEE*, **33**(4):35–45, 2013.

[L14a] Liangzhen Lai, , et al. “Accurate and inexpensive performance monitoring for variability-aware systems.” In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pp. 467–473, Jan 2014.

[L14b] Liangzhen Lai, , et al. "SlackProbe: A Flexible and Efficient In Situ Timing Slack Monitoring Methodology." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **33**(8):1168–1179, Aug 2014.

[Lai13] L. Lai et al. "SlackProbe: A low overhead in situ on-line timing slack monitoring methodology." In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 282–287, 2013.

[Lee12] J. Lee et al. "ECO cost measurement and incremental gate sizing for late process changes." *ACM Transactions on Design Automation of Electronic Systems*, **18**(1):16, 2012.

[Li03] H. Li et al. "Deterministic clock gating for microprocessor power reduction." In *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on*, pp. 113–122. IEEE, 2003.

[Liu10] Q. Liu et al. "Capturing post-silicon variations using a representative critical path." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **29**(2):211–222, 2010.

[Liu12] W. Liu et al. "NBTI effects on tree-like clock distribution networks." In *Proceedings of the great lakes symposium on VLSI*, pp. 279–282. ACM, 2012.

[lp] "lp_solve." http://lpsolve.sourceforge.net/.

[LRK09] Man-Lap Li, Pradeep Ramachandran, Ulya R Karpuzcu, Siva Hari, and Sarita V Adve. "Accurate microarchitecture-level fault modeling for studying hardware faults." In *High Performance Computer Architecture. IEEE International Symposium on*, pp. 105–116. IEEE, 2009.

[Mah10] NN Mahatme et al. "Analysis of soft error rates in combinational and sequential logic and implications of hardening for advanced technologies." In *Reliability Physics Symposium (IRPS), 2010 IEEE International*, pp. 1031–1035. IEEE, 2010.

[mbe]

[Mem08] Seda Ogrenci Memik et al. "Optimizing thermal sensor allocation for microprocessors." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **27**(3):516–527, 2008.

[Mer14] Pietro Mercati et al. "A Linux-governor based Dynamic Reliability Manager for android mobile devices." In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pp. 1–4. IEEE, 2014.

[Min11] E. Mintarno et al. "Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, **30**(5):760–773, 2011.

[Min13] E. Mintarno et al. "Workload dependent NBTI and PBTI analysis for a sub-45nm commercial microprocessor." In *IEEE InternationalReliability Physics Symposium (IRPS)*, pp. 3A.1.1–3A.1.6, 2013.

[Mit07] S. Mitra et al. "Built-In Soft Error Resilience for Robust System Design." In *Integrated Circuit Design and Technology, 2007. ICICDT '07. IEEE International Conference on*, pp. 1–6, May 2007.

[Muk03] Shubhendu S Mukherjee et al. "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor." In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, p. 29. IEEE Computer Society, 2003.

[ope] "opencores." http://opencores.org.

[Pop07] Paul Pop et al. "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems." In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pp. 233–238. ACM, 2007.

[QZA11] Xuan Qi, Dakai Zhu, and Hakan Aydin. "Global scheduling based reliability-aware power management for multiprocessor real-time systems." *Real-Time Systems*, **47**(2):109–142, 2011.

[R09] Vijay Janapa Reddi, , et al. "Voltage emergency prediction: Using signatures to reduce operating margins." In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 18–29. IEEE, 2009.

[RCK15] S. Rehman, Kuan-Hsun Chen, F. Kriebel, A. Toma, M. Shafique, Jian-Jia Chen, and J. Henkel. "Cross-Layer Software Dependability on Unreliable Hardware." *Computers, IEEE Transactions on*, **PP**(99):1–1, 2015.

[Reb10] B. Rebaud et al. "Digital timing slack monitors and their specific insertion flow for adaptive compensation of variabilities." In *International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation*, PATMOS'09, 2010.

[Res01] P. J Restle et al. "A clock distribution network for microprocessors." *IEEE Journal of Solid State Circuits*, **36**(5):792–799, 2001.

[Res04] P. Restle et al. "Timing uncertainty measurements on the Power5 microprocessor." In *Proc. IEEE International Solid State Circuits Conference*, pp. 354–355. IEEE, 2004.

[RKS14] Semeen Rehman, Florian Kriebel, Duo Sun, Muhammad Shafique, and Jörg Henkel. “dTune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects.” In *Proceedings of the 51st Annual Design Automation Conference*, pp. 1–6. ACM, 2014.

[RMD07] Tajana Simunic Rosing, Kresimir Mihic, and Giovanni De Micheli. “Power and reliability management of SoCs.” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, **15**(4):391–403, 2007.

[Rot12] Efraim Rotem et al. “Power-management architecture of the intel microarchitecture code-named sandy bridge.” *IEEE Micro*, (2):20–27, 2012.

[Sat07] T. Sato et al. “A Simple Flip-Flop Circuit for Typical-Case Designs for DFM.” In *IEEE International Symposium on Quality Electronic Design*, march 2007.

[Sch03] D. K Schroder et al. “Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing.” *Journal of Applied Physics*, **94**(1):1–18, 2003.

[Shi02] Premkishore Shivakumar et al. “Modeling the effect of technology trends on the soft error rate of combinational logic.” In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 389–398. IEEE, 2002.

[Ska03] Kevin Skadron et al. “Temperature-aware microarchitecture.” In *ACM SIGARCH Computer Architecture News*, volume 31, pp. 2–13. ACM, 2003.

[SM10] Ranjani Sridharan and Rabi Mahapatra. “Reliability aware power management for dual-processor real-time embedded systems.” In *Proceedings of the 47th Design Automation Conference*, pp. 819–824. ACM, 2010.

[Sri05] Jayanth Srinivasan et al. “Lifetime reliability: Toward an architectural solution.” *Micro, IEEE*, **25**(3):70–80, 2005.

[Tel95] G. E Téllez et al. “Activity-driven clock design for low power circuits.” In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, pp. 62–65. IEEE Computer Society, 1995.

[Tsc09] J. Tschanz et al. “Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance.” In *VLSI Circuits, Symposium on*, june 2009.

[Vel12] J.B. Velamala et al. “Statistical aging under dynamic voltage scaling: A logarithmic model approach.” In *IEEE Custom Integrated Circuits Conference*, pp. 1–4, 2012.

[Ven10] Ganesh Venkatesh et al. “Conservation cores: reducing the energy of mature computations.” In *ACM SIGARCH Computer Architecture News*, volume 38, pp. 205–218. ACM, 2010.

[Vis06] C. Visweswariah et al. “First-order incremental block-based statistical timing analysis.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **25**(10):2170–2180, 2006.

[Wan06] Nicholas J Wang et al. “ReStore: Symptom-based soft error detection in microprocessors.” *Dependable and Secure Computing, IEEE Transactions on*, **3**(3):188–201, 2006.

[Wan07] W. Wang et al. “An integrated modeling paradigm of circuit reliability for 65nm CMOS technology.” In *IEEE Custom Integrated Circuits Conference*, pp. 511–514. IEEE, 2007.

[Wan08] X. Wang et al. “Path-RO: a novel on-chip critical path delay measurement under process variations.” In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2008.

[Wan13a] L. Wanner et al. “Hardware Variability-Aware Duty Cycling for Embedded Sensors.” *IEEE Transactions on Very Large Scale Integration Systems*, **21**(6):1000–1012, 2013.

[Wan13b] Lucas Wanner et al. “VarEMU: an emulation testbed for variability-aware software.” In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, p. 27. IEEE Press, 2013.

[Wei84] R. P. Weicker. “Dhrystone: a synthetic systems programming benchmark.” *Commun. ACM*, October 1984.

[wik] “Intel Turbo Boost.” http://en.wikipedia.org/wiki/Intel_Turbo_Boost.

[Woo09] Kyoungho Woo et al. “Dual-DLL-based CMOS all-digital temperature sensor for microprocessor thermal monitoring.” In *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pp. 68–69. IEEE, 2009.

[Xie10] L. Xie et al. “Representative path selection for post-silicon timing prediction under variability.” In *Proc. ACM/IEEE Design Automation Conference*, june 2010.

[Zha14] Hongyan Zhang et al. “GUARD: Guaranteed reliability in dynamically reconfigurable systems.” In *Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference*, pp. 1–6. ACM, 2014.

[Zhu06] Dakai Zhu. "Reliability-aware dynamic energy management in dependable embedded real-time systems." In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pp. 397–407. IEEE, 2006.