UNIVERSITY OF CALIFORNIA, SAN DIEGO

Low-Power Integrated-Circuit Implementation Exploiting System and Application Information

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

in

Electrical Engineering (Computer Engineering)

by

Seokhyeong Kang

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Sujit Dey
Professor Bill Lin
Professor Tajana Simunic Rosing
Professor Dean Michael Tullsen

2013

DEDICATION

- *To my wife, Kyunglim, without whose love, encouragement and sacrifice this thesis would not have been finished.*
- *To my daughter, Eunwoo, for her lovely smiles and sweet hugs.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank all of the people who helped make this dissertation possible.

First and foremost, I am thankful to my advisor, Professor Andrew B. Kahng, whose guidance, encouragement and support enabled me to complete my Ph.D. course and this thesis. His enthusiasm and passion always stimulate me, and his attitude on research will remain as a priceless lesson in my life.

I certainly feel lucky to have worked with many bright colleagues in Professor Kahng's VLSI CAD Lab. I would like to thank former members Dr. Kwangok Jeong, Dr. Kambiz Samadi and Professor Hailong Yao, and current members Tuck-Boon Chan, Siddhartha Nath, Wei-Ting Chan, Vaishnav Srinivas, Jiajia Li, Ilgweon Kang and Hyein Lee. We had wonderful projects, and we spent great times together. I wish them the best in their future.

I would also like to thank my thesis committee members, Prof. Chung-Kuan Cheng, Prof. Sujit Dey, Prof. Bill Lin, Prof. Tajana Simunic Rosing and Prof. Dean Michael Tullsen for taking time out of their busy schedules to review and evaluate my research work. I am grateful for their valuable feedback.

Last, but not least, I would like to thank my parents, Shindong Kang and Kyungnyeo Park, parents-in-law, Taesil Kang and Gilja Seo, and the rest of my family for their endless love and encouragement. My wife, Kyunglim, whose love and sacrifice allowed me to finish this journey. My daughter, Eunwoo, who has made me always happy with her lovely smiles and sweet hugs. This thesis is dedicated to them.

The material in this thesis is based on the following publications.

- Chapter 3 is based on:
  - Andrew B. Kahng, **Seokhyeong Kang**, Hyein Lee, Igor L. Markov and Pankit Thapar, "High-Performance Gate Sizing with a Signoff Timer", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, to appear.
  - Jin Hu, Andrew B. Kahng, **Seokhyeong Kang**, Myung-Chul Kim and Igor L. Markov "Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing", *Proc. IEEE/ ACM International Conference on Computer-Aided Design*, 2012, pp. 233–239.
  - Andrew B. Kahng and **Seokhyeong Kang**, "Construction of Realistic Gate Sizing Benchmarks With Known Optimal Solutions", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 153–160.

- Chapter 4 is based on:
  - Andrew B. Kahng, **Seokhyeong Kang** and Bongil Park, "Active-Mode Leakage Reduction with Data-Retained Power Gating", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2013, pp. 1209–1214.
- Chapter 5 is based on:
  - Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(3) (2012), pp. 404–417.
  - Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Recovery-Driven Design: A Power Minimization Methodology for Error-Tolerant Processor Modules", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 825–830.
  - Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs", *Proc. International Symposium on High-Performance Computer Architecture*, 2010, pp. 119–129.
  - Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 825–831.
- Chapter 6 is based on:
  - Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Enhancing the Efficiency of Energy-Constrained DVFS Designs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2012).
- Chapter 7 is based on:
  - Wei-Ting Chan, Andrew B. Kahng and **Seokhyeong Kang**, "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits", *Proc. IEEE International Conference on Computer Design*, 2013, to appear.
  - Andrew B. Kahng and **Seokhyeong Kang**, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", *Proc. ACM/IEEE Design Automation Conference*, 2012, pp. 820–825.

- Chapter 8 is based on:
  - Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "Many-Core Token-Based Adaptive Power Gating", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2013).
  - Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "TAP - Token-Based Adaptive Power Gating", *Proc. International Symposium on Low Power Electronics and Design*, 2012, pp. 203–208.
  - Kwangok Jeong, Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "MAPG: Memory Access Power Gating", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2012, pp. 1054–1059.

My coauthors (Wei-Ting Chan, Dr. Jin Hu, Dr. Kwangok Jeong, Dr. Myung-Chul Kim, Professor Rakesh Kumar, Hyein Lee, Professor Igor L. Markov, Professor Tajana S. Rosing, Professor John Sartori, Dr. Richard Strong and Pankit Thapar, listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

1976	Born, Busan, South Korea
1999	B.Sc., Electrical Engineering, Pohang University of Science and Technology, Pohang, South Korea
2001	M.Sc., Electrical Engineering, Pohang University of Science and Technology, Pohang, South Korea
2008	Senior engineer, Samsung Electronics Co., Ltd., Suwon, South Korea
2012	C.Phil., Electrical Engineering (Computer Engineering), University of California, San Diego
2013	Ph.D., Electrical Engineering (Computer Engineering), University of California, San Diego

All papers coauthored with my advisor Prof. Andrew B. Kahng have authors listed in alphabetical order.

- Andrew B. Kahng, **Seokhyeong Kang**, Hyein Lee, Igor L. Markov and Pankit Thapar, "High-Performance Gate Sizing with a Signoff Timer", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, to appear.
- Wei-Ting Chan, Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits", *Proc. IEEE International Conference on Computer Design*, 2013, to appear.
- Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "Many-Core Token-Based Adaptive Power Gating", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2013).
- Andrew B. Kahng, **Seokhyeong Kang** and Hyein Lee, "Smart Non-Default Routing for Clock Power Reduction", *Proc. ACM/IEEE Design Automation Conference*, 2013.
- Andrew B. Kahng, **Seokhyeong Kang**, Hyein Lee, Siddhartha Nath and Jyoti Wadhwani, "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. IEEE System-Level Interconnect Prediction, 2013*.
- Andrew B. Kahng, **Seokhyeong Kang** and Bongil Park, "Active-Mode Leakage Reduction with Data-Retained Power Gating", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2013, pp. 1209–1214.

- Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Enhancing the Efficiency of Energy-Constrained DVFS Designs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2012).
- Jin Hu, Andrew B. Kahng, **SeokHyeong Kang**, Myung-Chul Kim and Igor L. Markov "Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 233–239.
- Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "TAP - Token-Based Adaptive Power Gating", *Proc. International Symposium on Low Power Electronics and Design*, 2012, pp. 203–208.
- Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(3) (2012), pp. 404–417.
- Andrew B. Kahng and **Seokhyeong Kang**, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", *Proc. ACM/IEEE Design Automation Conference*, 2012, pp. 820–825.
- Andrew B. Kahng and **Seokhyeong Kang**, "Construction of Realistic Gate Sizing Benchmarks With Known Optimal Solutions", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 153–160.
- Kwangok Jeong, Andrew B. Kahng, **Seokhyeong Kang**, Tajana S. Rosing and Richard Strong, "MAPG: Memory Access Power Gating", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2012, pp. 1054–1059.
- Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Recovery-Driven Design: A Power Minimization Methodology for Error-Tolerant Processor Modules", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 825–830.
- Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs", *Proc. International Symposium on High-Performance Computer Architecture*, 2010, pp. 119–129.
- Andrew B. Kahng, **Seokhyeong Kang**, Rakesh Kumar and John Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 825–831.

ABSTRACT OF THE DISSERTATION

Low-Power Integrated-Circuit Implementation Exploiting System and Application Information

by

Seokhyeong Kang

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2013

Professor Andrew B. Kahng, Chair

A primary design goal for VLSI systems is to achieve low energy consumption while maintaining high performance. Increasing thermal densities and the portability of emerging computing systems demand further reduction of design power. However, in integrated-circuit (IC) designs, there is a tradeoff between energy and performance, and the solution space for any given design is bounded by the lowest possible energy and the highest possible performance. To minimize energy consumption under performance constraints, we seek to optimize the design up to the Pareto frontier of energy versus performance. Many system- and design-level techniques have been introduced to extend the achievable energy-performance envelope. For low-power IC implementation, this thesis first explores traditional design methodologies, which include gate sizing optimization and power gating. Gate sizing is an effective approach to optimize the tradeoff of power and delay in VLSI design. A sensitivity-guided metaheuristics approach is presented for high-quality, large-scale gate sizing. The proposed gate sizing method can minimize the power (energy) consumption under the timing (performance) constraint. Power gating is one of the most effective solutions available to reduce leakage power. However, power gating has not been practically usable in an active mode. In this thesis, a data-retained power gating is presented to enable power gating of flip-flops during the active mode.

Extensions of the energy-performance envelope can be achieved with new system- and design-level techniques such as (i) error-tolerant design, (ii) dynamic voltage and frequency scaling (DVFS), (iii) approximate arithmetic design, and (iv) adaptive power gating. However, with the new system-level techniques for energy-efficient design, conventional CAD flows or designs may constrain or reduce the benefits realized from these techniques; hence, new design methodologies are necessary for each technique. The innovative techniques proposed in this thesis exploit the system and application information, and connect them into design optimization and physical implementation to enable more energy-efficient designs. In other words, if we have better communication between system design and chip implementation, we can improve the design quality in terms of the low energy consumption. First, error-tolerant design allows timing errors, so frequently exercised paths should be optimized to reduce the error rate of the design. This means that a function-aware design optimization is required for the error-tolerant design. Second, to minimize lifetime energy in DVFS design, operating scenarios should be considered and scenario-aware optimization is required. Third, for the approximate designs, a tradeoff between data accuracy and power reduction can be used. Finally, to make an adaptive-power gating, we should retain internal data and control wake-up overheads. In each of these four directions, this thesis proposes novel optimization and design flows which expand the achievable envelope of low-power, high-performance VLSI system design.

Chapter 1

Introduction

A primary design goal of the VLSI system is to achieve low energy consumption and high performance. Increasing thermal densities and the portability of emerging computing systems demand further reduction of design power. However, in integrated-circuit (IC) designs, there is a tradeoff between energy and performance, and the solution space for any given design is bounded by the lowest possible energy and the highest possible performance. To minimize energy consumption under performance constraints, we seek to optimize the design up to the Pareto frontier of energy versus performance.

1.1 The Energy versus Performance Envelope

In general, for a VLSI system, the faster (smaller) the computation (delay), the more energy it consumes. This observation points to the tradeoff between the performance (delay) versus energy (power) consumption. In [190] [249], an energy-delay efficiency metric which captures any tradeoff between the energy and the delay of design has been introduced. As shown in Figure 1.1, according to the tradeoff, all possible designs are bounded by the lowest possible energy and the highest possible performance. In the energy-delay space, a goal of the VLSI system design is to implement a minimum energy design at a given performance target and to optimize the design up to the Pareto frontier of energy versus performance.

Gate sizing is an effective approach to optimize the tradeoff of power and delay in VLSI design. Gate sizing seeks to determine design parameters (e.g., gate width and threshold voltage) so as to optimize timing, area and power of a circuit for each gate, subject to constraints. The gate sizing optimization has been extensively studied, and a number of heuristics have been

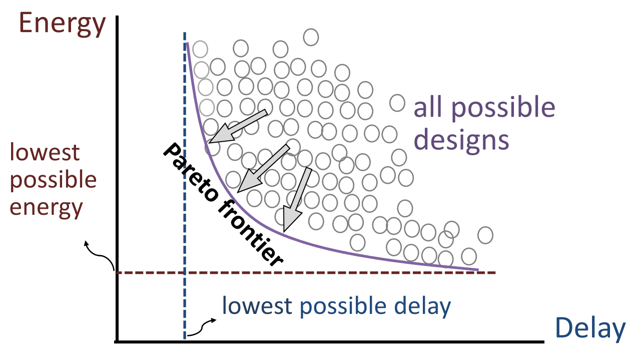


Figure 1.1: Energy versus performance envelope in VLSI design.

proposed. However, there have been no definitive comparisons (empirical or mathematical) of different techniques. Moreover, many published techniques make unrealistic assumptions about the underlying circuits, such as the possibility of continuous gate sizing and V_t assignment and the convexity of delay functions. Some publications fail to account for the effect of rounding when using a discrete gate library or do not account for realistic capacitance and slew constraints. Scalability to circuits with hundreds of thousands of gates is also an important issue, yet many publications use much smaller benchmarks mapped into outdated technologies.

To achieve an energy-efficient design, the use of clock gating and power gating to reduce dynamic power and static leakage, respectively, is well-understood by both researchers and IC designers [17]. *Clock gating* is considered to be one of the most effective techniques to reduce dynamic power, and its automatic application is supported by EDA tools [45]. Clock gating masks the clock signal when the corresponding circuits are not performing useful computations. *Power gating* [213] drastically reduces leakage power by introducing a switch between the voltage supply (and/or ground) and a given block of functional circuitry; the block's leakage is stopped when the switch cuts off the current path from supply to ground. To reduce active-mode leakage power, several approaches have been reported which combine clock gating and power gating [229] [49] [169] [160] [207]. However, these previous approaches have associated design complexity and overhead issues which limit their practical implementation.

1.2 Extending the Envelope with New Design Techniques

Extensions of the energy-performance envelope can be done with new system- and design-level techniques, such as (i) error-tolerant design, (ii) dynamic voltage and frequency

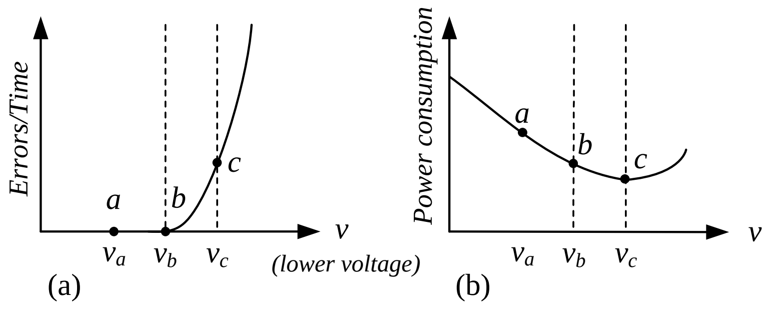


Figure 1.2: Error rate (a) and power consumption (b) versus voltage scaling in BTWC designs.

scaling (DVFS), (iii) approximate arithmetic design and (iv) adaptive power gating. However, with the new system-level techniques for energy-efficient design, conventional CAD flows or designs may constrain or reduce the benefits realized from the new system- and design-level techniques; hence, new design methodologies are necessary for each technique.

1.2.1 Error-Tolerant Design

The traditional goal of IC design is for the product to always operate correctly, even under worst-case combinations of non-idealities (process, voltage, temperature, wear-out, etc.). It is well-recognized that designing for worst-case operating conditions incurs considerable area, power and performance overheads [83], and that these overheads worsen with increased manufacturing or runtime variations in advanced technology nodes [15]. *Better-than-worst-case* (BTWC) design [41] allows reliability (in the sense of timing and, hence, functional correctness) to be traded off against performance and power. The central idea, as exemplified by the shadow-latch technique in *Razor* [83], is to design for average-case conditions (thus saving area and power) while adding an error detection and correction mechanism to handle errors that occur with worst-case variabilities.

Figure 1.2 illustrates power consumption under voltage scaling in BTWC designs. In the left plot, functional errors begin to occur below the voltage v_b, but we can reduce power consumption until we reach the voltage v_c, given the use of error correction. The right plot shows that below the voltage v_c, power consumption is increased because of recovery overhead.

The impact of BTWC design techniques is often limited in high-performance digital designs by a *critical operating point* or “wall of slack” phenomenon that limits voltage overscaling and, more importantly, is a direct consequence of today’s standard approach to power optimization. The Critical Operating Point (COP) hypothesis [189] (cf. Figure 1.2(a)), in the context of voltage scaling, states that a modern digital design will have a critical operating voltage V_c above which zero timing errors occur, and below which massive timing errors occur. The COP hypothesis is natural in light of how modern designs are optimized for power and area and subject to a frequency constraint: negative timing slack (on a combinational path) is cured by upsizing and buffering while positive timing slack is traded off for area and power reductions. Thus, in the final design, many timing paths are critical, and there is a “wall of slack”. The COP hypothesis states that overscaling beyond the critical voltage can abruptly cause error rates beyond what an error-tolerance mechanism can handle. According to [189], this has been confirmed in general-purpose microprocessors. A key motivation for our work is that COP behavior limits the applicability of voltage scaling in trading off reliability for power even in the context of BTWC design.

1.2.2 Dynamic Voltage and Frequency Scaling (DVFS)

Increasing thermal densities, reliability concerns, and the portability of emerging computing systems have made power and energy consumption primary concerns for microelectronic designers [15]. Dynamic voltage and frequency scaling (DVFS) [44] is a popular technique to reduce power and energy in situations where there is diversity in workloads (e.g., CPU-intensive versus memory-intensive, realtime versus non-realtime, etc.), compute conditions (e.g., wall-powered versus battery-powered, low system utilization versus high system utilization, etc.), or objective functions (e.g., single-thread performance versus throughput, energy versus energy-delay product, etc.) [150]. Diversity in operating conditions is exploited by reducing the supply voltage when performance constraints are relaxed. The effectiveness of DVFS at reducing power consumption is due to the strong dependence of both static and dynamic power on supply voltage.

To conserve energy, many DVFS-based mobile and embedded devices typically spend the majority of their lifetimes operating in a low-power mode. For example, Windows switches a DVFS-capable processor into a low-power mode whenever utilization is low [214]. A conventional multi-mode design methodology spends resources to optimize the critical paths in all modes and is therefore over-optimized at any single operating mode. Conventional design flows also require the user to specify all constraints at design time and disregard factors that are critical to optimizing energy efficiency, such as the amount of time spent in each mode.

1.2.3 Approximate Arithmetic

The approximate designs produce almost-correct results at the given required accuracy, and obtain power reductions or performance improvements in return. In some applications, however, more accurate or totally accurate results are required *under certain conditions*, e.g., image processing in security cameras would require cleaner images after detecting a motion. In contexts where the required accuracy changes during runtime, the accuracy of results should be configurable to maximize the benefit of approximate operations. Figure 1.3 illustrates how power benefits can be achieved with an accuracy-configurable design. The accuracy-configurable design can adapt to changing accuracy constraints by using different modes in each situation. To our knowledge, no previous work can configure the output accuracy during runtime, and each is thus restricted (or, best-suited) to particular application contexts. In contexts where the accuracy requirement can change dynamically, the previous methods' benefits from the accuracy tradeoff are reduced since the implementation must be targeted to the maximum accuracy requirement.

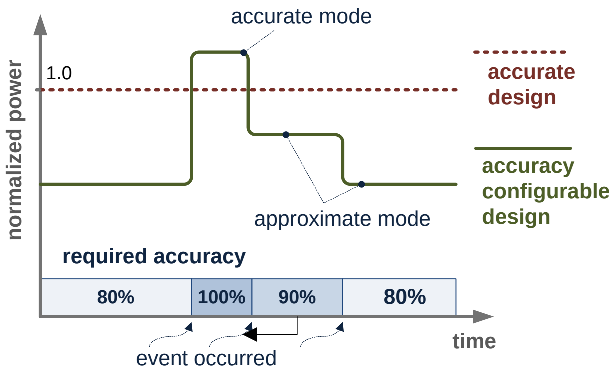


Figure 1.3: Power benefits from accuracy-configurable design.

1.2.4 Memory Access Power Gating

In mobile devices, operation time and peak processor performance are limited by battery capacity and chip thermal limits. These limits demand that all available power is used as efficiently as possible. However, a significant portion of power usage is leakage power. At the 32nm and 22nm technology nodes, leakage power ranges from 16.9% to 52.7% of total core power depending on circuit type, latency constraints, and temperature [162]. This leakage power translates into significant wasted energy if a core stalls waiting for a resource.

A core may stall quite often if it is intensively accessing the memory subsystem: every time a thread makes a memory request that misses in the L1 cache, the core is subjected to a variable access latency. This variable access latency often translates into a stall during which no forward thread progress occurs. Indeed, five of the SPEC 2006 [23] benchmarks (*GemsFDTD*, *gobmk*, *lbm*, *mcf*, and *milc*) spend more than 50% of their execution time waiting for the memory subsystem. Increased memory pressure in multi-core processors suggests that delays will become longer as more threads contend for the memory resource. Power-gating the core during a memory access can potentially mitigate costly leakage power dissipation during core stalls.

1.3 This Thesis

The innovative techniques proposed in this thesis exploit available system and application information, and creating new connections into design optimization and physical implementation to achieve more energy-efficient designs. First, an error-tolerant design allows timing errors, so frequently-exercised paths should be optimized to reduce the error rate of the design. This means that a function-aware design optimization is required for error-tolerant design. Second, to minimize lifetime energy in DVFS design, operating scenarios should be considered and scenario-aware optimization is required. Third, for the approximate designs, a tradeoff between data accuracy and power reduction can be used. Fourth, to make a memory access power gating, we should retain internal data and control wake-up overheads. In each of these four thrusts, this thesis proposes novel optimization and design flows, which expand the achievable power-performance envelope of VLSI system design, leading to a low-power integrated-circuit design. Figure 1.4 illustrates the scope and organization of this thesis.

The remainder of this thesis is organized as follows.

- Chapter 2 covers background and reviews prior works in the area of low-power design methodology. Two traditional literatures, on gate sizing optimizations and power gating techniques, are reviewed. In addition, four new system- and design-level techniques, (i) error-tolerant design, (ii) dynamic voltage and frequency scaling (DVFS), (iii) approximate arithmetic design, and (iv) adaptive power gating, are explored.
- Chapter 3 proposes a multi-threaded, stochastic optimization for gate sizing and V_t assignment to minimize leakage power subject to capacitance, slew and timing constraints. Scalability and solution quality of our sizer are validated on ISPD-2012/2013 Gate Sizing Contest benchmarks.

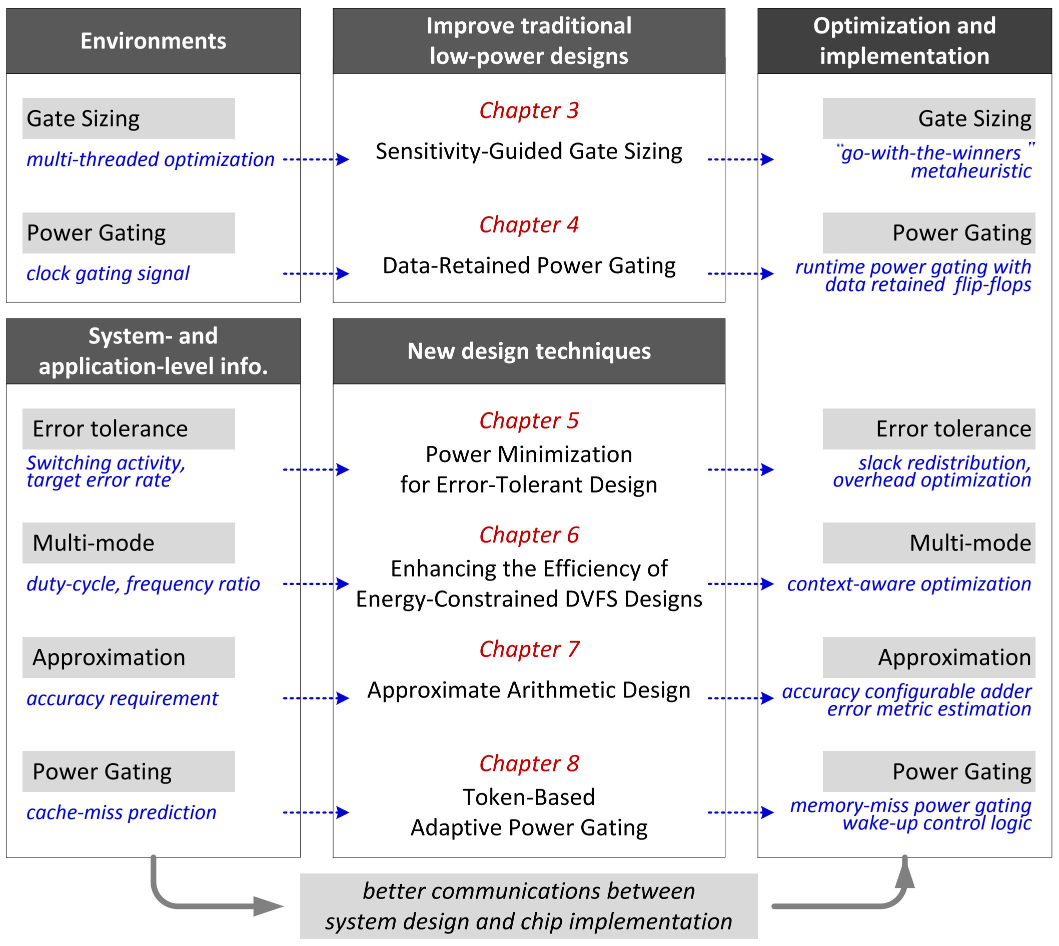

Environments
Improve traditional low-power designs
Optimization and implementation
Gate Sizing
multi-threaded optimization
Chapter 3
Sensitivity-Guided Gate Sizing
Gate Sizing
"go-with-the-winners" metaheuristic
Power Gating
clock gating signal
Chapter 4
Data-Retained Power Gating
Power Gating
runtime power gating with data retained flip-flops
System- and application-level info.
New design techniques
Error tolerance
Switching activity, target error rate
Chapter 5
Power Minimization for Error-Tolerant Design
Error tolerance
slack redistribution, overhead optimization
Multi-mode
duty-cycle, frequency ratio
Chapter 6
Enhancing the Efficiency of Energy-Constrained DVFS Designs
Multi-mode
context-aware optimization
Approximation
accuracy requirement
Chapter 7
Approximate Arithmetic Design
Approximation
accuracy configurable adder error metric estimation
Power Gating
cache-miss prediction
Chapter 8
Token-Based Adaptive Power Gating
Power Gating
memory-miss power gating wake-up control logic
better communications between system design and chip implementation

Figure 1.4: Scope and organization of this thesis.

- Chapter 4 proposes a *data-retained power gating* (DRPG) technique which enables power gating of flip-flops during active mode. We combine clock gating and power gating techniques, with the flip-flops being power gated during clock masked periods. We introduce a retention switch which retains data during the power gating. With the retention switch, correct logic states and functionalities are guaranteed without additional control circuitry.

- Chapter 5 proposes *recovery-driven design*, a design approach that optimizes a processor module for a target timing error rate rather than for correct operation. We show that significant power benefits are possible from a recovery-driven design flow that deliberately allows errors caused by voltage overscaling [108] [83] to occur during nominal operation, while relying on an error recovery technique to tolerate these errors. We present a detailed evaluation and analysis of such a CAD methodology that minimizes the power of a processor module for a target error rate.

- Chapter 6 explores the DVFS design space to identify the factors that affect DVFS efficiency. We propose two design-level techniques to enhance the energy efficiency of DVFS for energy constrained systems. First, we present a *context-aware DVFS design* flow that considers the intrinsic characteristics of the hardware design as well as the operating scenario – including the relative amounts of time spent in different modes, the range of performance scalability, and the target efficiency metric – to optimize the design for maximum energy efficiency. Second, we present a *selective replication-based DVFS design* methodology that identifies hardware modules for which context-aware multi-mode designs are energy-inefficient; such modules are candidates for replication. Selective replication creates module replicas for different operating modes.

- Chapter 7 proposes an *accuracy-configurable approximate (ACA)* adder for which the accuracy of results is configurable *during runtime*. Because of its configurability, the ACA adder can adaptively operate in both approximate (inaccurate) mode and accurate mode. The proposed adder can achieve significant throughput improvement and total power reduction relative to conventional adder designs. It can be used in accuracy-configurable applications, and improves the achievable tradeoff between performance/power and quality.

- Chapter 8 proposes a low-overhead technique to power gate an actively executing core during memory accesses. To this end, we design *Token-Based Adaptive Power Gating* (TAP), an architecture that provides two services. First, TAP tracks every system memory request and provides a lower-bound estimated time of arrival (ETA) for the response. Second, TAP tracks the state of every core in the system that can be power gated and provides a minimal-latency wake-up mode (with wake-up latency on the order of $2.4ns$ to $18.4ns$) to each core such that voltage drops on neighboring active cores will be within safety margins. Any power gating switch that interfaces with TAP is able to deterministically power gate its core and avoid any performance hit by not power gating for intervals that are shorter than the break-even duration for energy savings.

Chapter 2

Background and Prior Work

This chapter covers background and reviews prior works in the area of low-power design methodology. Two traditional literatures, on gate sizing optimizations and power gating techniques, are reviewed. In addition, four new system- and design-level techniques, (i) error-tolerant design, (ii) dynamic voltage and frequency scaling (DVFS), (iii) approximate arithmetic design, and (iv) adaptive power gating, are explored.

2.1 Traditional Low-Power Design Optimizations

For low-power IC implementation, Chapter 3 and 4 of this thesis focus on two traditional low-power design optimizations, gate sizing and power gating.

2.1.1 Gate Sizing

The problem of sizing standard cells in digital circuits has been extensively studied due to its importance, and many optimization techniques have been developed. Before discussing specific ideas, we introduce the optimization objectives and relevant constraints.

Gate Sizing Formulation. Consider a netlist $\mathcal{N} = (\mathcal{C}, \mathcal{I}, \mathcal{P})$, where $\mathcal{C}$ is the set of cells, $\mathcal{I}$ is the set of interconnects between cells, and $\mathcal{P}$ is the set of pins on $\mathcal{C}$. Also given is the input technology library $\mathcal{L}$, where each cell $c \in \mathcal{C}$ has a set of valid (manufacturable) sizes. Given $\mathcal{N}$ and $\mathcal{L}$, the objective of (discrete) gate sizing is to map $\mathcal{C}$ to $\mathcal{L}$ while minimizing the total power consumption subject to design constraints. We consider (i) slack constraints, (ii) capacitance constraints, and (iii) slew constraints.

Continuous Methods. The vast majority of gate sizing research has focused on finding *continuous* solutions, where parameters can take values within certain contiguous ranges. Some techniques optimize transistor parameters (e.g., threshold voltage), and others focus on entire standard cells and deal with drive strength and input-pin capacitances. However, as modern digital methodologies use only discrete cell types, continuous-valued parameters must be efficiently rounded to allowed discrete values.

Many different approaches to gate sizing have shown success, including:

- Linear programming [46] [62] [124] and network flow [199]
- Convex nonlinear optimization [67] [201] [202] [236],
  including Lagrangian relaxation [60] [67] [224] [236] [164]
- Slack budgeting [109]

Due to the complexity of modern designs, the two most scalable approaches are those based on *Lagrangian relaxation* and *sensitivity*. Instead of satisfying every imposed constraint, Lagrangian relaxation changes the original constrained problem into an unconstrained problem such that the solutions to the latter can be mapped back to the former. However, Lagrangian relaxation is typically formulated for continuous-variable functions and does not naturally handle discrete gate sizes. Numerical optimization techniques used with Lagrangian relaxation sometimes also assume convexity, which does not hold for practical circuit-delay models.

Discrete Methods. In contrast to continuous methods, discrete methods assume a fixed cell (and technology) library with a set of discrete cell sizes. While discrete methods avoid the difficult "rounding" problem that continuous methods face, they must directly solve an NP-hard problem [181]. Branch-and-bound [196] and dynamic programming [112] [164] [186] based approaches can be categorized as discrete.

Another discrete method, the sensitivity-based (iterative) approach, modifies individual components one at a time, e.g., by changing (i) transistor width, (ii) threshold voltage (in dual-V_t designs), (iii) source voltage (in dual-V_{dd} designs), and (iv) gate and/or wire resistances and capacitances. To further improve performance, the authors of [75] develop a multi-directional search, and the authors of [48] suggest using multistarts. Sensitivity-based downsizing approaches have been proposed in [86] [216] [222] [101] [100] [102]. *TILOS* [86] proposes a heuristic that sizes transistors iteratively, according to the sensitivity of the critical path delay to the transistor sizes (i.e., with maximum ratio of delay reduction / transistor width increase) in order to find an optimum. Equation (2.1) shows the sensitivity function of *TILOS*. ΔL and ΔD represent the change in leakage and delay for a resized transistor. The techniques proposed in [222] use the same sensitivity function as *TILOS*.

$$Sensitivity = \Delta L / \Delta D \tag{2.1}$$

For the cell sizing in [101], all cells are sorted in decreasing order of $\Delta L \times S$, where ΔL is the improvement in leakage after a cell is replaced with its less leaky variant, and S is its timing slack after the replacement has been made. The techniques proposed in [100] and [102] use sensitivity-based *downsizing* (i.e., begin with all nominal cell variants and replace cells on non-critical paths with long channel-length variants) heuristics for leakage optimization. In their heuristics, they define the sensitivity associated with a given cell instance as

$$Sensitivity = \Delta L / \Delta S \tag{2.2}$$

where ΔS represents the slack change of a given cell instance after downsizing, and ΔL indicates the leakage change of the cell instance after downsizing. The sensitivities are computed for all cell instances. The heuristics of [100] [102] select a cell with the largest sensitivity and downsize it to a logically equivalent cell. If there is no timing violation in incremental STA, this move is accepted and saved.

Recently, the authors of [186] have developed a Lagrangian-relaxation graph-based approach to efficiently assign cell types in very large netlists. Their optimizations have significantly improved power-performance tradeoffs in ICs designed at Intel. This work has also spurred the ISPD-2012 (Discrete) Gate Sizing Contest [184].

The ISPD-2012 Gate Sizing Contest. To more accurately capture the discrete gate sizing problem for modern technologies, researchers from Intel organized the ISPD-2012 Gate Sizing Contest [184]. The contest benchmarks are derived from the IWLS 2005 benchmarks [20] and have between 25K and 995K cells. The contest employs standard industry formats for benchmarks, including Verilog netlists, interconnect parasitics in IEEE SPEF format [14], and timing constraints in *Synopsys Design Constraints* (SDC) format [30]. The simplified standard-cell library implements 12 different logic functions, and 30 different cell types (options) for each logic function; the cell types correspond to three different threshold voltages (V_t) and ten different sizes for each V_t. The cell library also has lookup tables for (a) delay and transition time (slew), and (b) max load capacitance (in the *Synopsys Liberty* format) for each cell type. An industry timing engine – *Synopsys PrimeTime* [29] – is used as the reference timer. For simplicity, the contest does not capture false paths, placement-dependent distributed interconnect parasitic models, or multiple clock domains. The contest compares leakage power of solutions that satisfy all given constraints. Specific constraints include (i) no negative (setup) slack on all pins, (ii) a 300ps transition time upper bound, and (iii) a maximum load capacitance per cell type.

Stochastic Combinatorial Optimization. High-dimensional, hard combinatorial optimization problems do not admit such mathematically elegant solutions as Lagrangian relaxation, and are often solved using simulated annealing or other *metaheuristics* that combine local search with a global strategy [94] (e.g., stochastic hill-climbing, tabu search [93], or genetic crossover). These techniques are difficult to implement well (e.g., require heavy parameter tuning), are generally not reproducible, and require sophisticated mathematics to analyze their asymptotic performance.

The *go-with-the-winners* (GWTW) metaheuristic [39] repeatedly invokes greedy heuristics within randomized multistarts to (i) explore a large search space by maintaining a small set of best-seen solutions, and (ii) find a global optimum with high probability, as proven in [39]. Local minima are avoided when better intermediate solutions are found (e.g., in parallel search threads), from which multiple solutions may be derived. GWTW is asymptotically more efficient than purely independent randomized restarts; its performance is on par with that of simulated annealing and can be significantly easier to analyze [39].

The maintenance of the best-seen solutions in GWTW is akin to the "survival of the fittest" invariant in genetic algorithms [94]. However, GWTW does not employ genetic crossover, does not exclude repeat solutions, and allows the number of kept solutions to vary. GWTW can be viewed from the statistics perspective as *sequential importance sampling* and traces its roots to statistical physics [96]. In Chapter 3, we propose a gate sizing approach which adapts the GWTW metaheuristic [111].

2.1.2 Active-Mode Leakage Reduction

Power gating is the most effective available technique to reduce standby leakage, with benefits that are magnified by the increasing fraction of overall IC lifetime that modules spend in standby mode. With technology scaling, *active-mode leakage* becomes an increasingly significant portion of total dynamic power. Usami et al. [229] propose Run-Time Power Gating (RTPG) to extend the application of power gating to active-mode leakage reduction. Figure 2.1

shows the basic structure of RTPG. The enable signals of a gated clock design are exploited to control power switches for combinational logic gates. When the clock enable signal is 0, the power switch is turned off and active-mode leakage is cut off. The holders keep the input voltage of non-power-gated circuits.

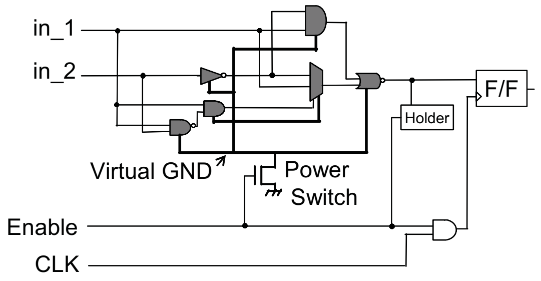


Figure 2.1: Basic structure of RTPG [229].

Several design (synthesis and layout) flows have been proposed for RTPG implementation. Bolzani et al. [49] present a synthesis flow to combine power gating and clock gating. They partition the circuit into a number of clusters that are clock-gated by the same registers. Li et al. [160] propose an activity-driven optimization for RTPG variant which integrates clock gating and power gating based on input data. Seomun et al. [207] provide a synthesis and physical design (placement) flow for RTPG circuits. Mistry et al. [175] present a RTPG which works concurrently with voltage and frequency scaling.

While RTPG can effectively reduce active-mode leakage power of combinational logic, the approach has several inherent limitations that hamper practical implementation. First, the RTPG approach significantly increases design complexity. Each cluster of gates requires its own control signal to control power gating transistors. Other overheads include special buffer trees that use the real power supply network, synthesis of high-fanout nets, power routing for the buffers, and so on. Further, the large number of virtual ground rails must be mutually isolated as well. Second, RTPG implementation incurs significant area overheads from its design complexity and additional circuits, e.g., bus holder circuits. Third, inrush current from power gating can diminish the amount of leakage reduction. If the clock-masked period is short or if flip-flop data is frequently changed, then RTPG will not be applicable due to the inrush current overhead.

Fukuoka et al. [89] present a clock gating scheme for partially-depleted SOI which controls V_t of each transistor by body biasing associated with the clock gating signal. Their approach reduces active-mode leakage of flip-flops with the dynamic body biasing. However, the body biasing technique requires significant design and area overheads from the biasing circuits and voltage regulators.

Kim et al. [142] have proposed a tri-mode power gating approach that provides a choice between a large leakage reduction without data retention (IDLE) and an intermediate level of leakage reduction with data retention (PARK). The authors of [142] add a single PMOS switch to an NMOS footer switch in parallel to provide the intermediate power-saving mode. However, their intermediate mode is applied to an entire submodule, and cannot be used for a fine-grained RTPG approach. In Chapter 4, we exploit the idea of an intermediate mode of power gating, and apply a similar approach within our proposed data-retained power gating approach.

2.2 New Techniques for Low-Power Design

Extensions of the energy-performance envelope can be achieved with new system- and design-level techniques. Chapters 5, 6, 7 and 8 of this thesis respectively focus on (i) error-tolerant design, (ii) dynamic voltage and frequency scaling (DVFS), (iii) approximate arithmetic design, and (iv) adaptive power gating.

2.2.1 Error-Tolerant Design

Error-tolerant designs trade off between design robustness and design quality (performance, power and area) [69] [77] [83] [91] [97] [221]. *Razor* [83] is a well-known technique to detect and correct timing errors due to frequency, temperature, and voltage variations. Razor detects timing violations by supplementing error-tolerant registers (*Razor flip-flops* – Figure 2.2) with *shadow latches*. A shadow latch strobes the output of a logic stage at a fixed delay after the main flip-flop; if a timing violation occurs, the main flip-flop and shadow latch will have different values, indicating the need for correction. Correction involves recovery using the correct value(s) stored in the shadow latch(es).

Error-tolerant designs reduce design constraints by allowing timing errors. An example is *timing speculation* [235], which increases the clock frequency and exploits error detection and recovery mechanisms to correct the resulting errors. Timing improvement from error-tolerant designs can lead to further power and area benefits over conventional designs. In other words, we can reduce the power and area of logic cells in a fanin cone by instantiating an error-tolerant register at the endpoint.

Error-Tolerant Registers with Error Recovery. As noted above, Razor and other related works [83] [77] [42] replace registers with specialized flip-flops which detect and correct timing errors on each endpoint by capturing the correct value at the shadow latches with a delayed

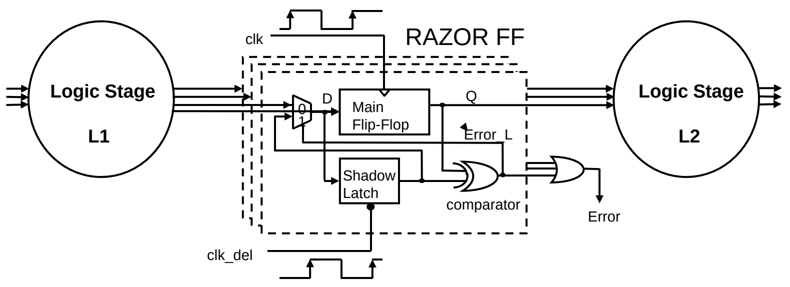


Figure 2.2: Structure of Razor flip-flop [83].

clock. The Razor technique [83] can correct timing errors within a specific safety margin of the error-tolerant register. *Razor II* [77] provides additional analyses of the Razor flip-flop – timing constraints, safety margin and clock scheme – and reduces complexity and area of the Razor flip-flop implementation. *STEM* [42] improves error detection capability with a second shadow latch. However, the duty cycle of the clock must be adjusted to avoid severe hold-time constraints introduced by the latch. *TIMBER* [69] masks temporal errors by borrowing time from successive pipeline stages.

Error Prediction Techniques. Error prediction techniques monitor transitions of data path signals before the clock edge. In [38], a stability checker design predicts timing errors due to a gradual increase in delay from wearout and aging effects. Another error prediction technique pads a given data path with a delay element and samples the delayed data path signal in another flip-flop, called the canary flip-flop [205]. A timing error is predicted when the value in the data path flip-flop differs from the value in the canary flip-flop. Error prediction based on duplicating critical paths and predicting timing errors on the original paths has been described in [50].

Replica Circuits for Error Masking. Several techniques compare output values in each cycle using redundant hardware circuits for error masking. *Paceline* [97] employs a leader-checker which checks timing errors due to overclocking. *CPipe* [221] enables reliable overclocking through core-replication. The outputs of the main combinational logic are compared with those of the duplicated logic in each cycle. This kind of approach can provide error resilience with high reliability, but typically has significant area and power overheads due to the redundant and duplicated logic circuits. Another approach, *CRISTA* [91], anticipates a timing error using an input pattern decoder; if critical paths are toggled by specific input patterns, the clock period is changed so that timing errors do not occur. Critical and non-critical paths are divided using a *path isolation* method. The CRISTA approach forces a significant design changes, and is difficult to apply to general processor designs.

Design-Level Optimization. In conjunction with error-tolerant techniques, a number of design-level optimizations [98] [91] [128] [129] have been proposed which identify and optimize critical paths that are frequently exercised during operation. However, such techniques in general do not consider the cost of the error-tolerant circuits during the optimization. *BlueShift* [98] identifies the most frequently violated timing paths during gate-level simulation, and optimizes the paths iteratively until the error rate is below a given target. To add timing slack to frequently-exercised paths, *BlueShift* uses two methods, forward body biasing of selected gates and application of tighter timing constraints to the frequently-exercised paths. Work on better than worst case (BTWC) logic synthesis [73] has also proposed to use activity information to reduce the error rate of an overscaled design. Whereas traditional synthesis tools attempt to minimize delay for a logic block, the proposed BTWC synthesis tool uses switching probability to break a tie when two equivalent logic decompositions have the same delay. Reducing switching activity can result in fewer errors for an overscaled design.

2.2.2 DVFS Design

Multi-Corner Multi-Mode Design. Recently, EDA manufacturers have included *Multi-Corner Multi-Mode (MCMM)* capabilities [28] [6] in their implementation tools. MCMM capabilities allow analysis and optimization of a hardware design for multiple modes, where modes are defined by a set of clocks, supply voltages, timing constraints, and libraries. MCMM is typically geared towards achieving design closure across all modes (e.g., test mode and mission modes) in a single pass, significantly reducing design turnaround times.

In Chapter 6 of this thesis, we show that variants of MCMM can be used for efficient DVFS designs, since MCMM can optimize a design for multiple voltage and frequency modes. However, capabilities must be added to identify the constraints that minimize energy for different duty cycles and ranges of scalability. We use MCMM sign off in our context-aware multi-mode design flow as well as in our baseline conventional design flow.

Heterogeneous CMP. DVFS is not the only technique to target multiple power and performance points. In a heterogeneous chip multi-processor (CMP) [150], each core can be designed for a different power and performance target, with tasks scheduled to cores depending on their performance or power requirements. Such designs incur significant overheads in terms of area, verification, complexity, and task scheduling. However, they may improve energy efficiency over DVFS designs by providing entire cores that are optimized for specific performance targets. The overheads associated with heterogeneous CMPs could potentially be reduced by replicating core functionalities at a finer granularity, rather than replicating the entire core (see Section 6.2.2).

Workload-Specific Data Path Customization. It has long been known that energy efficiency can be improved by customizing hardware for a specific workload. Work on conservation cores (C-cores) [232] proposes to synthesize application-specific cores to improve energy efficiency at a *specific* frequency and voltage. Application-specific cores can significantly reduce energy consumption for their target applications. The limitations of application-specific cores include increased design and verification overheads, limited generality beyond the applications and inputs for which the cores were synthesized, and significant area overhead for large applications (i.e., applications with multiple hot codes). Recent work suggests that the area overhead of conservation cores may be more acceptable if continued technology scaling results in a utilization wall, necessitating dark silicon [95].

Race-To-Halt. For some energy-based metrics and some specific (compute-intensive) workload conditions, an orthogonal approach to voltage scaling such as *race-to-halt* [79] may be useful for increasing energy efficiency. Race-to-halt proposes that it may be more energy-efficient to execute a task as quickly as possible, and then switch to a low-power sleep mode, than to operate at the lowest possible frequency and voltage that meets timing constraints. While race-to-halt can provide energy benefits over a naive DVFS approach in some scenarios, it suffers somewhat in generality (e.g., it does not benefit memory-intensive workloads). Also, while race-to-halt can potentially reduce energy, it cannot be used to reduce power due to thermal and reliability consequences. Note that DVFS does not preclude the deployment of a race-to-halt strategy: indeed, race-to-halt could well be an element of a task scheduling policy used by a DVFS processor that must, by its nature, be able to switch between different power and performance modes.

2.2.3 Approximate Design

Approximate Arithmetic Circuits. Guardbands for dynamic variations severely limit performance and energy efficiency of conventional IC designs. To overcome consequences of overdesign, several recent mechanisms for variation-resilient design [92] allow timing errors and manage design reliability dynamically. This and other relaxations of the requirement of correctness for designs have the potential to dramatically reduce costs of manufacturing, verification and test [15]. In resilient designs, errors can be corrected with redundancy techniques (*error-tolerance*), or accepted in some applications relating to human senses such as hearing and sight (*error-acceptance*). In the error-acceptance regime, approximation via a simplified or inaccurate circuit can increase performance and/or reduce power consumption.

Various approximate arithmetic designs have been previously proposed. Lu [166] introduces a faster adder which has shorter carry chains and considers only the previous *k* bits of input in computing a carry bit. Verma et al. [233] provide a variable-latency speculative adder ($VLSA$), which is a reliable version of the Lu adder [166] with error detection and correction. Shin et al. [211] propose a data path redesign technique for various adders which cuts the critical path in the carry chain. Zhu et al. [247] [246] propose three approximate adders – $ETAI$, $ETAII$ and $ETAIIM$. ETAI is divided into an accurate part and an inaccurate part to achieve approximate results. ETAII cuts carry propagation to speed up the adder, and ETAIIM modifies ETAII by connecting carry chains in accurate MSB parts. Kulkarni et al. [147] present a 2×2 underdesigned multiplier, and use it to build large power-efficient approximate multipliers. George et al. [90] define the concept of *probabilistic CMOS* (PCMOS), and implement efficient arithmetic using PCMOS. Shin et al. [212] propose a logic synthesis approach to design an approximate circuit.

Approximate Circuit Design. Recently, approximate computing has been explored as a means of improving energy efficiency for noise-tolerant applications. While approximate computation circuits have been shown to be effective at improving energy efficiency at the expense of perfect functional correctness, modern CAD tools are unable to perform design automation (e.g., approximate module replacement) for designs that contain approximate computation circuits. A necessary foundation for CAD tools that can create resource-efficient approximate designs is the ability to quickly and accurately estimate the output quality of designs whose composition includes approximate computation circuits. Such functionality is needed if CAD tools are to minimize the energy of a design during synthesis, optimization, etc. while maintaining acceptable output quality, as specified by system designers. In Chapter 7, we propose a flow that can analyze how errors originate and propagate in designs composed of approximate computation circuits to quickly and accurately estimate the output quality at nets in an approximate design. The following terminology is relevant to our treatment of approximate circuit design.

- *Error metric* (EM): a unit of measure that quantifies the deviation between the outputs produced by a functionally correct design and an approximate design. Below, we review several commonly-used EMs from the existing literature.
- *Approximate circuit*: a circuit that contains one or more approximate hardware modules. Figure 2.3 compares an approximate circuit to its accurate counterpart.
- *Composed EM* ($EM_{composed}$): the estimated EM value at an output or internal net in an approximate circuit.

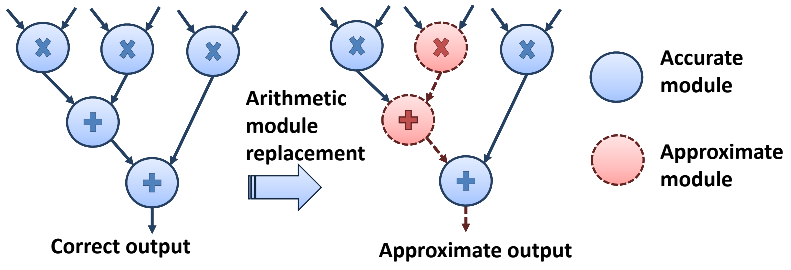


Figure 2.3: Illustration of approximation module replacement.

- *Pre-characterized EM* (EM_{char}): a sampled EM for an individual approximate hardware module that has been stored in a library. We measure EM_{char} using uniformly-distributed inputs and propose composition rules to designs with different input operand distributions (X).
- *Composition function*: a function that maps EM_{char} to $EM_{composed}$.
- $D_{ref}(X)$: the output of a correct circuit (not approximate) for an input distribution X.
- $D_{appx}(X)$: the output of an approximate circuit for an input distribution X.

Based on the definitions above, the *error metric composition* problem seeks to find a composition function for a composed EM as described in Equation (2.3), where EM^{i}_{char} denotes the EM_{char} of the i^{th} approximate module in an approximate circuit. As discussed below in Section 7.2, values of EM_{char} for different approximate hardware modules may be stored in a library for quick reference during computation of $EM_{composed}$.

$$EM_{composed} = f\left(EM^{1}_{char},\ EM^{2}_{char},\ ...,\ EM^{n}_{char}\right) \tag{2.3}$$

Error Metrics (EMs). Definitions of EMs from the literature are given in Equations (2.4) to (2.9). Note that $E[\cdot]$ indicates the *expected value* of a random variable.

$$ER = \sum_{X s.t. D_{appx}(X) != D_{ref}(X)} Pr(X) \tag{2.4}$$

$$ES = E[D_{appx}(X) - D_{ref}(X)] \tag{2.5}$$

$$ARES = E[(D_{appx}(X) - D_{ref}(X))/D_{ref}(X)] \tag{2.6}$$

$$MSE = E[|D_{appx}(X) - D_{ref}(X)|^2] \tag{2.7}$$

$$SNR = E[|D_{ref}(X)|^2/|D_{appx}(X) - D_{ref}(X)|^2] \tag{2.8}$$

$$MAXE = MAXIMUM[|D_{appx}(X) - D_{ref}(X)|] \tag{2.9}$$

- *Error rate* (ER) [146] is used to evaluate the likelihood of correctness in arithmetic operations. An accurate estimation of ER is important in the case where approximate circuits spend additional cycles for error corrections.
- *Error significance* (ES) [230] addresses the magnitude of errors. We define *ES* as a signed difference between correct and erroneous results.
- *Average relative error significance* (ARES) is used to measure the impacts of errors for image processing in [125] [247]. ARES is defined as the average absolute difference between correct and erroneous results, normalized to correct results. In digital signal processing (DSP) circuits, the magnitude of errors is important because small errors may be masked by other noise sources.
- *Mean squared error* (MSE) in [66] [237] and *signal-to-noise ratio* (SNR) in [66] [103] are common metrics to measure signal degradation in communication and image processing systems.
- *Maximum error* (MAXE) is defined as the maximum absolute value of produced errors. In [116], the MAXE metric is used to evaluate approximate circuits.

Approximate Arithmetic Modules. Various approximate arithmetic modules have been proposed in previous works, where aggressive timing and power benefits are obtained by breaking critical paths in the approximate module. To achieve a bounded error significance or configurable error rate, several techniques have been applied to reduce the severity of errors in these approximate hardware modules. ETAI [247] limits the maximum error by detecting a carry propagation and setting all lower sum bits to “1”. A similar compensation approach is used in Shin’s approximate adder [212], which detects a carry propagation using a specially designed truth table. By using error compensation approaches, the error can be reduced compared to simply breaking the carry chain.

ETAIIM [247], ACA-SD [125], Lu’s adder [166] and ACA-X [233] use a *carry-look-ahead* (CLA)-based approach to shorten the longest carry propagation path in the adder. These adders are composed of CLA submodules, and the numbers and sizes of the submodules can be configured at design time. The error significance and error rate can be configured by changing

Table 2.1: Categories of error analysis, propagation, and optimization works.

category	**(C1)**	**(C2)**	**(C3)**	**(C4)**
manipulated elements	logic cell	arithmetic	arithmetic	multiple levels
error source	approximate hardware	rounding	approximate hardware	overscaled Vdd
probabilistic errors	N	N	N	Y
reference	[230] [174] [212]	[84] [219] [58] [59] [179] [148] [155] [144]	[115] [117] [116]	[137] [64] [63] [231]

the length of carry propagation paths. Kahng et al. [125] also show that the errors can be detected and corrected in each CLA block, and that the accuracy can be configurable during runtime.

Analysis and Composition of Errors. We categorize existing works on hardware error analysis into four categories as shown in Table 2.1. In (C1), the works focus on searching for useful approximations during logic synthesis. Venkataramani et al. [230] work with existing commercial synthesis tools and simplify logic based on *approximate don't care* (ADC) information under a given error significance bound. Miao et al. [174] focus on a methodology to design more efficient adders by combining logic components to reduce the maximum error. In [212], Shin et al. provide a heuristic to search for useful approximations based on a truth table to study the tradeoff between the error rate and literal terms (hardware cost). Previous works in (C2) address rounding errors between floating-point and fixed-point conversions. In these works, the rounding errors are determined by the wordlength of hardware, and so are different from the errors induced by approximate hardware. In (C3), [116] and [117] use an interval-based approach (*integer arithmetic* (IA) or *affine arithmetic* (AA)) to propagate errors. The interval-based approach uses pre-characterized libraries for error estimations, but the runtime of characterization can increase when more intervals are required for large ranges of signals. In (C4), existing works assign overscaled supply voltages to achieve a graceful accuracy degradation. Kedem et al. [137] analyze propagations of errors induced by the degraded supply voltage, and they simplify the analysis by assuming that no error cancellations occur between multiple adders. Venkatesan et al. [231] propose the *MACACO* flow to evaluate propagations of errors induced by the overscaled supply voltage. They also apply this approach to characterize errors for different approximate adders. Chippa et al. in [64] [63] propose methodologies to analyze and optimize computing effort at different levels of abstraction, and also consider errors due to overscaled voltage supplies.

2.2.4 Power Gating

Power gating has been studied at both architectural and circuit levels. Microarchitectural works typically examine questions related to the use of different power-gating modes, selecting which circuits to power gate, and predicting when to power gate as well as control algorithms to avoid energy penalties from poor power gating decisions. Circuit-level papers typically analyze different circuit techniques aimed at reducing wake-up latency, efficiently retaining logic states, minimizing ground bounce, and achieving resilience to process variation.

Architectural-Level Power Gating. Hu et al. [113] propose power gating as a technique to reduce functional unit leakage power when applications underutilize their functional units. Specifically, they power gate the floating-point and fixed-point units according to three different predictors which are respectively ideal, time-based, and branch-misprediction-guided. The best technique (branch-misprediction-guided) is able to put functional units to sleep for up to 40% of total cycles with only 2% performance loss. The authors of [113] also develop equations to estimate the break-even points for power gating an out-of-order superscalar processor. However, although they build a power consumption model with precise analysis of virtual supply voltage during power gating, they do not consider the wake-up energy required to restore circuit nodes.

Lungu et al. [167] show that in many cases, the predictor of [113] can lead to increased energy consumption. A monitor that controls the use of power gating is introduced to bound the performance and energy penalty for misbehaved applications. Madan et al. [170] extend the idea of Lungu et al. to the core level, and propose a “guard mechanism” that reduces harmful use of power gating.

Power gating technology is also readily visible in leading commercial products. The recent Nehalem architecture employs power gating at the core level to reduce leakage power on idle cores, but $100ms$ is required to wake up a core [151] [158]. AMD [200] has improved this power gating technique by optimizing the wake-up sequence to skip built-in self tests and restoration of cache state; this results in wake-up times as short as 75μs. In today’s systems, the operation system typically power gates the cores in the idle loop. Of particular note for our work in Chapter 8: this misses out on opportunities for power gating during long memory accesses.

Circuit-Level Power Gating. In the realm of circuit innovation, the pioneering work of Horiguchi et al. [110] has been followed by many works on fundamental circuit design issues related to power gating, including switch-cell sizing, data-retention methods, physical-implementation methodologies, and mode-transition noise analysis and reduction. The recent survey of Shin et al. [213] gives an excellent summary of the history and highlights of power gating techniques.

Agarwal et al. [36] and Singh et al. [215] examine multiple sleep modes that feature different wake-up overheads and leakage power savings. Use of multiple sleep modes achieves an extra 17% reduction in leakage power compared to a single power gating mode. Also, one of the sleep modes can reduce leakage power by 19% while preserving circuit state. However, these energy savings are based on static traces of bus activity and do not address the runtime problem of predicting *when* to power gate. In addition, the reported results are likely optimistic since wake-up noise and the overhead of implementing low-voltage sleep control signal distribution are not considered.

To minimize ground bounce during mode transition, Kim et al. [141] control turn-on voltage (V_{GS}), which makes sleep transistors turn on in a non-uniform stepwise manner. Kim et al. [142] propose a tri-mode power gating structure in which a PMOS switch is combined in parallel with traditional NMOS power gating switches. The additional PMOS transistor supports intermediate power-saving state-retaining modes at low-supply voltage, and reduces ground bounce noise during transitions between normal and power-gated modes. Chowdhury et al. [70] propose a similar tri-mode (i.e., RUN, HOLD, CUT-OFF) power gating technique using PMOS switches in parallel with NMOS footer switches, combined with additional NMOS switches in parallel with PMOS header switches. Kim et al. [140] propose a programmable-width power gating switch that adjusts the widths of power gating switches to compensate for core-to-core process variation occurring in multi-core systems. Finally, Zhang et al. [244] propose a multi-mode power gating technique using three NMOS switches with different sizes and threshold voltages. Using various combinations of the three switches, they can provide multiple power-gating modes with different leakage savings. They also note that their method is tolerant to process variation.

Memory Access Aware Power Gating. Memory Access Aware Power Gating for MPSoCs [122] examines the potential to power gate an in-order core while monitoring a single memory bus and estimating memory latencies. A controller that sits at the memory bus sends explicit commands to each core to power gate and to wake up from a power-gated state. The controller estimates memory latencies by tracking whether each memory request is a row buffer hit or miss. However, this work does not consider out-of-order execution, and is limited in scalability to a system with a single memory bus, which precludes understanding of its application to data-center servers. In addition, it does not consider the importance of core location and state information for determining safe wake-up modes, the possibility of using staggered wake-up to reduce the latency of core wake-up from a power-gated state, or the scalability of their designs. By contrast, the token-based adaptive power gating that we report in Chapter 8 addresses these issues, is applicable to out-of-order cores, formally analyzes the energy savings for in-order cores, considers the importance of core location and wake-up stagger, and considers the scalability of the design to many-core processors.

Chapter 3

Gate Sizing for Leakage Reduction

The sizing problem in VLSI design seeks to determine design parameters (e.g., gate width and threshold voltage) for each gate, so as to optimize timing, area and power of a circuit, subject to constraints. The problem has been extensively studied, and a number of heuristics have been proposed. However, there have been no definitive comparisons (empirical or mathematical) of different techniques. Moreover, many published techniques make unrealistic assumptions about the underlying circuits, such as the possibility of *continuous gate sizing and V_t assignment* and the convexity of delay functions. Some publications neglect the effect of rounding when using a discrete gate library, or do not account for realistic capacitance and slew constraints. Scalability to circuits with hundreds of thousands of gates is also an important issue, whereas many previous publications use much smaller benchmarks mapped into outdated technologies. To address these shortcomings in published literature, Intel researchers have recently prepared and released an extensive infrastructure for research on large-scale gate sizing [184]. This includes (i) a set of benchmarks ranging from small to large, mapped into a modern discrete gate library, and (ii) a set of evaluation protocols that includes checking timing constraints using industry-standard software (*Synopsys PrimeTime* [29]) and measuring total leakage power for a particular sizing solution. This infrastructure has been used in the ISPD-2012/2013 Gate Sizing Contests, and provides a baseline for further research on this topic.

Within the research-oriented infrastructure used in the ISPD-2012/2013 Gate Sizing Contests, we develop a metaheuristic approach to gate sizing that integrates timing and power optimization, and handles several types of constraints. In this chapter, we describe a successful entry from the ISPD contest that achieves practical large-scale metaheuristic gate sizing and V_t optimization with a signoff timer (ST) in the loop. We also propose a new benchmark generation approach for (leakage) power-driven gate sizing. The main contributions of our work are as follows.

- We use a sensitivity-guided metaheuristic approach based on *sequential importance sampling* [39] that integrates power and timing optimization, and handles several types of constraints.
- We define a parameterized space of sensitivity functions for gate sizing and traverse this space using a multistart technique that naturally lends itself to efficient parallelization on multi-core and shared memory CPU architectures, and distributed systems.
- We develop new mechanisms that enable close tracking of an external signoff timer (ST) without undue loss of efficiency.
- We make a choice of internal delay and slew models that permit sufficient calibration to the external ST.
- We propose benchmark circuits with known optimal solutions for gate sizing.
- The proposed benchmarks resemble real designs in terms of size, path depth (number of logic stages), and net degree distribution. These parameters are extracted from real designs. The property of known optimal solution quality is maintained.

3.1 Sensitivity-Guided Metaheuristics for Gate Sizing

Our research reported in this section focuses on large-scale optimization of gate sizes under realistic circumstances. Our techniques accurately track circuit timing throughout the optimization process, ensure the satisfaction of several types of constraints, and identify the gates with the greatest impact on power-performance tradeoffs. Rather than approximate gate sizing by continuous convex optimization, as is done in many prior publications, we fully account for the discrete nature of the problem and the nonconvexities of circuit delay functions. Our optimizations try not to overlook available opportunities to improve power-performance tradeoffs, but are also fast and can quickly traverse large regions of the solution space. They are highly modular, and are organized in a hierarchy where high-level metaheuristics configure and drive heuristics, which are assembled from lower-level optimization blocks. The lower-level optimizations are in turn based on high-performance timing and power analysis, constraint repair, search, gain calculations, sorting and prioritization, as well as roll-back. Some of these ideas have been known before, but every hierarchical level in our techniques contains some novel elements. Moreover, in a highly studied and competitive field such as gate sizing, reliable individual components form only a small fraction of overall success — a larger fraction of success is in the selection, ordering and composition of these components, as well as the overall flow. Our most important decisions rely on new insights into large-scale gate sizing and its interaction with design constraints.

3.1.1 Metaheuristics for Gate Sizing

Our proposed heuristic has two stages – Global Timing Recovery (GTR) and Power Reduction with Feasible Timing (PRFT). GTR first seeks violation-free (feasible) solutions, and then PRFT iteratively reduces total leakage power of sizing solutions by local search, as illustrated in Figure 5.4. At each stage of our optimization flow, we parameterize the space of sensitivity functions, and traverse this space to find the best configurations of sensitivity by independent multistarts (Figure 3.2). After each multistart, we compare all obtained solutions and retain the best/non-dominated solutions. This is accomplished by adapting the go-with-the-winners (GWTW) metaheuristic (Section 2.1.1). However, our optimization is purely deterministic in that our multistart procedure begins with the small set of the best-seen solutions, whereas GWTW is typically randomized. Solutions after each stage are ensured to be feasible, which enables pruning of dominated solutions by GWTW.

3.1.2 Global Timing Recovery

This stage starts with an arbitrary cell configuration that is incrementally refined by increasing/decreasing gate sizes or downscaling/upscaling threshold voltages. The interaction of these steps in our implementation has been optimized for the case where the initial solution generally underestimates optimal power dissipation of individual gates and likely violates timing constraints. Therefore, we start with timing recovery by upsizing gates or downscaling threshold voltages. We observe that best-seen solutions for several ISPD-2012 benchmarks configure most cells at or close to their minimum-leakage configurations (Table 3.3), making minimum-leak-age configurations for all cells an appropriate initial setting. For benchmarks with tighter timing constraints, alternative initial settings may save runtime. However, our optimization techniques are sufficiently fast and robust to start with minimum-leakage configurations. Empirically, finding a timing-feasible solution in the “coarse-search” stage of our optimization accounts only for a single-percent fraction of the total required runtime (Section 3.1.5).

Starting with an underpowered configuration, we seek to generate feasible solutions by

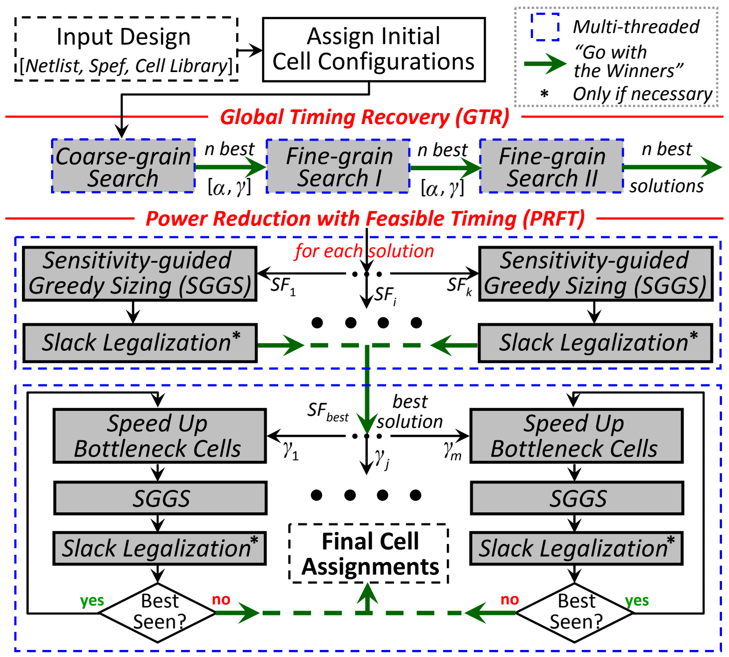


Figure 3.1: Algorithmic flow of our heuristic for timing recovery (GTR) and leakage reduction (PRFT).

monotonically (i) increasing cell sizes or (ii) lowering cell threshold voltages (V_t). Both cell upsizing and V_t downscaling are performed in smallest possible steps; the ordering of these actions is determined by their *sensitivities*, which are calculated by impacts on TNS and leakage power:

$$sensitivity_{GTR} = \frac{\Delta TNS}{\Delta leakage\_power^{power\_exponent}} \tag{3.1}$$

When estimating the impact of a single cell modification, invoking STA can be computationally prohibitive. Therefore, we approximate the impact on TNS of a single cell modification (m_i^k) on cell c_i using $NPaths_i$, the number of negative-slack paths that pass through c_i. We define the *nearest-neighbor set* of c_i to be the set of cells that have a driver (fanin) in common with c_i, and N_i to be the union of c_i's nearest-neighbor set and c_i itself. Then, we estimate $\Delta TNS(m_i^k)$ as

$$\Delta TNS(m_i^k) \approx \Sigma_{c_j \in N_i} - \Delta delay_j^k \times \sqrt{NPaths_j} \tag{3.2}$$

where $\Delta delay_j^k$ is an estimated delay change on c_j due to m_i^k. This approximation is based on the fact that any perturbation to cell c_i will change the delay of c_i, but also can impact the slacks of other cells that share path(s) with c_i (i.e., changing the V_t of c_i can change *required arrival*

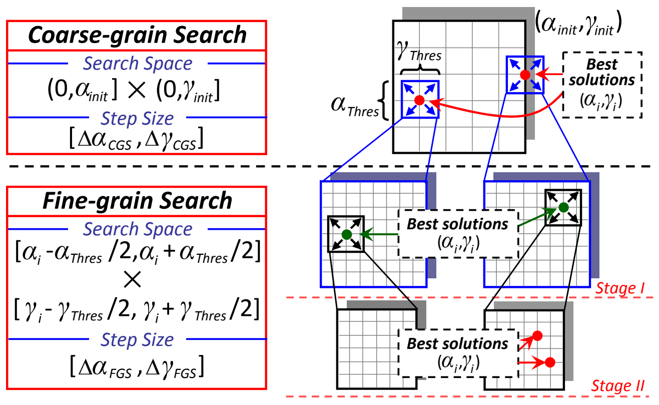

Coarse-grain Search
Search Space
(0,α_{init}] × (0,γ_{init}]
Step Size
[$\Delta\alpha_{CGS}$,$\Delta\gamma_{CGS}$]
Fine-grain Search
Search Space
[α_i -α_{Thres} /2,α_i +α_{Thres}/2]
×
[γ_i- γ_{Thres}/2, γ_i+ γ_{Thres}/2]
Step Size
[$\Delta\alpha_{FGS}$,$\Delta\gamma_{FGS}$]
(α_{init},γ_{init})
γ_{Thres}
α_{Thres}
Best solutions (α_i,γ_i)
Best solutions (α_i,γ_i)
Stage I
Best solutions (α_i,γ_i)
Stage II

Figure 3.2: Search-range changes during the GTR search procedure.

times (RATs) for its upstream cells, and change *actual arrival times (AATs)* for its downstream cells). To account for this, we introduce the factor $\sqrt{NPaths_j}$, which reflects the number of fanin and fanout cells of c_j that are affected by the delay change of c_j. If this effect is not accounted for, particularly for cells that are on critical paths, the impact on TNS will be underestimated. To more accurately estimate the impact on TNS when we resize[1] c_i, we must consider all cells in N_i, as changing the size will affect the capacitive load on the common driver, which affects their arrival (and transition) times. Following the empirical observation in [186], we assume that the propagation of increased transition time can be safely bounded to only the nearest neighbors.

We sort the changes by non-increasing sensitivity values and commit them in order (Algorithm 1). Given that each single-cell modification is evaluated assuming other cells are fixed, the inaccuracies of sensitivity accumulate as multiple cells are changed. Therefore, we only commit the first $\gamma\%$ of the modifications between two consecutive STA invocations. The variables $power\_exponent$ ($0 \leq \alpha \leq 3.0$) and $commit\_ratio$ ($0 < \gamma \leq 60\%$) determine specific multistart configurations. To effectively reduce the size of the search space, we perform multilevel search (Figure 3.2). Fine-grain search is performed on non-dominated configurations from coarser search, but with finer steps and over smaller ranges.

3.1.3 Power Reduction with Feasible Timing

From the Global Timing Recovery (GTR) stage, we obtain a feasible sizing solution with no timing, slew or maximum capacitance violations. However, the solution can improve further

[1]V_t changes do not affect the delays of neighboring cells.

Algorithm 1 Global Timing Recovery (GTR).

```
   Procedure TimingRecovery($\alpha$, $\gamma$, $N$)
   Input : power exponent $0 \leq \alpha \leq 3.0$, commit ratio $0 < \gamma \leq 60\%$, netlist $N$
   Output : sizing solution $S$
1: Set the current solution $S$ with a minimum-leakage setting;
2: Fix maximum capacitance violations; // Section 3.1.4
3: Run STA to initialize slack and delay values for the netlist N;
4: while ($!S.feasible()$ and $S.leakage < best\_seen\_leakage$) do
5:    Update $NPaths_i$ for each cell instance $c_i$ in the netlist $N$;
6:    $M \leftarrow \emptyset$; $counter \leftarrow 0$;
7:    for each cell instance, $c_i$ in the netlist $N$ do
8:       if cell $c_i$ is upsizable then
9:          $m^k.target \leftarrow c_i$; $m^k.change \leftarrow upsize$;
10:         $m^k.sensitivity \leftarrow \Delta TNS$ / $\Delta leakage\_power^{\alpha}(c_i, upsize)$;
11:         $M \leftarrow M \cup \{m^k\}$;
12:      end if
13:      if cell $c_i$ is not a $LVT$ cell then
14:         $m^k.target \leftarrow c_i$; $m^k.change \leftarrow V_t\text{-}downscaling$;
15:         $m^k.sensitivity \leftarrow \Delta TNS$ / $\Delta leakage\_power^{\alpha}(c_i, downscale)$;
16:         $M \leftarrow M \cup \{m^k\}$;
17:      end if
18:   end for
19:   while ($counter < \gamma \times M.size()$) do
20:      Pick a modification $m^k$ with maximum $sensitivity$ in $M$;
21:      Commit $m^k.change$;
22:      $M \leftarrow M \setminus \{m^k\}$;
23:      $counter \leftarrow counter + 1$;
24:      Fix maximum capacitance violations;
25:   end while
26:   Run STA to evaluate the current sizing solution $S$;
27: end while
28: if $S.feasible()$ then
29:   Update $best\_seen\_leakage$;
30: end if
```

since some cells are oversized during the timing recovery stage. We reduce leakage power while maintaining timing feasibility by alternating (i) *sensitivity-guided greedy sizing (SGGS)*, (ii) *slack legalization*, and (iii) speeding up *bottleneck* cells.

Sensitivity-Guided Greedy Sizing (SGGS). SGGS downsizes cells according to sensitivity while avoiding timing violations. Algorithm 2 presents pseudocode of our SGGS procedure. In this algorithm, $ISTA(c_i)$ is an incremental STA operation (i.e., update of timing analysis) after cell c_i is changed. In $ISTA(c_i)$, we start timing and slack updates at the fanin nodes of the changed cell. From the fanin nodes, transition times and AATs are propagated in the forward direction, and RATs are propagated in the backward direction.

The SGGS algorithm starts with STA and initializes all timing nodes. Sensitivities are

Algorithm 2 Sensitivity-Guided Greedy Sizing (SGGS).

```
Procedure SGGS(SF, S, N)
Input : sensitivity function $SF$, a feasible sizing solution $S$, netlist $N$
Output : sizing solution $S$ with reduced power
1: Run STA to initialize delay values for the given solution $S$;
2: $M \leftarrow \emptyset$;
3: for each cell instance, $c_i$ in the netlist $N$ do
4:     if cell $c_i$ is downsizable then
5:         $m^k.target \leftarrow c_i$; $m^k.change \leftarrow downsize$;
6:         $m^k.sensitivity \leftarrow ComputeSensitivity(c_i, downsize)$;
7:         $M \leftarrow M \cup \{m^k\}$;
8:     end if
9:     if cell $c_i$ is not a $HVT$ cell then
10:        $m^k.target \leftarrow c_i$; $m^k.change \leftarrow V_t\text{-}upscaling$;
11:        $m^k.sensitivity \leftarrow ComputeSensitivity(c_i, upscale)$;
12:        $M \leftarrow M \cup \{m^k\}$;
13:    end if
14: end for
15: while $M \neq \emptyset$ do
16:    Pick a modification $m^k$ with maximum $sensitivity$ in $M$;
17:    $S' \leftarrow SaveState(S)$;
18:    Commit $m^k.change$;
19:    $M \leftarrow M \setminus \{m^k\}$;
20:    $ISTA(m^k.target)$;
21:    if $!S.feasible()$ then
22:        $S \leftarrow RestoreState(S')$;
23:    else
24:        if cell $m^k.target$ is downsizable and not a $HVT$ cell then
25:            Recalculate $m^k.sensitivity$;
26:            $M \leftarrow M \cup \{m^k\}$;
27:        end if
28:    end if
29: end while
```

computed for all downsizable cells in Lines 3–14. We consider both gate downsizing and V_t upscaling for the sensitivity calculation. We define five sensitivities, as summarized in Table 3.1.

Table 3.1: Sensitivity functions for $SGGS$. $SF4$ and $SF5$ appear most successful (Table 3.4), and our metaheuristic produces better results when using multiple functions.

acronyms	sensitivity functions
$SF1$	$-\Delta leakage\_power/\Delta delay$
$SF2$	$-\Delta leakage\_power \times slack$
$SF3$	$-\Delta leakage\_power/(\Delta delay \times \#paths)$
$SF4$	$-\Delta leakage\_power \times slack/\#paths$
$SF5$	$-\Delta leakage\_power \times slack/(\Delta delay \times \#paths)$

$\Delta leakage\_power$ and $\Delta delay$ represent leakage power and cell delay changes after the downsizing of cell c_i. The variable $slack$ represents the slack at the output pin, and $\#paths$ is the number of timing paths that pass through the cell c_i. The $slack$ value is positive since the downsizing is applied to cells with positive slack. $\#paths$ is calculated similarly to $NPaths$ in Section 3.1.2, but including positive-slack paths. $SF1$ and $SF2$ have been used in [86] [100] and [101], respectively. We have added $\#paths$ into $SF3$ and $SF4$ to favor cells with smaller impact on the slacks of other cells. $SF5$ is a hybrid of $SF1$ and $SF2$; similar logic is used in [218], but without considering $\#paths$. In Lines 15–22, we select a cell c_i with maximum sensitivity, and downsize c_i or upscale its V_t. We perform incremental timing analysis and check for violations. If the sizing step creates a timing, slew or maximum capacitance violation, it is undone. The loop continues until M becomes empty.

Slack Legalization. $ISTA$ achieves a significant speedup over full-netlist STA through propagation of timing changes that are related only to updated instances. We achieve further speedup by blocking the propagation when changes to timing are below a $propagation\_threshold$.[2] Due to this limited accuracy, SGGS can overlook a small number of timing violations. Instead of using GTR, we use a *slack legalization* procedure to rectify small timing violations at a small leakage power cost.

In slack legalization, we first collect cells which are in critical (negative-slack) paths. These cells are sorted in decreasing order of $|\Delta delay|$ (delay improvement due to upsizing and V_t downscaling) and are modified in this order. Unlike GTR, slack legalization tracks slack changes after each cell modification, and ensures no timing degradation. Let $\Delta slack(c)$ be

[2]By default, $propagation\_threshold$ is set to $0.1ps$. Table 3.2 shows that ISTA runs faster if a higher $propagation\_threshold$ is given.

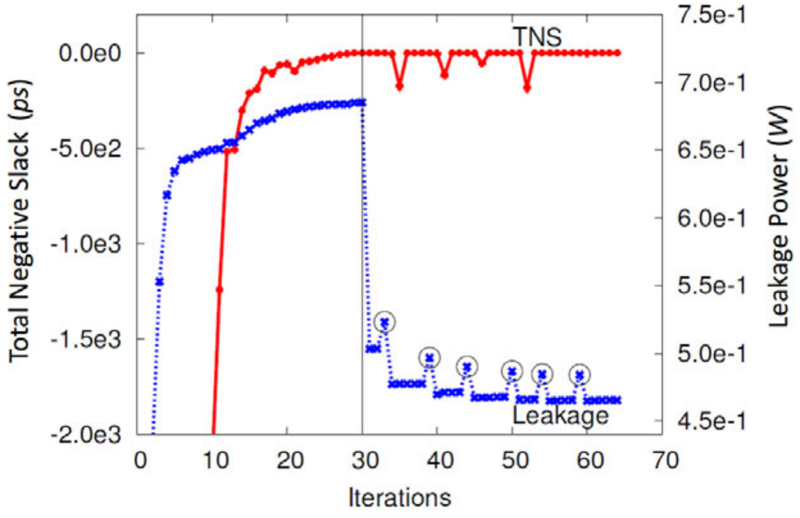


Figure 3.3: Progression of PRFT on *vga_lcd_fast*.

the slack change on output pin of cell c, and $C_{fi}(c)$ be set of fanin cells of cell c. After the modification of cell c_i, slack legalization restores the change if (i) $\Delta slack(c_i) \leq 0$, or (ii) $\Delta slack(c_i) + \Sigma_{c_j \in C_{fi}(c_i)} \Delta slack(c_j) \leq 0$. Slack legalization repeats the sizing until all timing violations are fixed.

Speeding up Bottleneck Cells. During greedy sizing, we size gates monotonically downward with lower-size or higher-V_t library cells, but the resulting solution is of course a local optimum. A key obstacle is that we have no timing slack available to allow further gate downsizing. Therefore, to recover timing slack with the least impact to power, we speed up *bottleneck* cells, i.e., cells that participate in many timing-critical paths. We identify these cells by a bottleneck analysis similar to that provided in EDA tools [29]. We then perturb the converged solution by assigning larger sizes or lower V_t cells, and then repeat the downsizing procedure.[3] To identify bottleneck cells, we estimate total slack changes by $\Delta delay \times \sqrt{\#paths}$ due to hypothetical cell upsizing (or V_t downscaling). We commit the first $\gamma\%$ of such modifications with largest $\Delta total\_slack$, then optimize leakage power with $SGGS$. Timing violations created by speeding up bottleneck cells or $SGGS$ are removed by slack legalization. To define specific multistart configurations, we sweep γ from 1% to 5% with a step size of 1%. The iterations (speeding up bottleneck cells + $SGGS$ + slack legalization) terminate when the solutions stop improving. Figure 3.3 illustrates the progression of PRFT. Starting with a feasible solution from GTR, PRFT iteratively reduces leakage power while maintaining timing feasibility.

[3]This recalls the large-step Markov chain approach [172].

3.1.4 Handling Capacitance and Slew Violations

Each standard cell can drive a certain maximum capacitance load defined in the library (e.g., based on the contest library, the maximum capacitance that can be driven by the smallest four-input *NAND* gate with high V_t is $3.2fF$). If a cell is overloaded, its output transition time slows down significantly, resulting in overall degradation of slacks in its fanout cone. In our heuristic, we remove maximum capacitance and slew violations at every iteration of GTR (Algorithm 1) by alternating *backward-* and *forward-traversal* repair as necessary. During *backward traversal*, we visit cells in a reverse topological order and continue to upsize driving cells until capacitance violations for the driving cells are removed or the driving cells reach their maximum sizes (whichever comes first). This procedure resolves most of the capacitance violations in the early iterations of GTR when cell sizes are relatively small. However, in later stages, cells on certain paths can be saturated at their maximum sizes,[4] and we must downsize some of their fanout cells. Therefore, during *forward traversal*, we visit cells in a forward topological order and continue to downsize fanout cells until capacitance violations for the current cells are removed or all fanout cells shrink to their minimum sizes (whichever comes first). Empirically, this requires one to two iterations of backward and forward traversals. As this happens, all output transitions become faster than the maximum slew allowed at the ISPD-2012 Gate Sizing Contest ($300ps$).

3.1.5 Analysis of Our Implementation

Our implementation, Trident, is written in C++, compiled with g++ 4.6.2 and validated on a 3.2*GHz* Intel Xeon E31230 Linux workstation with 8*GB* of memory, using four CPU cores. We compare it to the results of the ISPD-2012 Gate Sizing Contest on the ISPD-2012 benchmark suite [184]. Timing violations are verified by *Synopsys PrimeTime* [29], and leakage-power values are read from the official contest evaluation script [184]. In all experiments, we use the default settings described in Section 3.1.1.

Trident is a stand-alone tool that includes a built-in static timer and relies only on standard C++ libraries. Instead of analytical model-fitting, the built-in timer is based on library table lookups, linear interpolation, and timing propagation. With capacitive modeling of wires used in the contest, the timer correlates to *PrimeTime* within $10^{-3}ps$ precision but runs $60\times$ faster on average. Table 3.2 compares the runtimes of $ISTA$ and full-scale STA of our timer.

[4]For instance, if one inverter is driving numerous large cells, even a maximum-sized inverter cannot remove capacitance violations.

Table 3.2: Runtime comparisons of full-scale STA (sec) and incremental STA (ms) after changing the size of one cell.

benchmarks	FSTA	ISTA (*ms*)			
	(*sec*)	0ps	0.1ps	0.5ps	1.0ps
dma	0.233	1.495	0.845	0.620	0.508
pci_b32	0.271	0.982	0.717	0.388	0.348
des_perf	1.108	0.700	0.508	0.471	0.422
vga_lcd	1.729	22.75	8.108	7.822	2.069
b19	2.435	5.460	2.717	2.115	1.833
leon3mp	6.746	43.21	2.152	1.542	0.939
netcard	9.751	9.612	2.299	1.940	1.675
geomean	924.58×	2.86×	1.00×	0.77×	0.54×

Running in 3-4 threads, Trident generates feasible solutions for all 14 benchmarks in 83 hours using less than 6.0*GB* of memory. Our implementation dynamically assigns multistart configurations to available threads, and therefore it can readily issue more parallel threads as memory allows. An example runtime breakdown on the *netcard_slow* benchmark is as follows. **GTR** takes 31.3% (6.2% coarse search, 25.1% fine-grain search) of total runtime, of which 53% is spent in full-scale STA, 20% in TNS estimation, and 12% is spent in fixing maximum capacitance violations. **PRFT** takes 68.5% (45.4% SGGS, 23.0% perturbing iterations), of which 87.6% is in $ISTA$. I/O takes 0.2% of runtime.

3.1.6 Comparisons to the State of the Art

Table 3.3 compares Trident to top contestants at the ISPD-2012 Gate Sizing Contest. Performance results[5] for individual teams are quoted from [184]. Trident has found feasible sizing solutions for all circuits in the ISPD-2012 benchmark suite. Compared to the top three teams, Trident achieves the lowest leakage power for 13 out of 14 circuits (no parameter tuning to specific benchmarks has been employed). On average, Trident obtains leakage power improvement of 43%, 16%, 52% versus NTUgs (National Taiwan University), UFRGS-Brazil (Universidade Federal do Rio Grande do Sul), and PowerValve (National Tsing Hua University and Missouri University of S&T), respectively. Geometric means are calculated excluding infeasible benchmarks, which underrepresents the impact of our proposed techniques. All of our runs are finished within the corresponding hard runtime limits [184]. For further analysis, Table 3.4 provides the

[5]The contest considered only leakage and not dynamic power.

Table 3.3: Leakage power (W) and wall clock time (minutes) on ISPD-2012 benchmarks.

benchmarks	# of cells	ISPD-2012 contest results (leakage)				Trident			LPRS [87]	
		NTUgs	UFRGS	Power Value	best	leakage after GTR	leakage after PRFT	wallclock time	leakage power	wallclock time
dma_fast	(25.3K)	0.511	0.323	0.312	0.312	0.650	**0.299**	13.9	0.238	0.92
dma_slow	(25.3K)	0.205	0.158	0.147	0.147	0.211	**0.145**	9.9	0.132	0.79
pci_b32_fast	(33.2K)	0.512	**0.168**	0.226	**0.168**	0.348	0.183	13.0	0.136	0.92
pci_b32_slow	(33.2K)	0.203	0.115	0.116	0.115	0.185	**0.111**	10.2	0.096	0.87
des_perf_fast	(111K)	2.390	3.520	2.320	2.320	7.157	**1.842**	82.7	1.395	16.37
des_perf_slow	(111K)	0.674	0.884	0.697	0.674	0.922	**0.614**	70.1	0.570	25.31
vga_lcd_fast	(165K)	0.758	0.580	0.773	0.580	0.685	**0.471**	45.6	0.413	8.37
vga_lcd_slow	(165K)	0.415	0.378	0.391	0.378	0.454	**0.351**	87.5	0.328	5.67
b19_fast	(219K)	2.710	–	4.490	1.040	1.377	**0.771**	206.5	0.717	11.75
b19_slow	(219K)	0.627	0.614	0.736	0.614	0.718	**0.583**	213.9	0.564	9.15
leon3mp_fast	(649K)	–	–	4.940	2.020	1.989	**1.487**	1323.2 *	1.443	46.62
leon3mp_slow	(649K)	1.420	1.790	2.960	1.420	1.422	**1.341**	1274.0	1.334	38.98
netcard_fast	(959K)	2.010	2.300	2.970	2.010	1.997	**1.861**	1096.9	1.841	47.41
netcard_slow	(959K)	1.770	1.970	1.940	1.770	1.818	**1.770**	299.9	1.763	34.39
geometric mean		**1.43**×	**1.16**×	**1.52**×	**1.11**×	**1.53**×	**1.00**×		**0.90**×	

best parameter values found by our metaheuristics for individual benchmarks. Recently, Flach et al. [87] have made significant improvements on the ISPD-2012 benchmark optimization using Lagrangian Relaxation. Their approach shows 10% leakage power improvement over our results.

Comparing our approach to [186], we note the following.

- The winners of the contest (NTUgs) have implemented algorithms in [186] with additional improvements, and we include their contest results in Table 3.3.
- Intel released their results for five benchmarks (out of 14 total), where they observed significant room for improvement compared to the best contest results. Of these five benchmarks, we outperform [184] on four, and are slightly behind on one.

Hence, our optimizer appears competitive.

3.1.7 Comparison to Minimum-Leakage Solutions

In addition to reporting achievable results, we estimate available room for further improvement. Starting with minimum-leakage configurations for each cell instance, we heuristi-

Table 3.4: Parameters for GTR and PRFT associated with the best solutions found. This parameter sweep is included in reported runtimes. $SF1$–$SF5$ are described in Table 3.1.

benchmarks	GTR		PRFT	
	α	γ (%)	$SF\#$	γ (%)
dma_fast	0.91	24.5	SF5	1.0
dma_slow	1.00	10.0	SF5	5.0
pci_b32_fast	0.91	34.0	SF4	4.0
pci_b32_slow	1.11	36.0	SF5	4.0
des_perf_fast	0.85	46.5	SF5	1.0
des_perf_slow	0.83	8.5	SF2	3.0
vga_lcd_fast	0.70	17.5	SF5	4.0
vga_lcd_slow	1.00	10.0	SF4	3.0
b19_fast	1.33	16.5	SF2	4.0
b19_slow	1.50	7.5	SF5	1.0
leon3mp_fast	0.71	7.0	SF4	1.0
leon3mp_slow	0.89	4.0	SF4	2.0
netcard_fast	0.57	4.0	SF3	1.0
netcard_slow	2.67	4.0	SF3	1.0

cally fix slew and maximum capacitance violations while increasing leakage power by between 5.8% (*netcard*) and 53.5% (*dma*). In other words, we estimate the leakage cost of achieving electrical feasibility with respect to only load and slew constraints (ignoring timing). The ratio of total leakage power of our timing-feasible configurations to that in solutions constructed as just described gives an indication of the additional leakage penalty needed to fix timing violations. Table 3.5 shows that for the largest benchmarks the penalty is very small, and our solutions likely cannot be improved by more than several percent. Benchmarks with tighter timing constraints ("fast") require greater leakage penalty to achieve timing feasibility, and this is especially true for smaller benchmarks. This suggests that the availability of a strong gate-sizer could allow the tightening of timing constraints for large (power-constrained) designs. Based on leakage power values in Table 3.3, the last column in Table 3.5 approximates the amount of total negative slack that can be removed by each doubling of leakage power from the electrically feasible solutions that we have constructed.

Table 3.5: Leakage power ratios of electrically feasible solutions to our best solutions. Since our solutions are (timing) feasible, ΔTNS equals TNS of electrically feasible solutions.

benchmarks	minimum leakage	ratio	$\frac{\Delta TNS}{log_2 Ratio}(\mu s)$
dma_fast	0.073	4.08	1.92
pci_b32_fast	0.063	2.89	1.95
des_perf_fast	0.268	6.88	1.31
vga_lcd_fast	0.303	1.54	31.2
b19_fast	0.522	1.48	11.0
leon3mp_fast	1.311	1.15	148
netcard_fast	1.766	1.06	216
dma_slow	0.073	1.98	3.40
pci_b32_slow	0.063	1.75	3.25
des_perf_slow	0.268	2.29	2.58
vga_lcd_slow	0.303	1.16	77.4
b19_slow	0.522	1.12	24.2
leon3mp_slow	1.311	1.02	643
netcard_slow	1.766	1.002	2290

3.2 High-Performance Gate Sizing with a Signoff Timer

In this section, we describe a successful entry from the ISPD-2013 contest that achieves practical large-scale metaheuristic gate sizing and V_t optimization with a **signoff timer (ST)** in the loop.

The software system we describe integrates a number of previously known components and ideas with new ones. Significant effort was spent on identifying *the most pertinent ideas, techniques and components* (the optimization framework, models for interconnect capacitance, delay and slew, etc.) whose runtime-quality tradeoffs are consistent with the desired performance envelope of a high-performance sizer. Another area of major importance is *partitioning the overall optimization into separate stages* that pursue specific goals and call for dedicated components. The *handoff between such stages* has also been critical to the overall performance. Given the large amount of computation involved, *effective use of parallel-computing resources* is required.

3.2.1 Interconnect Modeling

We now review fast delay models and explain how we have selected models appropriate for high-performance optimization. A core insight, well understood in the field, is that early optimization does not require signoff timing accuracy and can be performed with simpler, faster delay models. Therefore, we perform empirical studies of known models to assess tradeoffs between (i) accuracy versus signoff timing, (ii) computation complexity and runtime, and (iii) impact on sizing results.

Figure 3.4 illustrates basic modeling of interconnects. In Figure 3.4(a), delay from pin X to pin Y is composed of gate ($cell1$, $cell2$) delay and wire (A-B) delay. For the gate/cell delay calculation, lookup table-based nonlinear delay models (NLDMs) are widely used and represent functions of input slew and output capacitance in library (*Synopsys Liberty*) files. With the NLDM, cell delay and slew estimation from a signoff timer (ST) can be reproduced with negligible errors. However, incorrect wire slew estimation (in pin B) can lead to large errors in the slew and delay estimation for $cell2$.

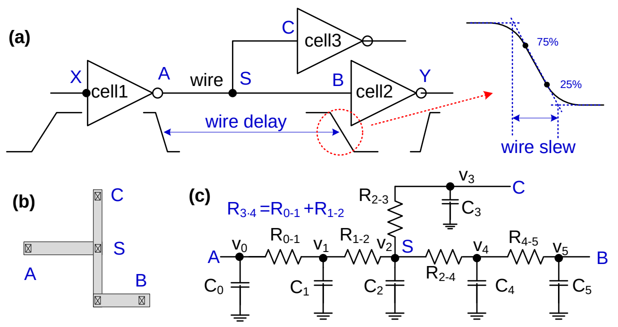


Figure 3.4: Interconnect modeling; (a) wire between *cell1* and *cell2*, (b) wiring tree with a Steiner point *S*, and (c) RC segment tree with RC nodes (N = 5).

Delay Modeling. We consider Elmore delay (EM) [82], D2M [40], and two 2-pole (DM1, DM2) [133] interconnect models. More complex models, such as PRIMA [182] and RICE [198], are more difficult to implement and too slow for high-performance gate sizing. Furthermore, empirical results in Section 3.2.8 show that our modeling is conducive to highly competitive results.

In an RC tree with nodes v_0, ..., v_N (v_0 is the source) as shown in Figure 3.4(c), let C_i be the capacitance at node v_i for $0 < i \leq N$, and let R_{ki} be the total resistance of the intersection (overlap) between the unique path from v_0 to v_i and the unique path from v_0 to v_k. The Elmore delay from node v_0 to node v_i is given by

$$EM_i = \sum_{k=1}^{N} R_{ki} \times C_k. \tag{3.3}$$

Elmore delay is the first moment of impulse response and can be inaccurate when there is a high degree of resistive shielding. Alpert et al. [40] propose the D2M (delay with 2 moments) metric which is a simple function of the first two circuit moments, m_1 and m_2 respectively. Starting with $m_0 = 1$, the j^{th} moment of the impulse response [40] for node v_i is defined as

$$m_j^{(i)} = -\sum_{k=1}^{N} R_{ki} \times C_k \times m_{j-1}^{(k)} \tag{3.4}$$

Higher moments for each subnode in a net can be calculated by traversing the RC tree recursively. We can express the delay models in terms of the first and second moments as follows.

$$EM = -m_1 \qquad D2M = \ln 2 \frac{m_1^2}{\sqrt{m_2}} \tag{3.5}$$

$$DM1 = \frac{1}{2}(-m_1 + \sqrt{4m_2 - 3{m_1}^2}) \ln(1 - \frac{m_1}{\sqrt{4m_2 - 3{m_1}^2}}) \tag{3.6}$$

$$DM2 = \ln 2\sqrt{2m_2 - {m_1}^2} \tag{3.7}$$

Wire Slew Model. We consider the PERI [135] and scaled S2M [37] models. The PERI model is given as

$$PERI(v_j) = \sqrt{T_{v_i}^2 + (\ln 9 \times m_1)^2} \tag{3.8}$$

where T_{v_i} and $PERI(v_j)$ are the slews at nodes v_i and v_j, respectively, and m_1 is the first moment of node v_j. The S2M model is given as

$$S2M(v_j) = \sqrt{T_{v_i}^2 + \ln 9 \frac{\sqrt{-m_1}}{\sqrt[4]{m_2}} \sqrt{2m_2 - m_1^2}} \tag{3.9}$$

where m_1 and m_2 are the first and second moments of node v_j and T_{v_i} is the slew of node v_i.

Capacitance Model. Empirical formulas for delay and transition time of gates depend only on the input slew rate and a single load capacitance, called *effective capacitance*, which represents the cumulative effect of the load. We have implemented McCormick's effective capacitance model [173] based on a normalized 2D lookup table. The method is iterative and converges to an effective capacitance value, but is slower than closed-form delay models. For ISPD-2013 testcases, *total capacitances* are very close to *effective capacitances* for more than 85% of nets, providing sufficient accuracy for our delay and slew calculation in early optimization stages. Therefore, our calculations of gate delay and transition time use *total capacitance* instead of McCormick's *effective capacitance* model; this improves runtime without any noticeable loss of accuracy. Repeated calibration with signoff timing increases accuracy of modeling at later stages.

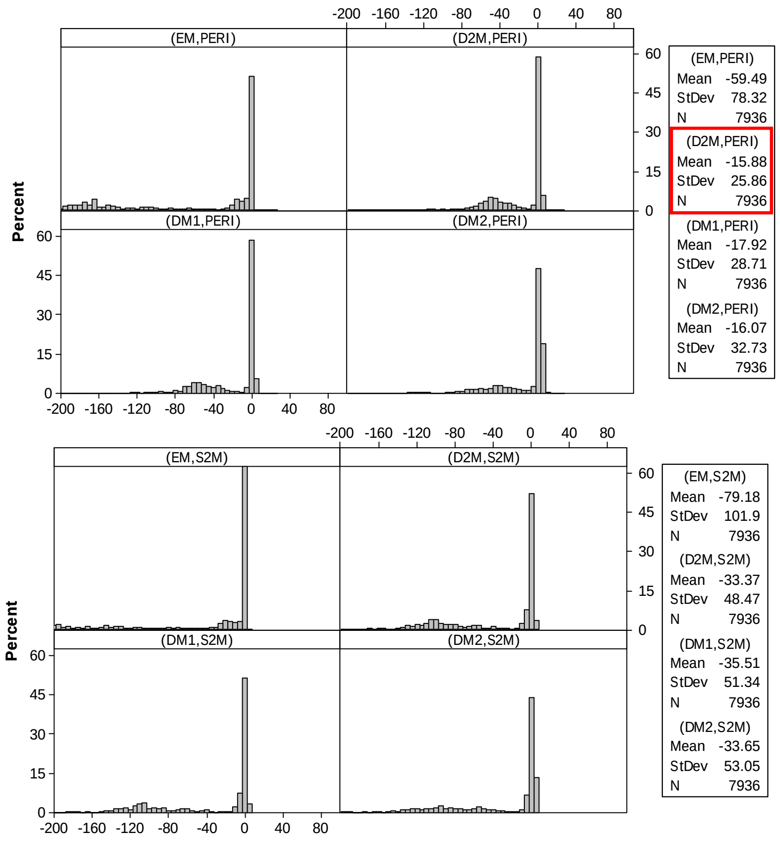


Figure 3.5: Endpoint slack error distribution reported by the signoff timer (x-axis: slack error (ps), y-axis: percentage of endpoints; testcase: *fft_fast*).

Model Selection. To select appropriate wire delay and slew models, we evaluate the timing discrepancy between our sizer and the signoff timer. We implement each model for wire delay and slew, then perform STA on ISPD-2013 benchmarks. Figure 3.5 illustrates endpoint slack error distributions for several combinations of delay models (EM, D2M, DM1 and DM2) and slew models (PERI and S2M). Total capacitance is used instead of the effective capacitance model. The plots show that (D2M, PERI) and (DM1, PERI) exhibit negligible error at around 60% of endpoints, while for other models this statistic is $< 50\%$. The error distribution for the (D2M, PERI) combined model exhibits smaller mean (-15.9ps) and standard deviation (25.9ps), which is why we select it for STA. We validate the overall optimization flow in Section 3.2.8 on ISPD-2013 contest benchmarks with a signoff timer.

3.2.2 High-Performance Gate Sizing

We follow the general outline of the Trident methodology [111] that is based on stochastic importance-sampling metaheuristics and sensitivity-guided optimization. A major improvement upon [111] is accounting for interconnect delay and additional constraints — both extensions require the development of several new algorithmic components and closed-loop control techniques.[6] During early optimization stages, our framework performs similar parameter sweeps (including *power exponent* and *commit ratio*) to those in [111], but with coarser steps, so as to accommodate slower STA and stringent runtime constraints.[7] In response to the inclusion of interconnect delay, sensitivity functions have been revised and additional optimization steps are developed.

Calibration-Free Early Optimization. Given that calibration with the signoff timer is time-consuming, we first "warm-up" metaheuristics with a low-accuracy internal timer in order to optimize parameters of individual search heuristics. When timing constraints are loose, this stage may be sufficient to produce feasible or near-feasible solutions quickly. In general, it also enables more effective use of parallel computing resources.

Offset-Based Timing Calibration. Moon et al. [177] introduced the idea of improving the accuracy of a given STA engine by periodically invoking a signoff timer and storing slack differences (offsets) at every timing endpoint. When the STA engine produces new estimates (e.g., during optimization), they are adjusted by slack offsets. Following up on this idea, we perform calibration with the signoff timer at every iteration of heuristic search. We use slack offsets both in full and incremental STA. As a result, there is a perfect agreement with signoff timing immediately after calibration, but the discrepancy slowly increases as cells are changed during gate sizing optimization. The frequency of calibration is determined by the maximal fraction of cells that are allowed to change. We evaluated possible thresholds of 5%, 10%, 15% and 30% of cells in terms of average slack errors. Figure 3.6 shows the results, e.g., with the 5% threshold we see that slack errors average $< 10ps$. Based on these observations, we have chosen 5% and 10% thresholds for our overall optimization flow.

Dedicated Critical Path Optimization. After GTR attempts to satisfy timing constraints, we further optimize critical paths by (i) downsizing of non-critical fanout cells, (ii) peephole optimization with Gray codes, and (iii) critical path optimization with heuristics.

[6]Compared to interconnect delay modeling, cell delay modeling is relatively straightforward with lookup table-based NLDMs. At the ISPD-2012 contest, most teams implemented fast internal STA engines that exactly matched *Synopsys PrimeTime* results on ISPD-2012 benchmarks.

[7]Thus, further runtime-quality tradeoffs are possible.

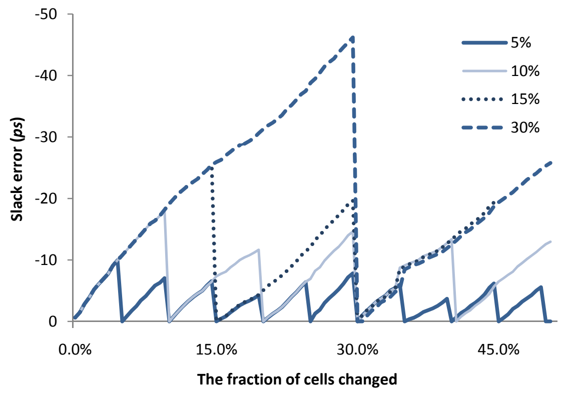
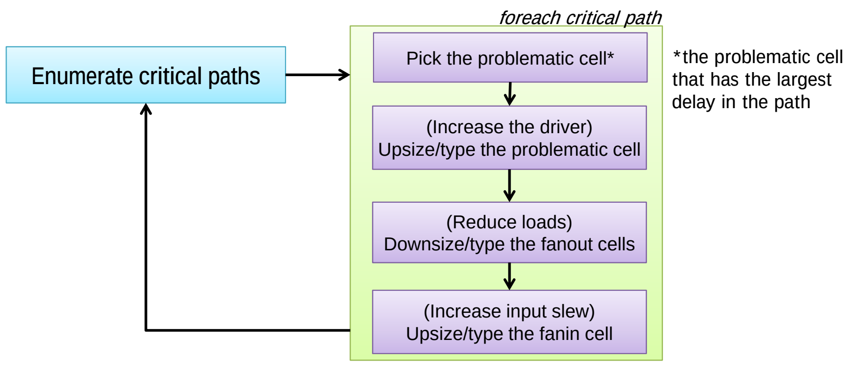


Figure 3.6: The impact of *calibration frequency* on *slack error* while sizing *fft_fast* benchmark.

(i) *Downsizing of non-critical fanout cells*. Large and low-V_t cells can be faster and can help reducing path delays, but their larger input capacitances degrade upstream slews and delays. The presence of interconnects aggravates this effect by increasing the overall capacitance. Given both cell and wire delay increase, upsizing alone is insufficient for reliable timing recovery. Our insight is that downsizing certain non-critical cells can reduce critical path delay. In particular, we focus on fanout cells of the cells lying on critical paths — downsizing these fanout cells reduces capacitance driven by critical cells. As a side effect, the delay of those fanout cells increases, thus they should not themselves lie on critical paths. We select downsizing candidates as fanout cells of critical path cells c based on the *sensitivity function*, $SF_{down} = size(c)/C_{out}(c)$, where $C_{out}(c)$ is the capacitance driven by cell c. If downsizing a candidate cell decreases negative slack, we restore previous size and continue to the next candidate.

(ii) *Peephole optimization using a Gray code.* We consider several cells at a time and exhaustively evaluate size combinations within a given radius of current sizes. For example, if three choices are considered for three cells — one size up, one size down and no change, — then 27 combinations would be evaluated. The use of a Gray code, i.e., traversing all size combinations by modifying one cell at a time, accelerates incremental timing analysis. In particular, our incremental STA engine performs timing updates faster when the amount of change is smaller.

(iii) *Critical path optimization with heuristics.* We optimize the delay of critical paths using several gate sizing methods. Cell delay in the critical path can be reduced in three ways: (a) increase the drive strength; (b) reduce output loads; and (c) improve input slew. These methods are implemented into our sizer as described in Figure 3.7, Algorithm 3 and Algorithm 4. In the procedure, critical paths are enumerated first. Then, a problematic cell that has the largest delay



Enumerate critical paths
foreach critical path
Pick the problematic cell*
(Increase the driver) Upsize/type the problematic cell
(Reduce loads) Downsize/type the fanout cells
(Increase input slew) Upsize/type the fanin cell
*the problematic cell that has the largest delay in the path

Figure 3.7: Critical path optimization with heuristics.

Algorithm 3 Critical path optimization with heuristics (*OptCritPath*).

```
Procedure OptCritPath(C)
Input : critical path $C$
Output : sizing solution $S$
1:  $UP\_GB \leftarrow init\_up\_gb$;
2:  $DN\_GB \leftarrow init\_dn\_gb$;
3:  $curTNS \leftarrow TNS$;
4:  while $(TNS < 0)$ do
5:      for each cell instance, $c_i$ in the netlist $N$ do
6:          $m.target \leftarrow c_i$;
7:          $m.sensitivity \leftarrow -cellDelay(c_i)$;
8:          $M \leftarrow M \cup \{m\}$;
9:      end for
10:     while $M.size()! = 0$ do
11:         Pick a modification $m$ with maximum $sensitivity$ in $M$;
12:         $UpsizeCellGreedy(m.target, UP\_GB)$;
13:         $DownsizeFOCellGreedy(m.target, DN\_GB)$;
14:         Pick the pin $p$ that has the largest input slew;
15:         Get fanin cells $FIs$ of pin $p$;
16:         for all cell, $fi$ in $FIs$ do
17:             $UpsizeCellGreedy(fi, UP\_GB)$;
18:             $DownsizeFOCellGreedy(fi, DN\_GB)$;
19:         end for
20:         $M \leftarrow M \setminus \{m\}$;
21:     end while
22:     Run STA to evaluate the current sizing solution $S$;
23:     if $prevTNS \leq curTNS$ then
24:         Decrease $DN\_GB$;
25:         Decrease $UP\_GB$;
26:     end if
27: end while
```

is picked from the most timing-critical path (in Lines 5–9, Line 11). To fix the problematic cell, cell upsizing (Line 12) and fanout downsizing (Line 13) are tried in turn. Since large input slews

Algorithm 4 Upsize cells and downsize cells procedures.

```
Procedure $UpsizeCellGreedy(c, UP\_GB)$
Input : target cell $c$, a guardband for upsizing $UP\_GB$
Output : sizing solution $S$
1: if $c$ is upsizable then
2:     Upsize cell $c$;
3:     $prevTNS \leftarrow curTNS$;
4:     Run $STA$ to evaluate the current sizing solution $S$;
5:     if $\Delta slack < UP\_GB$ or $curTNS < prevTNS$ then
6:         Revert cell $c$;
7:     end if
8: end if

Procedure $DownsizeFOCellGreedy(c, DN\_GB)$
Input : target cell $c$, a guardband for downsizing $DN\_GB$
Output : sizing solution $S$
1: Get fanout cells $FOs$ of cell $c$;
2: for each cell instance, $fo$ in $FOs$ do
3:     if $fo$ is downsizable then
4:         Downsize cell $fo$;
5:         $prevTNS \leftarrow curTNS$;
6:         Run $STA$ to evaluate the current sizing solution $S$;
7:         if $\Delta slack < DN\_GB$ or $curTNS < prevTNS$ then
8:             Revert cell $fo$;
9:         end if
10:    end if
11: end for
```

make large cell delays, we also optimize the fanin cells (Lines 16–19) to improve the input slew. Each cell sizing is made in a greedy manner, i.e., if the sizing makes a current TNS worse (Lines 5 and 7 in Algorithm 4), then we revert the sizing. Guardbands (UP_GB, DN_GB) are used to control cell changes. If the slack improvement ($\Delta slack$, Lines 5 and 7 in Algorithm 4) is not greater than the guardbands, our sizer does not commit the cell change. We have observed that our sizer cannot find a feasible solution if the cells are oversized too much. Thus, to avoid the oversizing, we use a larger guardband for upsizing than that for downsizing. In the loop in Figure 3.7, the optimization might get stuck if there is no cell to change. In this situation, we decrease the guardband (with the range of -5ps – -10ps) to allow more cells to be changed (Lines 23–25 in Algorithm 3). For the timing analysis, either ST or our timer can be used. Although our timer gives inaccurate timing estimates (it can be either pessimistic or optimistic depending on each path), as it is much faster than ST access, we can use our timer at the first round of optimization and then use ST to obtain accurate timing at the later round of the optimization procedure.

Sensitivity Functions. To identify the most promising cells to size, several stages of our optimization take into account (i) the direct impact of sizing a given cell on its slack, (ii) the required increase in leakage power, and (iii) the number of critical paths whose slack is improved. These parameters are combined into a *sensitivity score*, by which candidate cells are ranked. Thus, non-critical cells are not considered for upsizing during sensitivity-guided optimization, but small cells lying on numerous critical paths (bottleneck cells) are given higher priority. In practice, no single sensitivity function dominates other functions and the most accurate computations are prohibitively expensive. In Trident [111], the authors approximate the impact of single-cell changes on total negative slack. Significant efficiency is achieved by only propagating cell delay, and this abstraction works well for the ISPD-2012 contest infrastructure where only *gate* delays are computed. In contrast, the ISPD-2013 contest adds interconnect considerations and requires more comprehensive delay modeling. In particular, one must model slew degradation in wires on a timing path and the impact of slew on delay.

To track the impact of single-cell changes more accurately, we calculate slack updates considering both delay and slew. In our two GTR stages, we account for slack change ($\Delta slack$), the number of paths passing through the cell ($\#paths$), and the change in leakage ($\Delta leakage\_power$). The latter is raised to power ($power\_exponent$) — a configurable parameter for the sensitivity function (SF).

$$SF_{GTR} = \frac{\Delta slack \times \#paths}{\Delta leakage\_power^{power\_exponent}} \tag{3.10}$$

The PRFT stage uses sensitivity functions from Trident [111].

3.2.3 Overall Optimization Flow

Figure 3.16 introduces our optimization flow. We initially rely on an internal timer. A multi-threaded metaheuristic optimizes individual parameters of lower-level search heuristics, similar to the GTR (global timing recovery) stage in Trident [111], but with interconnect delay models and constraints. The second stage performs timing calibration with the signoff timer and seeks to produce a feasible solution with respect to signoff timing. This stage also uses techniques from GTR [111], but is more accurate and more constrained in terms of runtime. The third stage performs power reduction with feasible timing and roughly corresponds to PRFT in [111].

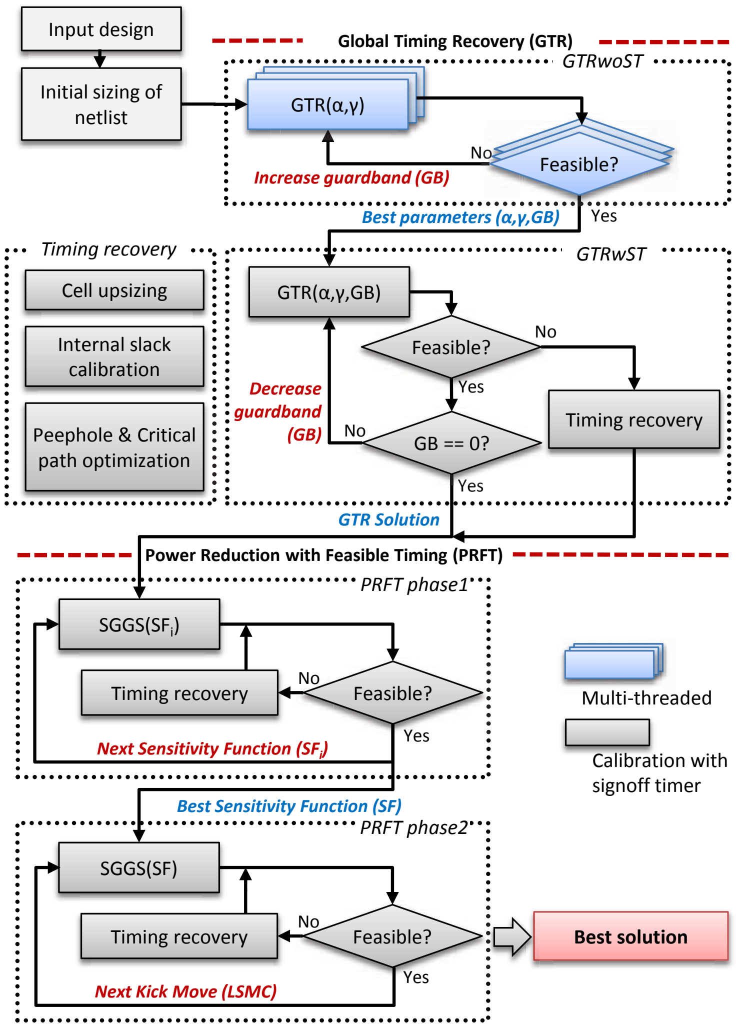


Figure 3.8: Overall optimization flow.

Stage 1. GTR without a Signoff Timer (GTRwoST). This stage seeks to satisfy timing constraints with respect to our internal STA engine. Due to discrepancies with the signoff timer, this solution may not be signoff-feasible. Further improvements will be performed by slower yet more accurate optimization stages. However, much of the work in exploring the overall solution space is performed at this early stage with a fast timer. Moreover, this stage optimizes the configurations of sensitivity functions (e.g., power exponent α and commit ratio γ in Trident [111]) by metaheuristic search based on importance sampling. If a timing feasible solution cannot be found, *guardband* (GB)[8] is applied until a feasible solution is found with a loose timing constraint.

[8]A positive (negative) GB means tighter (looser) timing constraint than the original clock period.

Stage 2. GTR with a Signoff Timer (GTRwST). With the best parameters from first phase of GTR, this phase finds a feasible solution based on the signoff timer. To remove the timing discrepancy versus the signoff timer, we calibrate internal slack values as described in Section 3.2.2. In this step, our sizer tries to find a feasible solution with the GB obtained from GTRwoST. If our sizer finds a feasible solution, it increases the GB until that the GB is zero or that it cannot find a feasible solution. If no feasible solution is found in GTRwST, our sizer invokes timing recovery with dedicated critical path optimization techniques that are discussed previously. The use of GB helps to avoid excessive oversizing in GTR stages and balance timing paths. If our sizer is forced to recover timing, it will keep upsizing cells on critical paths more, which might degraded timing due to side effects (e.g., interconnect delays, large load capacitances). By decreasing GB to loose timing constraints for all timing paths, the oversizing can be avoided and the slacks of all paths may become evenly distributed.

Stage 3. Power Reduction with Feasible Timing (PRFT) is performed in two phases. Starting with the best feasible solution from GTRwST, our sizer performs a sensitivity-guided greedy downsizing to reduce leakage power subject to timing constraints. The greedy optimization is attempted with different sensitivity functions, and violations that occur as a result of downsizing are fixed with timing recovery iterations (upsizing). We keep track of the best seen solution from different sensitivity functions and carry it over to the second phase (or otherwise, the configuration with the smallest amount of violations). Downsizing occasionally introduces constraint violations, which we fix on-demand with iterative upsizing, peephole optimization and critical path optimizations (similar to GTRwST). The latter two optimizations are also capable of improving leakage. The *second phase of PRFT* uses the most successful sensitivity function from the first phase and performs additional greedy (down) sizing with kick-moves. Kick-moves described in [111] can be viewed as large-step Markov chain (LSMC) optimization [172].

3.2.4 Signoff Timer Interface

Given frequent timing calibration, the interface between the internal timer and the signoff timer must be efficient. ISPD-2013 contest infrastructure [19] includes a file-based interface that requires saving and reading large files, and starting a new instance of the signoff timer on each invocation. This interface is particularly inefficient for incremental STA and when performing targeted optimization of critical paths. Instead, we interface with the signoff tool using a *Tcl*-socket interface, similar to the one in UCSD SensOpt [35].[9] Our interface is illustrated in

[9]Unix sockets are a standard mechanism for interprocess communication. Commercial signoff timing tools, such as *ExtremeDA GoldTime* [10] and *Cadence Encounter Timing System* [2], provide API access to the core timing engine (e.g., attributes of pins and

Table 3.6: Results for our sizer (GTR + PRFT) on ISPD-2013 benchmarks in *fast* mode.

benchmarks	leakage (*mW*)		CPU time		1st place team		2nd place team		3rd place team	
	GTR	PRFT	total (*min*)	limit (*min*)	leakage (*mW*)	CPU (*min*)	leakage (*mW*)	CPU (*min*)	leakage (*mW*)	CPU (*min*)
usb_phy_fast	1.78	**1.60**	**0.14**	48	1.64	0.25	2.63	2.71	1.61	0.58
usb_phy_slow	1.12	**1.08**	**0.13**	48	1.08	0.25	1.09	0.67	1.08	0.40
pci_b32_fast	155.39	105.30	**2.40**	48	112.64	2.91	504.98	48.03	**96.11**	23.61
pci_b32_slow	67.60	60.20	**1.80**	48	60.17	2.25	136.95	48.02	**57.89**	9.58
fft_fast	678.90	342.10	**5.50**	48	361.30	6.93	974.60	48.03	**224.53**	30.54
fft_slow	144.40	96.25	**3.48**	48	98.15	4.1	418.99	48.03	**90.32**	22.78
cordic_slow	1546.30	394.70	**25.70**	50	563.09	26.16	1961.09	60.06	**323.71**	49.65
des_perf_slow	616.40	391.90	**19.23**	72	395.86	20.48	2823.88	72.26	**353.80**	67.85
edit_dist_fast	1260.40	**689.90**	**42.20**	84	704.82	44.25	9769.38	84.18	—	—
edit_dist_slow	721.30	**487.90**	**31.60**	84	489.47	33.21	7485.66	84.18	90.31	22.78
matrix_m_slow	1118.48	**562.40**	**77.30**	84	570.74	65.17	7540.34	84.16	—	—
netcard_fast	5764.40	**5277.40**	**190.77**	310	—	—	—	—	—	—
netcard_slow	5371.04	**5184.20**	**148.60**	310	5371.10	336.30	—	—	—	—

Figure 3.9, including client-server *Tcl* socket code [32].

When timing calibration is initiated, we launch the signoff timer and open a Unix socket. An open socket allows a program to send commands (e.g., cell sizing and timing query commands) to the signoff timer and receive data (e.g., updated transition time and slack). Changes made to gate sizes during optimization (GTR and PRFT) are communicated to the signoff timer, which returns results of incremental STA used to re-calibrate our internal timer. In our sizer, communications with the signoff timer are always performed in conjunction with the internal timer.

3.2.5 Handoff Between Optimization Stages

As noted earlier, each optimization stage in our sizer pursues its own goals, often using parallel search, and hands off the best solutions found to the next stage. This modularity is not only convenient for software development, maintenance and testing, but also allows us to combine optimizations with very different runtime-quality tradeoffs as well as carefully tune each stage for reliability and performance. Two GTR stages seek violation-free (timing feasible) solutions; GTRwoST is more computationally efficient than GTRwST, but less accurate. PRFT stage iteratively reduces total leakage power while respecting timing constraints. Combining

arcs in the timing graph) through sockets.

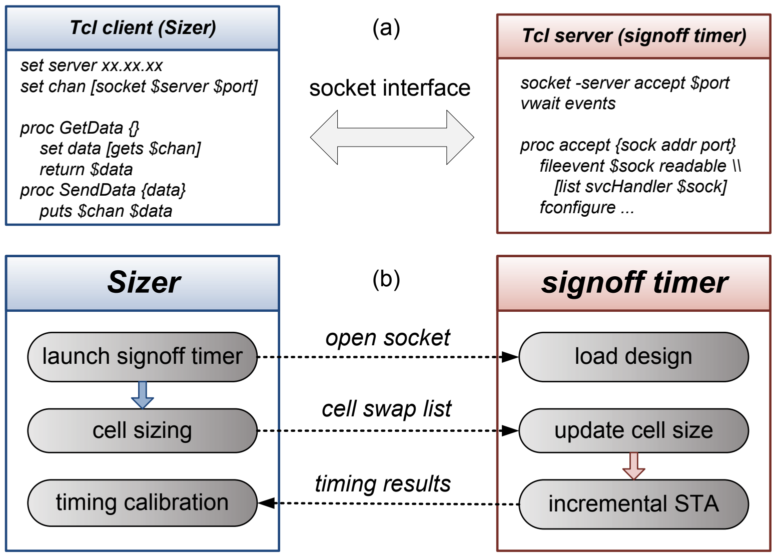
Table 3.7: Results for our sizer (GTR + PRFT) on ISPD-2013 benchmarks in *normal* mode.

benchmarks	clock period (ps)	# of cells	leakage (mW) GTR	leakage (mW) PRFT	runtime total (min)	runtime limit (min)	leakage from top ranked teams 1st place (mW)	2nd place (mW)	3rd place (mW)
usb_phy_fast	300	608	1.63	1.77	**0.14**	240	1.61	1.68	6.55
usb_phy_slow	450	608	1.11	1.07	**0.13**	240	1.08	1.07	1.12
pci_b32_fast	750	30603	145.5	102.52	**2.40**	240	**96.51**	106.93	—
pci_b32_slow	1000	30603	65.1	59.07	**1.80**	240	**57.89**	59.26	77.18
fft_fast	1400	32766	592.02	301.84	**5.50**	240	**226.20**	321.45	637.81
fft_slow	1800	32766	128.01	93.24	**3.48**	240	**90.34**	97.71	106.68
cordic_fast	2626	42903	3893.34	**1098.43**	**25.70**	300	—	—	—
cordic_slow	3000	42903	1546.35	492.16	**25.70**	300	**323.79**	443.61	1077.73
des_perf_fast	1140	113112	3662.49	**1498.12**	**19.23**	360	—	—	—
des_perf_slow	1300	113112	513.87	368.62	**19.23**	360	**353.00**	380.44	2391.83
edit_dist_fast	3000	126665	1049.54	621.94	**42.20**	420	**596.32**	639.01	—
edit_dist_slow	3600	126665	632.52	466.44	**31.60**	420	**447.40**	468.45	—
matrix_m_fast	2200	156440	4571.79	**2339.75**	**77.30**	420	—	—	—
matrix_m_slow	2800	156440	957.26	501.58	**77.30**	420	**469.73**	512.85	1381.37
netcard_fast	2000	982258	5764.43	**5278.1**	**190.77**	1650	5317.84	—	19152.00
netcard_slow	2400	982258	5371.04	**5179.95**	**148.60**	1650	5302.27	5371.10	5245.66

such diverse optimization stages requires particular attention to the handoff between them. Note that for all of benchmarks where timing constraints were satisfied after GTRwST stage (Table 3.7), GTRwoST finds timing-feasible solutions in eight, and the signoff timer takes $< 50\%$ runtime in those cases. Thus, using our internal timer without calibration during early search is compatible with later optimization stages, helps reduce runtime and allows the sizer to explore a larger solution space.

3.2.6 Scalability

Support for Parallelism. The initial search for a timing-feasible configuration (GTRwoST) is performed in parallel. Our implementation uses up to 16 threads. As these threads are essentially independent, further scalability is mostly limited by memory usage and diminishing returns in terms of solution quality. While searching for feasible cell sizes/configurations, GTRwoST identifies best parameters for search heuristics in GTR, as explained in Section 3.1. To exploit parallelism in further stages, it is important to parallelize invocations of the signoff timer and have multiple licenses available. The file-based interface provided with the ISPD-2013 contest



```
set server xx.xx.xx
set chan [socket $server $port]

proc GetData {}
    set data [gets $chan]
    return $data
proc SendData {data}
    puts $chan $data
```

```
socket -server accept $port
vwait events

proc accept {sock addr port}
    fileevent $sock readable \\
      [list svcHandler $sock]
    fconfigure ...
```

Figure 3.9: Socket interface between our sizer and the signoff timer: (a) *Tcl* socket code and (b) timing calibration using the socket interface.

infrastructure does not support parallel invocation of the signoff timer. We found such extensions challenging with our socket-based interface as well, mostly due to non-deterministic race conditions. Assuming a reliable infrastructure for parallelism, last-stage local optimizations appear *a priori* amenable to parallel execution. However, one must first study runtime breakdown and identify bottlenecks.

Runtime Breakdown and Bottleneck Analysis. Figure 3.10 shows the runtime breakdown of our sizer for individual ISPD-2013 benchmarks and the fraction of runtime taken by the signoff timer. In both fast and normal modes, difficult netlists such as *cordic* and *edit_dist* take longer in GTRwoST because it is harder to satisfy timing constraints, even with respect to our internal timer. Once a timing-feasible solution is found, GTRwST runtime is less sensitive to the difficulty of the benchmark, even though GTRwST is much slower due to calibration with the signoff timer. This trend is also apparent in the percentage contribution of the signoff timer being comparatively small for these benchmarks in both modes. When GTRwoST does not find a timing-feasible solution (*pci_b32_fast*, *fft_**), relatively more time is spent in signoff timer calls.

3.2.7 Comparisons to Prior Research

ISPD-2012 and 2013 Gate Sizing Contests [184] [185] have dramatically changed the landscape of research in the field. In particular, the benchmarking infrastructure developed by

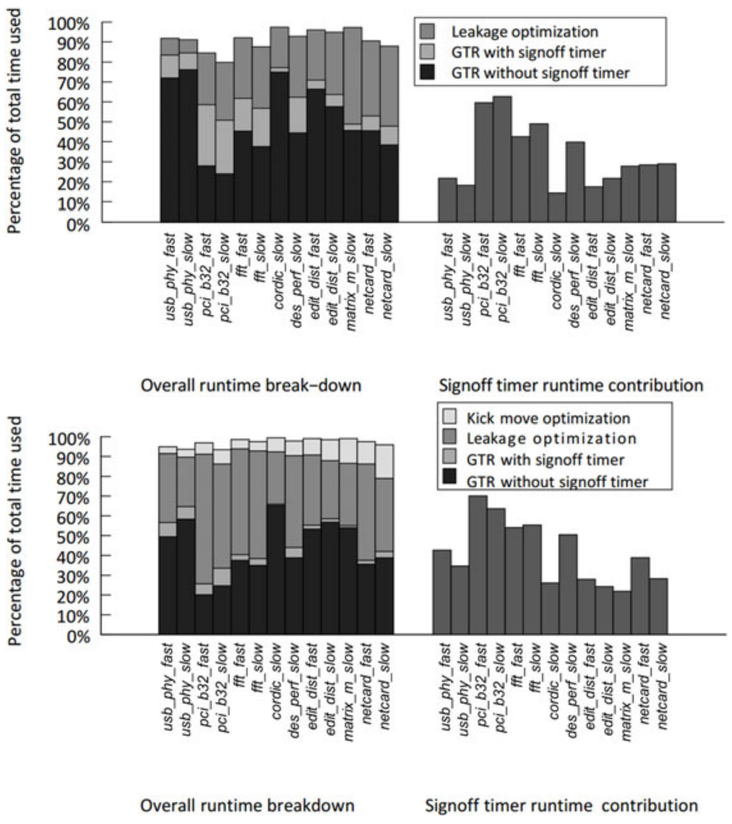


Figure 3.10: Runtime breakdown for our sizer on ISPD-2013 benchmarks in fast mode (above) and normal mode (below).

Intel researchers does not have academic precedents in terms of

- using discrete gate sizes and V_t assignment,
- relying on an industry-standard signoff timer,
- increasing the scale of optimization to netlists with hundreds of thousands of cells,
- using realistic technology models (cell timing, drive, power) and timing constraints, and
- imposing capacitance and slew constraints.

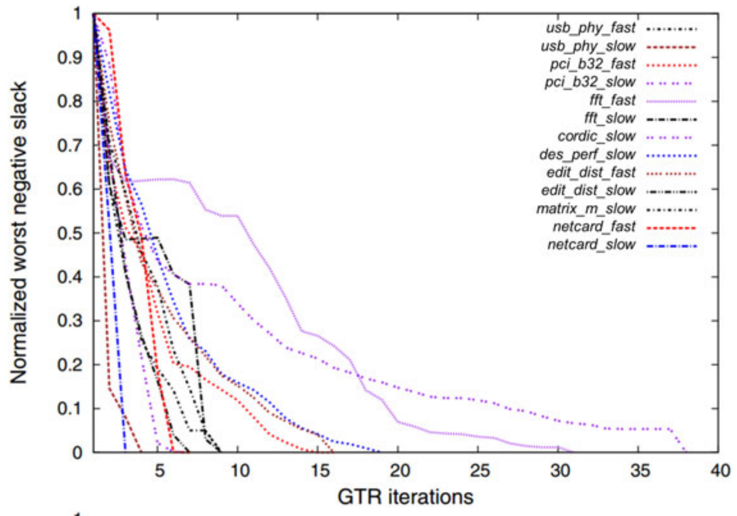
UCSD SensOpt [35] uses sensitivity-guided optimization for post-layout discrete gate sizing. It communicates with an industry signoff timer through a *Tcl* socket interface. UCLA OA Sizer [33] implements greedy optimization, linear programming, Lagrangian relaxation and dynamic programming, while relying on the OAGear-Static-Timer [9]. The two ISPD contests themselves attracted several dozen research teams from all over the world, but few teams produced competitive solutions. Consequently, few publications describe relevant algorithms.

Core optimization techniques used in our work — metaheuristic optimization with importance sampling and sensitivity-guided search — have been developed in [111], and we extend them. Our implementation runs approximately ten times faster on ISPD-2012 benchmarks than results in [111] indicate, but results in slightly higher leakage power. Unlike [111], we keep track of interconnect delay and slew, and per-pin timing slack (plus, offsets with respect to a signoff timer). Given the relatively simple timing models used in the ISPD-2012 contest, [111] did not need to directly invoke a signoff timer. Our optimization must invoke a signoff timer that leads to major structural changes; timing calibration with the signoff timer is required to make metaheuristics from Section 3.1 successful when interconnects are considered.

The performance of Lagrangian relaxation techniques on ISPD-2012 benchmarks is described in [161] [165]. Empirically, runtimes show significant improvement over [111], but at the cost of increased leakage. Interconnect delay modeling and optimization are not discussed in [161] [165]. These considerations completely change the nature of the overall optimization, making it impossible to reliably extrapolate the performance of Lagrangian relaxation to the ISPD-2013 benchmark suite, as several algorithmic components must be developed to enable a full comparison.

3.2.8 Empirical Validation

Optimization Trajectories Pursued by Our Sizer. Figure 3.11 illustrates the reduction of normalized worst negative slack (WNS) with GTR iterations during GTRwoST and GTRwST on multiple ISPD-2013 benchmarks. GTRwoST reduces WNS quickly because many cells can be upsized to improve circuit delay. Figure 3.12 illustrates the progress of normalized leakage power with GTR iterations during GTRwoST and GTRwST. When our sizer does not find a feasible solution in GTRwoST in the first 8-10 iterations, leakage power quickly increases with the number of changes made to cells, but then saturates when few possible cell moves are available.
The Impact of Timing Calibration. We evaluate leakage-power optimization with PRFT in five cases: (i) frequent calibration (after every 5% of cells change), (ii) one-time calibration before PRFT, (iii) no calibration, (iv) using a 5ps guardband (GB) without calibration, and (v) using a 10ps GB without calibration. For each case, Figure 3.13 shows leakage (normalized to case (i) after timing recovery), total negative slack (TNS) and worst negative slack (WNS) after PRFT and timing recovery. Frequent calibration achieves smallest leakage power without timing violations.



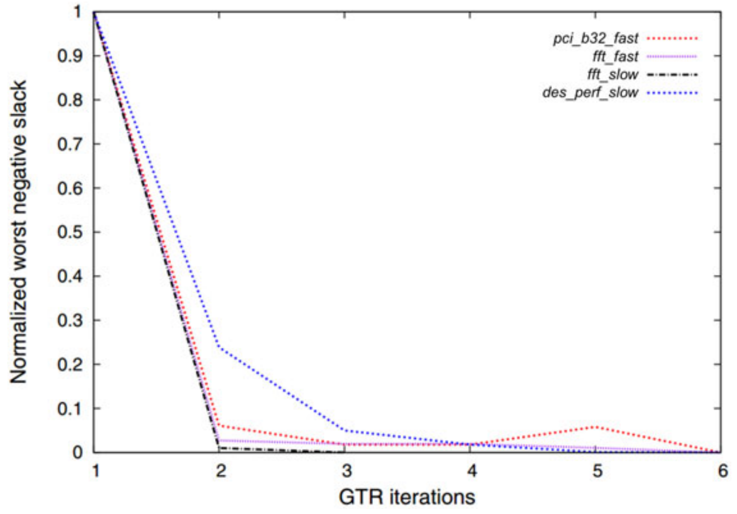


Figure 3.11: Normalized WNS during GTR without (above) and with (right) signoff timer in fast mode.

Without calibration (Case 3), solutions may be infeasible with respect to signoff timing. Feasible solutions can sometimes be produced without any calibration by using a guardband (Cases 4 and 5), but this pessimism will limit leakage reduction. For example, violation-free solutions are produced with a $10ps$ guardband for *pci_b32_fast* and $5ps$ for *fft_fast*, but leakage power increases by 6% versus frequent calibration, due to the excessive cell upsizing. At the PRFT stage, WNS and TNS are larger (better) with calibration due to pessimism in our internal timer (Figure 3.5). One-time calibration exhibits the largest errors in WNS and TNS, suggesting that the timing discrepancy with the signoff timer increases as more cells undergo size changes.

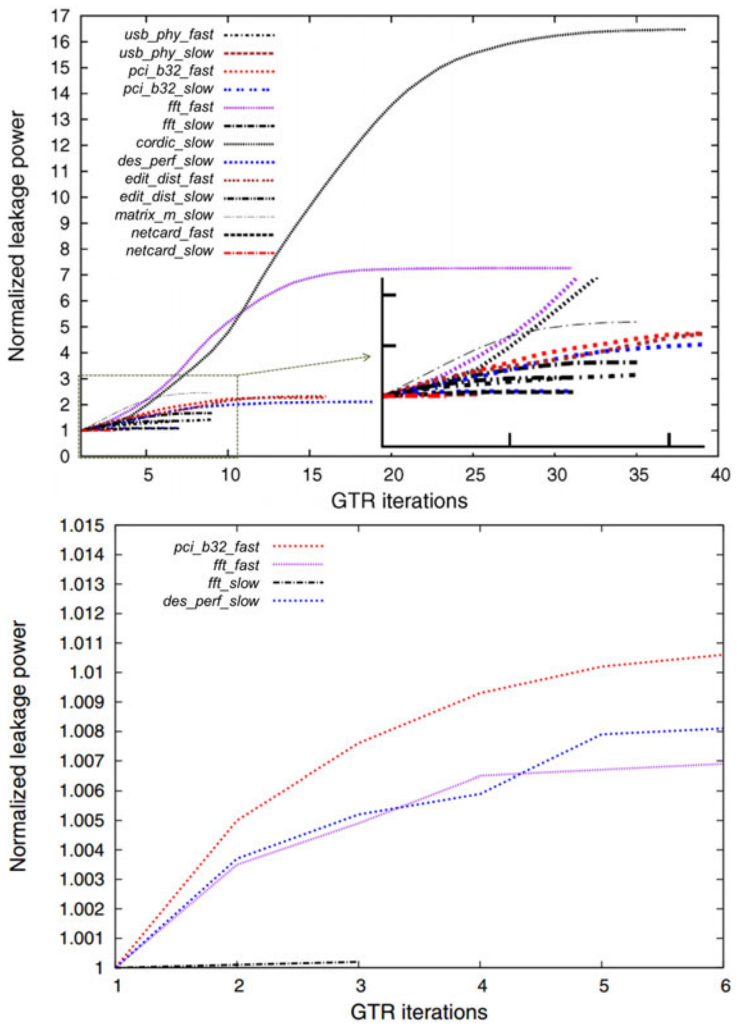


Figure 3.12: Normalized leakage during GTR without (above) and with (below) signoff timer in fast mode.

Comparisons to ISPD-2013 Contest Results. Tables 3.6 and 3.7 report our results on ISPD-2013 benchmarks and official ISPD-2013 contest results. Our sizer finds violation-free solutions for all test benchmarks in normal mode. By the primary contest metric (leakage power), our sizer places between the first- and second-place teams. Our results with the secondary metric (based on leakage power and runtime) are ahead of the first-place team; we achieve smaller leakage, spend less runtime, and find feasible solutions on more benchmarks.

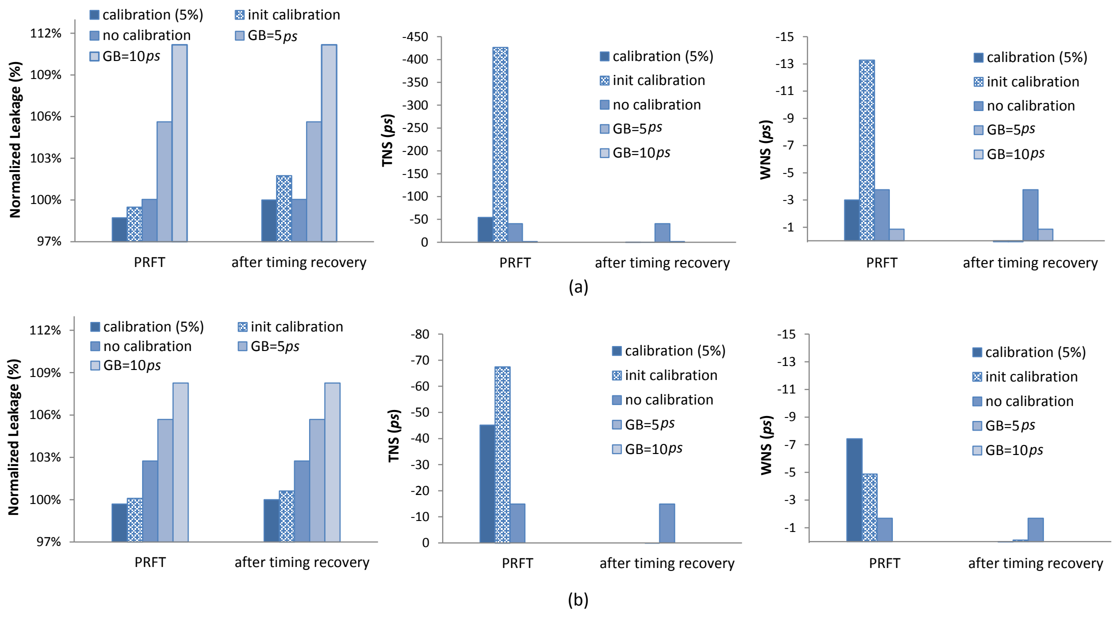


Figure 3.13: The impact of calibration on leakage reduction and timing recovery: (a) *pci_b32_fast*, (b) *fft_fast*.

3.3 Construction of Realistic Benchmarks with Known Optimal Solutions

As we have seen, the *sizing problem* in VLSI design seeks to assign design parameters (width and/or length and/or threshold voltage) to each gate, so as to optimize timing, area and/or power of the design subject to constraints. The problem has been extensively studied, and a number of heuristics have been proposed. However, finding an optimal gate sizing solution is NP-hard [181], and the suboptimality of sizing solutions has never been adeqautely quantified and analyzed for existing heuristics. Real circuits have unknown optimal solution quality, and thus do not shed much light on heuristic suboptimality. On the other hand, artificial circuits with known optimal solution quality – along with any implications they might have for suboptimality of heuristics – are viewed as unrealistic. Thus, the need for further research and development on gate sizing methods has been unclear. In this section, we focus on sizing for leakage reduction, and propose a new method for generating *realistic sizing benchmark circuits with known optimal sizing solutions*, which enables systematic and quantitative comparisons of available gate sizing heuristics.

For evaluation of CAD heuristics, several methods of generating synthetic benchmarks

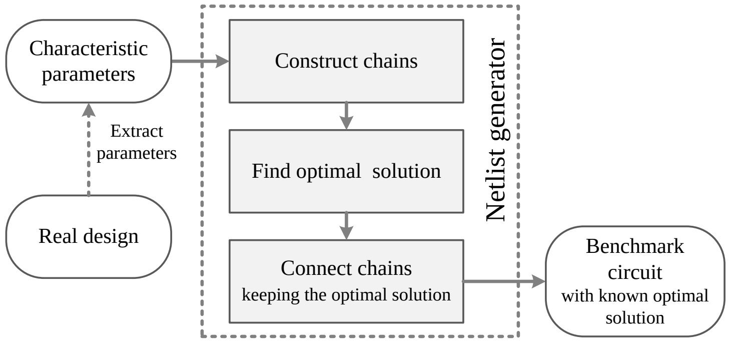

Characteristic parameters
Extract parameters
Real design
Construct chains
Find optimal solution
Connect chains keeping the optimal solution
Netlist generator
Benchmark circuit with known optimal solution

Figure 3.14: Generation of benchmark circuits with known optimal solutions.

that match real designs have been proposed. Darnauer and Dai [76] generate random benchmark circuits based on Rent's rule. Their code generates random circuits with a specified number of inputs, outputs, blocks, terminals per cell, and Rent parameter. Hutton et al. [118] define properties such as size, delay, physical shape, edge-length distribution and fanout distribution, and generate combinational circuits to match a given parameterization. Stroobandt et al. [220] provide parameterized (by Rent exponent and net degree distribution) benchmarks with user-selected library cells. With these synthetic benchmarks, various CAD heuristics can be compared to each other, but the suboptimality of the heuristics cannot be measured.

Suboptimality of existing heuristics has been studied for VLSI problems such as synthesis, placement, partitioning, and buffer insertion. Hagen et al. [104] show how to quantify the suboptimality of heuristic algorithms for NP-hard placement and partitioning problems arising in VLSI layout. They construct scaled instances from the original problem and execute the heuristic. If the heuristic solution cost increases at a faster rate than the scaling of the heuristic instance itself, then a lower bound on the heuristic's suboptimality can be inferred. PEKO (placement examples with known optimal solutions) [57] and its extension PEKU (placement examples with known upper bounds) [74] enable estimation of suboptimality of several timing-driven placement algorithms; the core approach involves perturbing an original design to obtain a new design with similar topological properties and a known optimal solution.

Our present work builds on the recent work of Gupta et al. [99], which to our knowledge is the only work in the literature to address suboptimality of (leakage-driven) gate sizing heuristics. The authors of [99] (i) propose *eyechart* benchmark circuits which can be optimally sized using dynamic programming methods, and (ii) use eyecharts to evaluate the suboptimalities of several gate sizing algorithms. However, [99] does not address the difference or similarity between real designs and eyechart circuits. In [99], the eyechart circuits are built from three basic

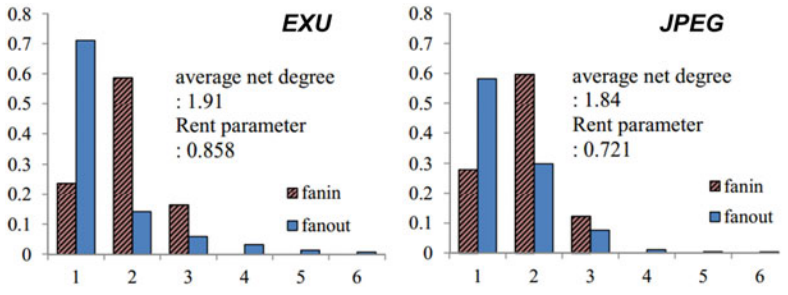


Figure 3.15: Circuit characteristics for two real designs (*EXU*: OpenSPARC T1 execution unit; *JPEG*: JPEG encoder).

topologies – chain, mesh and star – and the resulting topologies differ substantially from those of real designs in terms of Rent parameter, path length and other parameters.[10] Thus, the eyecharts may be helpful in measuring suboptimality of heuristics, but do not have clear implications for heuristic performance on real designs. Furthermore, [99] does not provide any automated flow for eyechart circuit generation.

In this section, we provide more realistic benchmarks with known optimal solutions for gate sizing problems. Figure 3.14 shows the flow of our benchmark circuit generation. (i) To create a circuit with known optimal gate sizing solution, we construct multiple chains (for which optimal sizing solutions can be found by dynamic programming), then connect the chains with inter-chain nets *without affecting the property of having a known leakage-optimal sizing solution*. (ii) During the circuit construction, circuit topology is constrained according to user-specified parameters (path depth, and fanin/fanout distributions) so that the constructed benchmarks show similar characteristics to real designs. (iii) The inter-chain connections can be added in many possible ways, which gives the potential for greater topological diversity than the previous construction of [99].

3.3.1 Benchmark Considerations

For benchmark circuit generation, *realism* and *tractability to analysis* are opposing goals since (i) determining the optimum solution is usually intractable in real designs, and (ii) constructions for which optimum solution costs are known are often considered "artificial" [104]. We begin by considering this tension between realism and tractability in benchmark circuits.

First, to construct a *realistic benchmark*, we must use characteristic design parameters

[10]Eyecharts used in [99] have large depth (650 stages) and small Rent parameter (0.17). Table 3.11 below shows that real designs have path depths of 20 – 70, and Rent parameter values of 0.72 – 0.86.

in the benchmark generation. Many works in the literature classify or parameterize circuits according to an empirical power-law scaling phenomenon that governs statistics of interconnects among and within subcircuits (cf. the well-known Rent parameter or Rent exponent [153]). Distributions of net degrees, or of the numbers of fanins and fanouts per cell instance, are additional important circuit characteristics. Figure 3.15 shows circuit characteristics (fanin, distribution, fanout distribution, average net degree and Rent parameter) of two real design blocks; each shows different characteristic parameters. In our work, to construct realistic benchmarks we use four design characteristic parameters: (i) number of primary (PIs and POs), (ii) (maximum) path depth, (iii) fanin distribution, and (iv) fanout distribution. These four parameters can be configured in advance, and our benchmark generator makes net connections according to the given parameters subject to a given (setup) timing constraint.

Second, for generated benchmarks to permit *known optimal gate sizing solutions*, some simplifications are required. The eyechart work of [99] achieves tractable optimal solutions by simplifying the cell timing library to eliminate slew dependency. In this work, we consider both input slew and output capacitance in the library (*Synopsys Liberty*) characterization, but ignore interconnect delay. By omitting interconnect delay, it is possible to find an optimal sizing solution for simple (chain) topologies using dynamic programming (DP). Compared to the previous eyechart work, we would also like to consider all possible topologies so as to satisfy our goal of realistic benchmark topologies.

Our key insight is that instead of separating the netlist generation and optimization stages as in the eyechart approach, we can find optimal cell sizes *during* the benchmark netlist generation. We then augment the benchmark circuit without disturbing the existing, known optimal solution. More precisely: (i) we construct gate-chains to realize a specified number of primary input/output ports and a specified path depth; (ii) we add fanins and fanouts to cells on the chains to match given fanin and fanout distributions; (iii) we find optimal sizing solutions for cells in each chain using DP; and (iv) finally, we connect the chains using *connection cells* while preserving the optimal gate sizing solution of each chain.

3.3.2 Benchmark Generation Details

Table 3.8 shows input parameters to our benchmark generation process. To simplify the procedure, we assume that the numbers of primary inputs and primary outputs are both equal to N. I and O respectively indicate the maximum numbers of fanins and fanouts to any given cell instance. Given the five input parameters, our flow generates N chains, each of which consists of K cells. We connect the chains using *connection cells* according to the prescribed fanin and fanout distributions. The result is a netlist with $K \times N + C$ cells, where C is the number of connection cells.

Table 3.8: Input parameters for benchmark generation.

parameter	description
T	timing path delay upper bound
N	number of primary inputs/outputs
K	(maximum) data path depth
$fid(i)$	fanin distribution (# of cells with i = 1, ..., I fanins)
$fod(j)$	fanout distribution (# of cells with j = 1, ..., O fanouts)

To generate the circuit properly, the input parameters must satisfy three constraints.

1. The timing budget T should be larger than minimum delay of a chain of K cells.
2. The total numbers of fanins and fanouts in the circuit should satisfy the equality of Equation (3.11).
3. The prescribed proportion of single-fanout cells, $fod(1)$, should be larger than the proportion of connection cells since connection cells have only one fanout.

We note that in real circuit designs (such as shown in Figure 3.15), fanout distribution tends to follow a power law, with $fod(1)$ typically greater than 0.6. Thus the third constraint above can be easily satisfied in realistic benchmarks.

$$\sum_{i=1}^{I} i \times fid(i) = \sum_{o=1}^{O} o \times fod(o) \tag{3.11}$$

Algorithm 5 describes the procedure of benchmark generation. In the pseudocode, $gate(i, j)$ represents a gate at the j^{th} stage of the i^{th} chain. G_{co} is the set of connection cells. G_{fi} is the set of gate cells with open fanin ports. $DP(G_{chain}, T)$ is a dynamic programming procedure which finds an optimal cell sizing to minimize leakage power subject to the timing constraint T. We consider arrival times at the output side of any given gate, e.g., the arrival time at the output of gate g is denoted by a_g. Cell delay along the timing arc of cell g from the input that is connected to cell c is denoted by d_g^c. Finally, net delay along the net connecting c and g is denoted by $w_{c,g}$.

Algorithm 5 Netlist generation flow.

```
Procedure NetlistGen(T, K, N)
Input : timing budget $T$, number of depths $K$, number of chains $N$
Output : netlist with known optimal solution
1: Initialize $gate(i, j)$, where $i = 1, ..., N$ and $j = 1, ..., K$;
2: $G_{co} \leftarrow \emptyset$, $G_{fi} \leftarrow \emptyset$;
3: for $j = 1$ ; $j \leq K$ ; $j \leftarrow j + 1$ do
4:     for $i = 1$ ; $i \leq N$ ; $i \leftarrow i + 1$ do
5:         Assign fanin number to $gate(i, j).fanin$;
6:         Assign fanout number to $gate(i, j).fanout$;
7:         for $k = 2$ ; $k \leq gate(i, j).fanout$ ; $k \leftarrow k + 1$ do
8:             Attach connection gate $c$ to $gate(i, j)$;
9:             $G_{co} \leftarrow G_{co} \cup \{c\}$;
10:        end for
11:        if $gate(i, j).fanin > 1$ then
12:            $G_{fi} \leftarrow G_{fi} \cup \{gate(i, j)\}$;
13:        end if
14:    end for
15: end for
16: for $i = 1$ ; $i \leq N$ ; $i \leftarrow i + 1$ do
17:    $G_{chain} \leftarrow gate(i, j)$, where $j = 1, ..., K$;
18:    $DP(G_{chain}, T)$; // find optimal gate size under $T$
19: end for
20: Update timing for all gates ($gate(*)$ and $G_{co}$);
21: while $G_{co} \neq \emptyset$ do
22:    Select gate $c$ from $G_{co}$ with maximum arrival time;
23:    for each gate $g \in G_{fi}$ do
24:        Select gate $g$ with minimum arrival time;
25:        if $a_c + w_{c,g} + d_g^c \leq a_g$ then
26:            Connect $c$ and $g$;
27:            $G_{fi} \leftarrow G_{fi} \setminus \{g\}$;
28:            break;
29:        end if
30:    end for
31:    $G_{co} \leftarrow G_{co} \setminus \{c\}$;
32: end while
```

First, we generate N chains, each with depth K (Lines 1–15), as shown in Figure 3.16(a). For each of the $K \times N$ cells, we assign (i.e., instantiate) a gate according to the fanin distribution fid (Line 5). Cells in the first stage ($stage_1$) should be assigned one-input gates. Then, we assign the number of fanouts to the output of each cell (Line 6). Cells in the last stage ($stage_k$) have a single fanout. For remaining cells, the number of fanouts is assigned according to the fod. We have explored two alternative strategies for the fanin and fanout assignments: (i) *arranged assignment*, which assigns larger fanins to later stages and larger fanouts to earlier stages, and (ii) *random assignment*, which assigns fanins and fanouts in arbitrary order. The arranged assignment improves connectability among the chains, while the random assignment improves diversity of the resulting topology.

Second, we attach connection cells to open fanouts (Lines 7–10), as illustrated by the red lines in Figure 3.16(a). The number of connection cells, C, is the same as the number of open fanin ports, as expressed by Equation (3.12).

$$C = K \times N \sum_{i=1}^{I} (i-1) \times fid(i) \tag{3.12}$$

The fanin number of connection cells follows the fanin distribution (fid). For connection cells which have more than one fanin, open fanouts in the same stages are connected to the connection cell (Lines 11–13), as illustrated in Figure 3.16(b).

After attaching all connection cells, we perform dynamic programing (DP) with timing budget T for each chain (Line 18). The DP finds the optimal gate sizing which minimizes the leakage power for the chain. After the gate sizing, the sizes of attached connection cells will be set to minimum possible values since they do not have a timing constraint.

Finally, we connect all connection cells to other cells having open fanin ports (Lines 21–32). Before connecting them, the arrival time for each cell is computed by static timing analysis (STA), which is run with the timing budget T (Line 20). Connections between connection cells and open fanin cells are made only if the timing constraints are satisfied. In Figure 3.16(c), cell c and cell g can be connected when the arrival time of g via c ($a_c + w_{c,g} + d_g^c$) is less than the arrival time of g through the chain path. If timing slack of the connection cell is large, sizing heuristics can recognize them easily and the problem complexity will be the same as with a chain topology. To prevent this situation, we minimize timing slack of connection cells when making connections. Connection cells and open fanin ports are sorted according to their arrival times. Then, a connection is tried first between a connection cell with large arrival time and an open fanin port with small arrival time (Lines 22, 24). The connection cells do not change

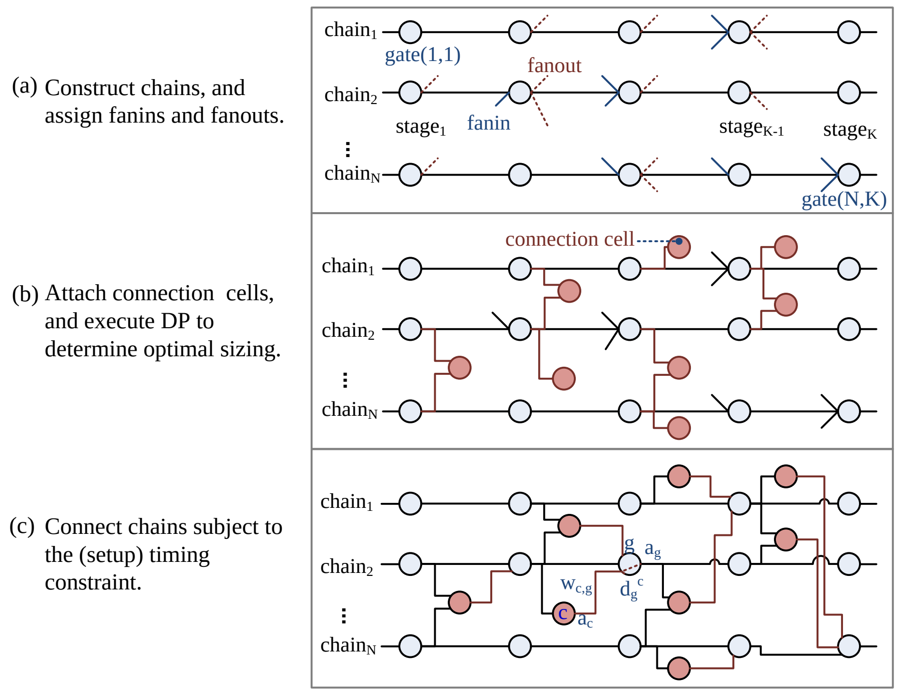

(a) Construct chains, and assign fanins and fanouts.
(b) Attach connection cells, and execute DP to determine optimal sizing.
(c) Connect chains subject to the (setup) timing constraint.
chain1
chain2
chainN
gate(1,1)
fanout
fanin
stage1
stageK-1
stageK
gate(N,K)
connection cell
g
ag
wc,g
dgc
c
ac

Figure 3.16: Netlist generation flow.

the optimal chain solution since they have minimum gate size. If we upsize them, there is no benefit to the timing slack of the main chain, and the optimal gate sizing of the chain does not change. Without timing constraints, our algorithm guarantees complete connection between open fanins and fanouts by virtue of Equation (3.11). With timing constraints, some ports can remain unconnected, which we address in Section 3.3.4 below. The open input ports are assigned with logic high (Vdd) or low (Vss) according to the logic type of the cell. This assignment does not change the optimal solution.

After completing all the connections, we end up with a benchmark circuit of $K \times N + C$ cells with known optimal gate sizing for minimum leakage. (A small detail: when we use the generated circuit as a sizing benchmark, we initially assign maximum cell size (with highest leakage and fastest timing) to each instance, so as to avoid giving the leakage optimization tool any information about the optimal solution.)

3.3.3 Experimental Setup

Our netlist generator is implemented in C++ and produces a benchmark netlist in *Verilog HDL* ($.v$) with the corresponding delay models ($.lib$). Two types of delay and power models are used from the previous eyechart work [99][11] – (i) LP: linear increase in power with size for gate sizing context, and (ii) EP: exponential increase in power with size for V_t or gate-length bias. The LP and EP power models have eight and three gate sizes (i.e., cell variants per master), respectively. We also use the delay models ($.lib$) from the ISPD-2012 Gate Sizing Contest [184] to generate realistic benchmark circuits. To analyze the problem complexity of generated netlists, and suboptimality of standard sizing tools, we perform experiments on a 2.8*GHz* Linux workstation with 64*GB* RAM, using six different gate sizing methods – (i) two commercial gate sizing and leakage optimization tools (*BlazeMO v2013* [1] and *Cadence Encounter v11.1* [6]);[12] (ii) two publicly available academic sizers (a web-available *UCLA sizing tool* [9] (*Greedy*) which greedily swaps cells according to a $\Delta power/\Delta delay$ sensitivity function, and the UCSD sensitivity-based leakage optimizer [35] (SensOpt) with $\Delta power \times slack$ sensitivity function);[13] and (iii) two gate sizing programs under the ISPD-2012 contest infrastructure (Trident [111] and LPRS [87]). To generate realistic benchmark circuits, we use eight ISPD-2012 testcases. In our experiments, we measure the suboptimality of the various gate sizing heuristics, defined as

$$Suboptimality = \frac{power_{heur} - power_{opt}}{power_{opt}} \tag{3.13}$$

3.3.4 Generated Benchmarks

We now present observations regarding generated benchmarks and their difficulty in power (leakage) optimization. We furthermore compare the benchmarks and real designs in terms of characteristic parameters. Figure 3.17 shows the schematic of a generated netlist with 10 chains and path depth of 20. In the netlist, chains are connected to each other in arbitrary order, and various topologies can be found.

A connection between chains can be made when the newly generated path has positive (or zero) slack with respect to the timing constraint. As a result, some cells in the chain will have open ports and some connection cells will remain unconnected. If the number of unconnected cells is large, the generated netlist will deviate from the specified fanin and fanout distributions. As noted above, to improve the connectability we can assign the larger fanins to later stages,

[11] According to the authors of [99], EP corresponds to the multi-V_t context, and LP corresponds to the gate-length biasing context.

[12] These are referred to as *Comm1* and *Comm2* below. We do not give the mapping – i.e., which tool is *Comm1* and which is *Comm2* – in order to maintain anonymity as required by the tools' licenses.

[13] Details of the UCSD SensOpt tool are available at the website [35]. The tool performs post-layout cell swapping using the *Tcl* socket interface to a golden STA tool, *Synopsys PrimeTime*.

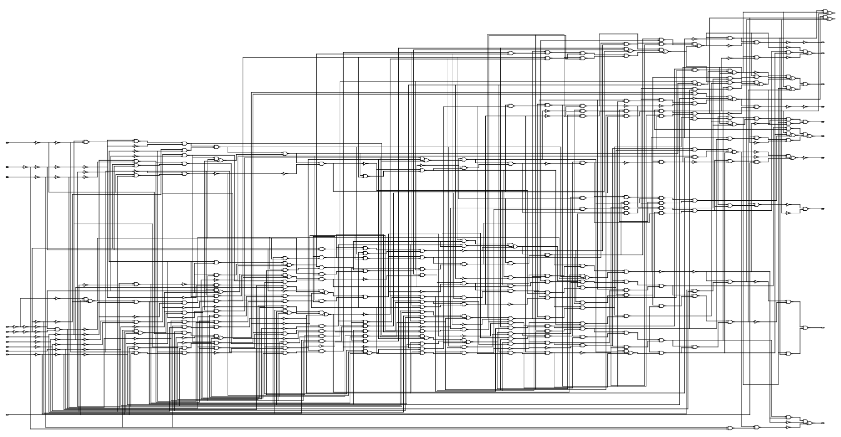


Figure 3.17: Schematic of generated netlist (N = 10, K = 20).

and larger fanouts to earlier stages. However, such an *arranged assignment* can reduce the difficulty of the sizing optimization for the benchmark: many connection cells will have loosely constrained timing (i.e., large slack), and this makes it easy to find the optimal solution. To consider both connectability and optimization difficulty, we mix the two alternative strategies – arranged and random assignments – in Algorithm 5, Lines 5 and 6. Table 3.9 shows the failure rate of connections among chains and problem complexity (suboptimality) according to the different mixtures of the arranged and random assignments. In the experiment, N and K values are fixed (40), and the EP power model is used. Suboptimality and runtime are obtained for the commercial tool (*Comm1*). The results show that 25% arranged assignments in practice results in over 99% connectivity, while also affording a sufficient problem complexity (11.2% suboptimality). The 100% random assignment shows smaller suboptimality (7.7%) for gate sizing because it results in many unconnected gates (17%). In all experiments reported below, we use the mix of 75% random and 25% arranged assignments.

Since our benchmark generator makes chains first, then connects the chains to each other, we have assessed the problem complexity of benchmarks before and after the chain connection. Table 3.10 shows the suboptimality of leakage reduction for the commercial tool and the greedy method. The results show that the complexity (difficulty) of gate sizing increases with the number of chain connections. The chain-only structures are easy to solve, and heuristics show small suboptimalities ($\sim$3%). However, with added chain connections, the observed suboptimality (and inferred instance difficulty) increases significantly.

Table 3.11 shows the characteristic parameters of ISPD-2012 testcases for the benchmark generation. The real circuits do not follow the second constraint of our netlist generator

Table 3.9: Connectability and complexity (suboptimality) of generated netlists according to different proportions of arranged and random assignments.

arranged	random	unconnected	subopt.	runtime
100%	0%	0.00%	2.6%	108*sec*
75%	25%	0.00%	6.8%	97*sec*
50%	50%	0.25%	10.3%	120*sec*
25%	75%	0.75%	11.2%	225*sec*
0%	100%	17.0%	7.7%	311*sec*

Table 3.10: Instance complexities and tool suboptimalities (in percent) for chain-only and connected-chain topologies.

# of	# of	chain-only		connected	
chains	stages	*Comm1*	*Greedy*	*Comm1*	*Greedy*
(a) EP library					
40	20	2.4%	0.3%	10.4%	8.7%
40	40	2.1%	1.3%	10.3%	11.1%
80	20	2.0%	0.5%	10.3%	10.9%
80	40	2.1%	1.3%	9.9%	10.9%
(b) LP library					
40	20	1.7 %	3.1%	7.7%	17.9%
40	40	2.4 %	3.5%	12.0%	18.5%
80	20	1.9 %	3.3%	12.3%	19.1%
80	40	2.5 %	3.5%	15.9%	19.6%

(Equation (3.11)) since the numbers of primary inputs and primary outputs are different. For this reason, we select fanin and fanout distribution numbers that are only similar (not identical) to those of the real designs when we perform the benchmark generation. From the results, generated circuits show similar design size, path depth and average fanin (fanout); this offers hope that our benchmark generation approach can provide realistic benchmark circuits for gate sizing.

3.3.5 Suboptimality of Heuristics

Figure 3.18 (respectively, Figure 3.19) shows suboptimality and runtime of heuristics (including *Comm1*) when the number of chains (respectively, number of stages) increases in the

Table 3.11: Characteristic parameters of real designs and generated benchmarks.

testcase	# of cells	# of depth	clock cycle (fast/slow)	fanin distribution				fanout distribution					
				1	2	3	4	1	2	3	4	5	6
dma	25K	29	770/900	0.222	0.595	0.082	0.099	0.777	0.059	0.072	0.036	0.008	0.006
pci_b32	33K	30	660/720	0.309	0.621	0.037	0.030	0.847	0.045	0.064	0.013	0.004	0.002
des_perf	111K	29	735/900	0.353	0.493	0.112	0.040	0.625	0.213	0.073	0.038	0.015	0.011
vga_lcd	165K	26	610/700	0.292	0.607	0.034	0.065	0.850	0.031	0.078	0.004	0.001	0.001

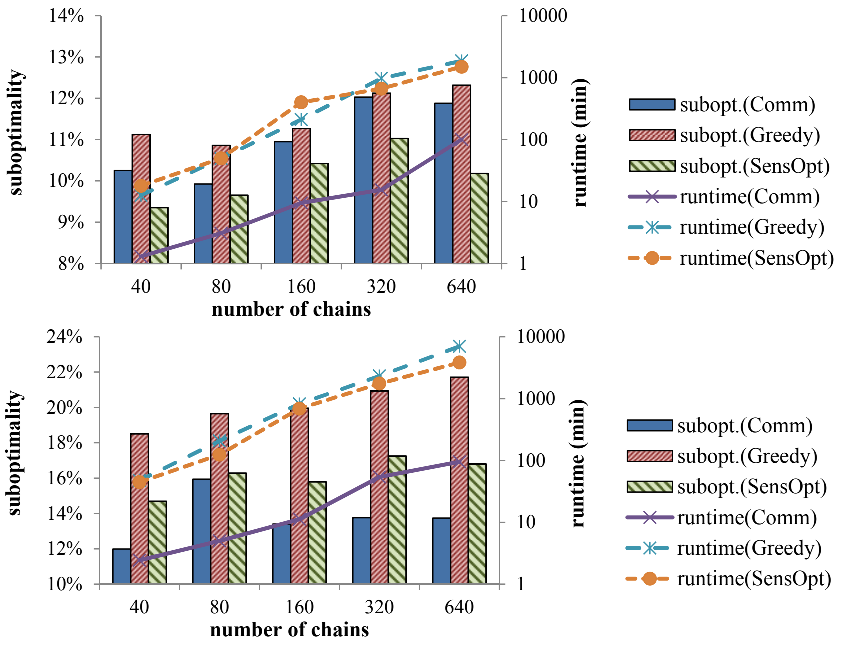


Figure 3.18: Suboptimality and runtime for different numbers of chains N (number of stages K = 40) with EP and LP libraries.

benchmark circuits.[14] From Figure 3.18, we see that the suboptimality increases slightly according to the design size, while runtime increases exponentially with the number of chains since the total number of paths increases rapidly with chain number. When the number of stages increases (Figure 3.19), the suboptimality increases especially for the *Comm1* and SensOpt solvers. We recall that the LP library model has a larger number (eight) of sizing candidates than the EP library model (three). We believe that this is a contributing factor behind the greedy and sensitivity-based optimizations showing larger suboptimality and runtime with LP library-based instances.

Figure 3.20 shows suboptimality and runtime results when the testcases have different

[14]The same fanin and fanout distributions have been used for the experiments in Figure 3.18 and Figure 3.19 – fid: 0.3, 0.6, 0.1, fod: 0.6, 0.1, 0.2, 0.1 .

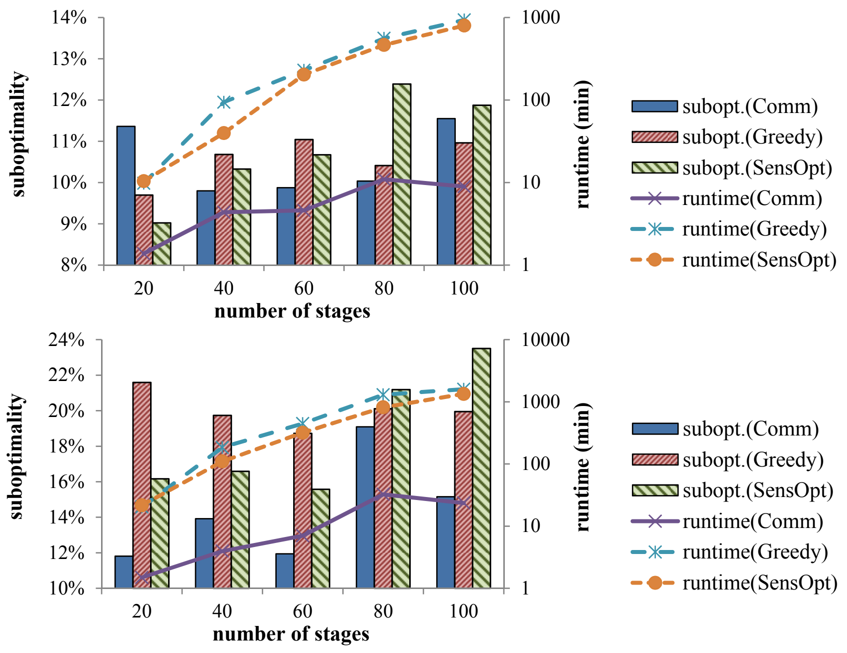


Figure 3.19: Suboptimality and runtime for different numbers of stages K (number of chains N = 100) with EP and LP libraries.

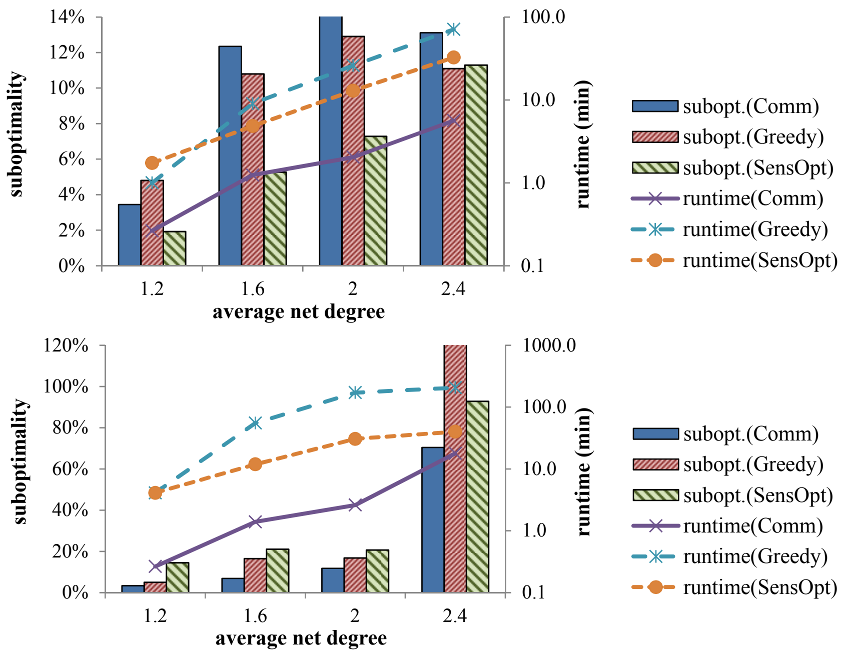


Figure 3.20: Suboptimality and runtime for different average net degrees (number of chains N = 40, number of stages K = 40) with EP and LP libraries.

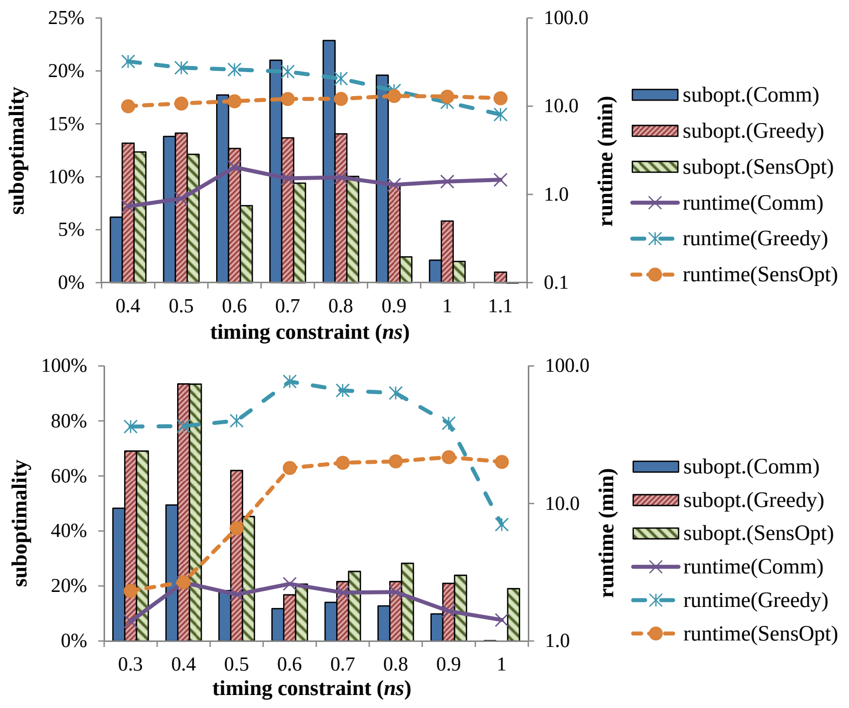


Figure 3.21: Suboptimality and runtime for different timing constraints (number of chains N = 40, number of stages K = 40, average net degree = 2.0) with EP and LP libraries.

topological complexities. To estimate the effect of netlist complexity, we change fanin and fanout distributions in the benchmark generation, such that each benchmark has a different average net degree. From the results, we see that suboptimality and runtime increase significantly according to the design complexity. With average net degree of 2.4 and the LP library, large suboptimality ($> 70\%$) is apparent for each heuristic.

In addition, we study the effect of delay constraints on suboptimality and complexity. Figure 3.21 shows suboptimality and runtime results when the testcases are generated with different delay constraints. The testcases have the same topology (number of chains, number of stages and net degree). However, the suboptimalities achieved by each heuristic differ widely according to the timing constraint. From the results, netlists with tight delay constraint lead to greater heuristic suboptimality, especially with the LP library.

Table 3.12 shows suboptimality and runtime results for the generated netlists in Table 3.11. The results show that prevailing sizing methods are suboptimal for realistic benchmark circuits by up to 76.7%, 22.0% and 271.0% for Trident [111], LPRS [87] and the commercial tools, respectively. Among the testcases, *ng_des_perf_fast* shows the largest suboptimality; we believe that this is a consequence of having a tighter timing constraint than the other testcases.

Table 3.12: Suboptimality with respect to known optimal solution for generated netlists.

testcases	optimal	Trident		LPRS [87]		*Comm1*		*Comm2*	
	leakage (W)	subopt.	runtime	subopt.	runtime	subopt.	runtime	subopt.	runtime
ng_dma_fast	0.2795	77.20%	121	18.20%	121	111.60%	73	206.30%	24
ng_dma_slow	0.1326	23.90%	109	8.80%	125	47.40%	104	104.50%	30
ng_pci_b32_fast	0.5115	60.60%	116	10.30%	118	73.30%	95	163.90%	35
ng_pci_b32_slow	0.2631	81.00%	114	9.30%	129	77.00%	87	199.20%	41
ng_des_perf_fast	0.8770	76.70%	574	22.00%	970	152.00%	456	271.10%	108
ng_des_perf_slow	0.4329	14.80%	917	8.50%	767	28.90%	497	96.60%	154
ng_vga_lcd_fast	1.9468	74.40%	543	11.10%	1125	75.80%	862	218.10%	161
ng_vga_lcd_slow	0.8647	44.50%	523	7.00%	1078	22.00%	779	105.10%	219

When we compare the results over actual ISPD-2012 testcases (Trident and LPRS results from Table 3.3), the artificial and real netlists suggest different relative and absolute suboptimalities for the sizing heuristics. We believe that this is for several reasons, notably (i) our enhanced eyechart-like benchmarks can be more challenging for a given heuristic since they have connection cells, and (ii) we use the same path depth for each chain, while real circuit will have different depths for the various critical paths. We continue to explore ways to improve the matching to results on real designs and real libraries, while maintaining the important property of having a known optimal sizing solution.

3.4 Conclusions and Future Directions

Thirty years of research on gate sizing have generated a large number of interesting ideas, but leave unclear how to architect and develop a leading-edge gate sizing tool. Significant improvements made by recent industry tools [186] suggest that newer, more powerful optimization methods could find even better power-performance tradeoffs in practice. This possibility has been confirmed at the ISPD-2012 Gate Sizing Contest, organized by researchers from Intel [184], where none of the contestants dominated on the entire benchmark set. Each team excelled on a small subset of benchmarks, and the best results on some benchmarks have been produced by teams not in the top three. These data reflect the importance and complexities of the (discrete) gate sizing problem, as well as the amount of room for further improvement, despite significant recent progress.

The most sophisticated published techniques for gate sizing are analytical in nature and assume continuous sizing and/or V_t assignment, and sometimes implicitly assume convexity in their optimization approach. These techniques can be frustrated by the combinatorial nature of discrete sizing and by the nonconvexity of circuit delay caused by side capacitance and tabular delay lookups. Combinatorial techniques can also be found in the literature, but either do not scale to large circuits (e.g., branch-and-bound) or remain limited to greedy optimization, which can become stuck in local optima due to the nonconvexity of delay. Significant progress has been recently achieved using dynamic programming, e.g., by some ISPD-2012 contestants [186]. However, these techniques may be less efficient on circuits with significant reconvergence, and may require long runtimes for large circuits or libraries.

In this chapter, we observe that gate sizing retains some aspects of convexity *in the global sense*, and that carefully prioritized greedy optimization can bring significant improvement. To this end, we develop several insights into (i) the sensitivity functions that lead to effective prioritization of gate upsizing and (ii) how these functions can be implemented efficiently. The best configurations of sensitivity functions apparently depend on circuit structure (depth, width, number of paths, etc.). Therefore, we parameterize the space of sensitivity functions, and develop metaheuristics that traverse this space by independently invoking lower-level heuristics at individual points. This optimization leverages *sequential importance sampling* from statistical physics [94], in the form of the *go-with-the-winners (GWTW)* metaheuristic that was previously analyzed in [39] and shown to perform on par with simulated annealing. To make this approach practical, we develop high-performance implementations of individual heuristics.

Empirical results on ISPD-2012 benchmarks, following the ISPD-2012 Gate Sizing Contest protocol, show that our implementation outperforms the best results recorded at the contest for all but one benchmark. Our implementation outperforms each individual contestant by a large margin, but by no means does it give the last word on the subject. Rather, many opportunities opened by our research remain unexplored, and we foresee that empirical performance can be improved further. Such improvements, as well as the ones reported in our work, will significantly enhance the power-performance tradeoffs in future generations of integrated circuits.

Our development of a high-performance gate sizing optimization has brought to light several major challenges, including: major challenges.

- identify interconnect delay models whose accuracy-vs.-complexity tradeoffs are compatible with large-scale optimization;
- develop an internal timer fast enough for move-based optimization, yet accurate enough

to track a signoff timer;

- satisfy timing, slew and capacitance constraints;
- obtain sharp tradeoffs between timing and power reduction; and
- use parallel computing resources effectively.

In solving these challenges, we have put significant effort into not only individual techniques and components, but also into the entire system, paying attention to the stability and scalability of optimization. Our software achieves highly competitive results compared to the ISPD-2013 contest winners. In particular, it outperforms the winners according to the secondary metric and places between first and second according to the primary metric. Given how recently the ISPD-2013 contest concluded, it is likely that our ongoing research will yield further improvements in the near future.

In addition to the gate sizing heuristics, we have proposed a new benchmark generation technique for gate sizing which constructs *realistic* circuits with known optimal solutions. Our generated netlists closely resemble real designs in terms of instance count, path depth, interconnect complexity, and net degree/fanin/fanout distributions; all of these attributes are parameters of the netlist generation. When we compare our generated benchmarks with real designs, we also see similarities with respect to other circuit characteristics such as average net degree and Rent parameter.

Our benchmarks with known optimal solutions enable systematic and quantitative study of the suboptimality of common sizing heuristics, with respect to key parameters of the circuit topology. In particular, our experimental results with web-available academic tools and commercial tools show that prevailing leakage-driven sizing methods are suboptimal for realistic benchmark circuits by up to 76.7%, 22.0% and 271.0% for Trident [111], LPRS [87] and the commercial tools, respectively. At the same time, our results also show discrepancies between inferences obtained using our generated circuits and those obtained using real circuits. However, all of our results suggest that commercial tools may still suffer from significant suboptimality, and/or that existing methods have "similar" degrees of suboptimality.[15] Our ongoing work seeks to address the above-mentioned discrepancies. In addition, we are working to handle more realistic delay models, possibly in the context of realistic benchmarks with tight upper bounds on the optimal gate leakage.

[15]This being said, we would like to make it clear that our study is not intended to imply or make any value judgment whatsoever regarding any commercial tools; certainly, it has not been our intent to perform any 'benchmarking' through our study.

3.5 Acknowledgments

Chapter 3 is in part a reprint of “Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012 and “High-Performance Gate Sizing with a Signoff Timer”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013.

I would like to thank my coauthors Dr. Jin Hu, Professor Andrew B. Kahng, Dr. Myung-Chul Kim, Hyein Lee, Professor Igor L. Markov and Pankit Thapar.

I also thank Professor Marcelo Johann and Mr. Tiago Reimann for helps on the benchmark experiments.

Chapter 4

Active-Mode Leakage Reduction with Data-Retained Power Gating

The use of clock gating and power gating to reduce dynamic power and static leakage power, respectively, is well-understood by both researchers and IC designers [17]. *Clock gating* is considered to be one of the most effective techniques to reduce dynamic power, and its automatic application is supported by EDA tools [45]. Clock gating masks the clock signal when the corresponding circuits are not performing useful computations. *Power gating* [213] drastically reduces leakage power by introducing a switch between the voltage supply (and/or ground) and a given block of functional circuitry; the block's leakage is stopped when the switch cuts off the current path from supply to ground.

To reduce active-mode leakage power, several approaches have been reported which combine clock gating and power gating [229] [49] [169] [160] [207]. However, these previous approaches have associated design complexity and overhead issues which limit their practical implementation.

In this chapter, we propose a new circuit-level technique which enables power gating of flip-flops during active mode. We combine both clock gating and power gating, such that flip-flops are power gated during clock masked periods. Our key contributions are as follows.

- The proposed technique enables concurrent clock and power gating, and thus achieves significant leakage power reduction during active mode.
- We introduce a *data retention switch* which sustains the voltage level of virtual ground to retain data in flip-flops.

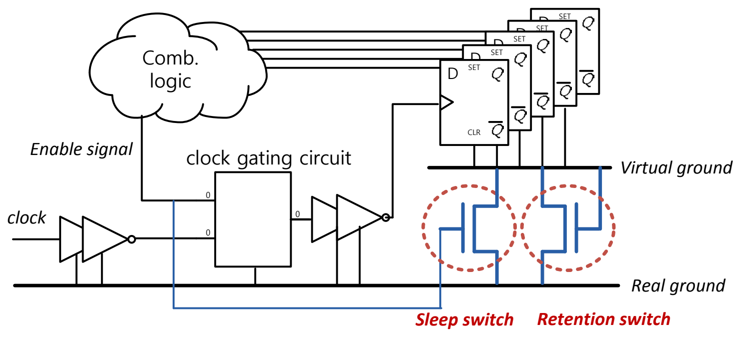


Figure 4.1: Proposed circuits to combine clock gating and power gating.

- We provide empirical confirmation of the leakage power reduction achieved by the proposed technique over conventional power gating approaches.

4.1 Data-Retained Power Gating

4.1.1 Integrated Clock and Power Gating

Most commercial logic synthesis tools [7] [26] support automatic insertion of clock gating logic without any modification of RTL codes. The inserted clock gating logic has clock gating control and enable signals. Clock signals are transparent during enable periods, and masked during disable periods. During a given masked period, the state of clock-gated flip-flops stays unchanged. As a consequence of recent product architectures as well as commercial synthesis tools' capabilities, flip-flops are masked for most of a given IC's operating time [145]. This offers an immediate motivation: If we could apply a power gating scheme to flip-flops during this masked period, then we could reduce active-mode leakage power. However, active-mode power gating requires that internal data state be retained, and according to existing practice, this requires huge overheads on both operation (e.g., data control to save and restore) and circuit design (e.g., retention flip-flops).

We introduce a new switch circuit to combine clock gating and power gating as shown in Figure 4.1. In the figure, the switch consists of two transistors; one is a normal sleep switch and the other is a retention switch. When the clock gating is disabled, the sleep switch is off. However, the retention switch induces a threshold voltage drop between virtual ground and real ground. This voltage drop reduces the operating voltage of flip-flops as well as leakage current. At the same time, the flip-flops can retain state with the reduced voltage.

The idea of a retention switch has been previously proposed by Kim et al. [142]. However, their technique requires additional layout area to implement N-well for the PMOS transistor. The PMOS can be replaced with an NMOS transistor by connecting the source and gate terminals to virtual ground. With such an approach, although the virtual ground may rise up to $V_{n,t}$ (NMOS threshold voltage), the flip-flops can retain state with reduced leakage.

Figure 4.2 shows HSPICE simulation results for data-retained power gating of a DFQ flip-flop in TSMC 65GP technology. Figure 4.2(a) shows the voltage of virtual ground according to the clock enable signal (en). Figure 4.2(b) shows the current (on Vss) of the flip-flop for both the DRPG and conventional (no power gating) cases. During the clock- and power-enabled period ($en = 1$), both cases show the same leakage power consumption. During power-gated (clock-disabled) periods ($en = 0$), the proposed retention switch sustains the voltage of virtual ground as $0.25V$, which achieves significant leakage savings (35%). The internal status of logic can be retained since $0.75V$ of supply voltage is sufficient to retain the flip-flop data [53].

In the conventional power gating, the voltage of virtual ground goes to supply voltage upon wake-up, which causes a large inrush current. However, in our approach, inrush current ((x) in Figure 4.2) is small due to the suspended virtual ground voltage, and the inrush current overhead is compensated during the idle state; charge stored at the virtual ground rail can be used to supply leakage power ((y) in Figure 4.2).

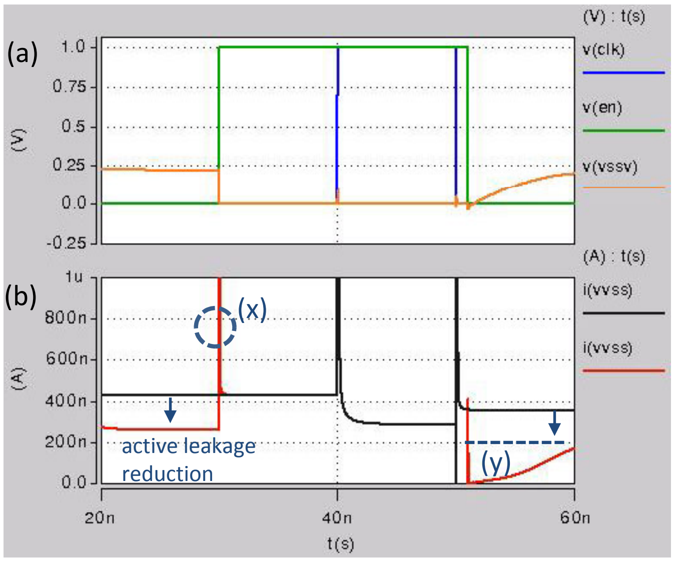


Figure 4.2: HSPICE results for DFQ (TSMC 65GP) cell. (a) Gated-clock (clk), clock enable (en) signals and virtual ground voltage (vssv). (b) Current plot on Vss (black: without power gating, red: DRPG).

4.1.2 Flip-Flop Implementations

The suspended virtual ground affects the output value of the flip-flop during power gating. Non-zero output value causes significant leakage overhead on the flip-flop's fanout cells. To solve this problem, we add a level-shifter circuit into the flip-flop. Figure 4.3 shows a schematic of the proposed flip-flop circuit. We add $P0$, $N0$ and $N1$ switches into the conventional flip-flop circuit to adjust the voltage level of output port (Q).

A conventional level shifter which is placed at the output of the flip-flop has significant delay and area overhead. For example, HSPICE-measured delay overhead can be over $400ps$ in a $65nm$ LP process at worst corner; such a delay impact cannot be ignored. We observe that a conventional level shifter changes operating voltage level between two different operating voltages, while the proposed circuit changes ground level from $V_{n,th}$ to $0V$. In light of this requirement, the level shifter circuit can be located at the input rather than at the output of the final buffer. This reduces the delay overhead, and also allows use of minimum transistor size in the implementation.

When the gate voltage of the $Pinv$ transistor is $V_{n,th}$, $Pinv$ turns ON and Q will be Vdd. Hence, the $P0$ and $N0$ transistors turn OFF and $N1$ is completely ON. Finally, the gate voltage level of $Ninv$ transistor goes to $0V$. On the other hand, when the gate voltage of the $Pinv$ transistor is Vdd, then the gate voltage of $P0$ is low and $P0$ completely turns ON. Hence, the gate voltage of $N0$ will go high and the $Ninv$ transistor turns ON. Finally, the output voltage of the final inverter is $0V$ and $N1$ transistors will turn OFF. The transistor ratio of $P0||N0$ and $N1$ should have a large value to minimize delay overhead. We have empirically determined transistor sizes based on HSPICE simulation results. Since $N1$ is only used to achieve $0V$ for the gate voltage of $Ninv$, its transistor width is minimum ($120nm$). Widths of $P0$ and $N0$ are $400nm$ and $200nm$, respectively, in the TSMC 65GP process.

With the additional devices, the flip-flop has a delay overhead, which we examine in detail in Section 4.2.2 below.

4.1.3 Physical Implementation

During standard-cell placement, flip-flops driven by the same clock gating logic are placed within a bounded region. In other words, since they are tightly coupled to each other and have the same clock behavior, commercial place-and-route (P&R) tools place them closely together. In addition, the clock gating logic is placed near its related flip-flop cluster – e.g., in the center of the cluster. Thus, a sleep control signal (enable signal of clock gating logic)

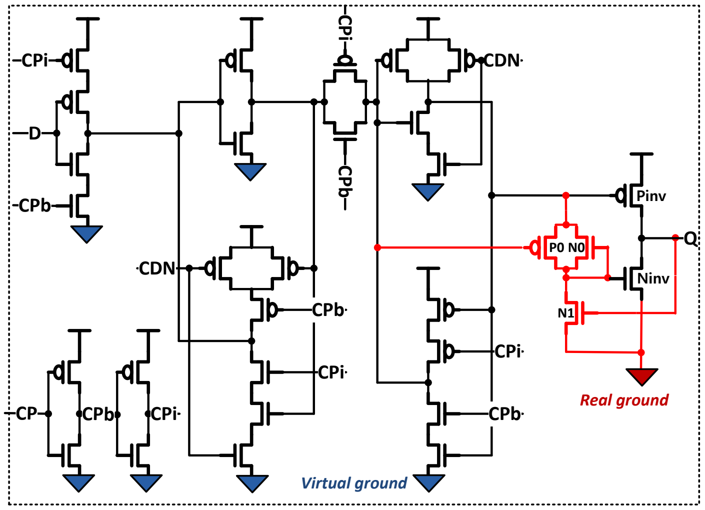


Figure 4.3: Flip-flop implementation with a level shifter. We add $P0$, $N0$ and $N1$ switches to adjust the voltage level of output port (Q).

requires just one or two buffers to control the sleep switch transistors, and can immediately turn on the sleep switches. To guarantee the correct operation of DRPG, flip-flops should be woken up before the arrival time of the clock signal that comes from clock gating logic. The feasibility of DRPG is validated in Section 4.2.2.

Our data-retained power gating can be implemented with global power gating (data is not retained) as shown in Figures 4.4(a) and (b) for the header switch and footer switch cases. For the footer switch case, additional *AND* gates are required. *PGEN* is a global power gating enable signal and *CKEN* is a clock enable signal. When DRPG is combined with global power gating, flip-flops will have three modes – (i) active mode ($PGEN = 1$ & $CKEN = 1$), (ii) retention mode ($PGEN = 1$ & $CKEN = 0$) and (iii) standby mode ($PGEN = 0$).[16]

Some modern design methodologies use multi-bit flip-flop cells, which can reduce physical design overhead since each can be treated as a single standard cell. This is also amenable to data-retained power gating by including sleep and retention switch inside as shown in Figure 4.4(c). A global power gating switch is not included in the standard cell implementation, and can be connected as shown in Figure 4.4(a). Figure 4.5 shows the physical layout of a four-bit DRPG flip-flop. In this layout, four DRPG flip-flops share a single sleep switch, which is controlled by a clock enable signal.

[16]In standby mode, current paths from supply to ground are cut off with conventional power gating. In this chapter, we do not address advantages and overheads of conventional power gating techniques, since they have been extensively studied in previous works (e.g., [213]).

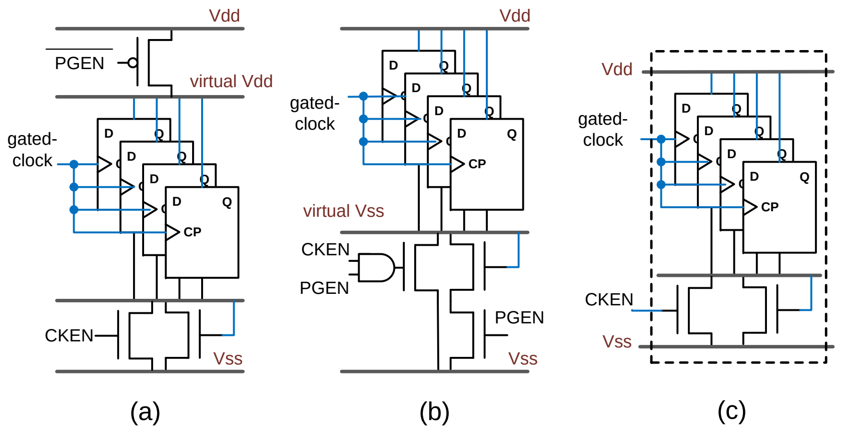


Figure 4.4: Implementation examples with DRPG – (a) global power gating with header switches, (b) global power gating with footer switches, and (c) standard cell implementation for a multi-bit flip-flop. [*PGEN*: global power gating enable; *CKEN*: clock enable signal.]

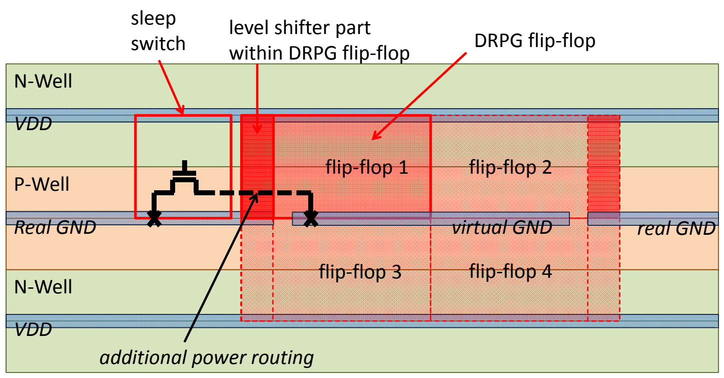


Figure 4.5: Physical layout of a four-bit DRPG flip-flop.

4.2 Experimental Results

To analyze leakage power, cell delay, and functionality of the proposed power gating, we perform circuit-level and design-level experiments. We implement our data-retained flip-flop with high V_t (HVT), normal V_t (NVT) and low V_t (LVT), and gate-length biasing, and evaluate delay and leakage power consumption of the implemented flip-flops (Section 4.2.2). We compare our data-retained flip-flop and a conventional retention flip-flop when they are used within a DRPG context (Section 4.2.3). Finally, with design-level implementations, we provide empirical confirmation of the leakage reduction afforded by DRPG (Section 4.2.4).

Table 4.1: Delay, leakage and area results of proposed data-retained flip-flops.

flip-flops		delay (ns)		delay overhead		single flip-flop			multi(8)-bit flip-flop		
V_t	cell-type	rising	falling	rising	falling	leakage (uW)	leakage reduction	area overhead	leakage (uW)	leakage reduction	area overhead
HVT	SDFQ	0.172	0.173	9.5%	19.3%	0.134	33.8%	15.0%	0.735	53.5%	8.0%
	SDFCNQ	0.190	0.177	4.3%	18.6%	0.140	36.4%	15.2%	0.750	56.5%	8.1%
	SDFSNQ	0.189	0.178	3.7%	19.5%	0.141	35.9%	17.6%	0.756	56.1%	9.4%
NVT	SDFQ	0.142	0.143	12.5%	20.7%	0.377	32.8%	15.0%	2.587	41.7%	8.0%
	SDFCNQ	0.168	0.146	14.3%	20.1%	0.416	34.5%	15.2%	2.823	43.9%	8.1%
	SDFSNQ	0.148	0.152	11.9%	20.1%	0.422	33.8%	17.6%	2.863	43.3%	9.4%
LVT	SDFQ	0.130	0.124	15.0%	19.1%	0.910	38.8%	15.0%	5.383	45.7%	8.0%
	SDFCNQ	0.155	0.127	16.9%	18.7%	0.982	42.0%	15.2%	5.737	49.6%	8.1%
	SDFSNQ	0.134	0.132	14.3%	18.2%	1.008	40.7%	17.6%	5.947	48.2%	9.4%

4.2.1 Experimental Setup

For the circuit-level experiments, we implement SPICE netlists of the proposed flip-flops (Figure 4.3) using TSMC 65GP SPICE models. To measure the cell delay and leakage power of implemented circuits, we use *Synopsys HSPICE vE-2010.12* [27]. For the design-level experiments, we use 11 open-source designs from the *OpenCores* site [22]. We use a TSMC 65GP cell library for the design implementation, and timing library models (*Synopsys Liberty*) for our data-retained flip-flops are prepared using *Cadence Library Characterizer v9.1* [3]. We synthesize the designs using *Synopsys DesignCompiler vF-2011.09* [26] and perform place-and-route with *Cadence Encounter Digital Implementation System v9.1* [6]. During synthesis, we use the clock gating optimization of DesignCompiler, which inserts clock gating cells automatically. We execute leakage optimization in DesignCompiler to replace clock-gated flip-flops with our data-retained flip-flops. After the placement and routing, we perform a post-layout leakage optimization with *UCSD SensOpt*.

4.2.2 Circuit-Level Implementations

We implement three types of flip-flops: *SDFQ* (D flip-flop with scan input), *SDFCNQ* (D flip-flop with scan and asynchronous reset signal), and *SDFSNQ* (D flip-flop with scan and asynchronous set signal). We also implement HVT (high V_t), NVT (normal V_t) and LVT (low V_t) versions for each flip-flop. We perform SPICE simulations for the implemented flip-flops with sleep and retention switches as shown in Figure 4.1. Table 4.1 shows $clock$-to-Q delay, cell

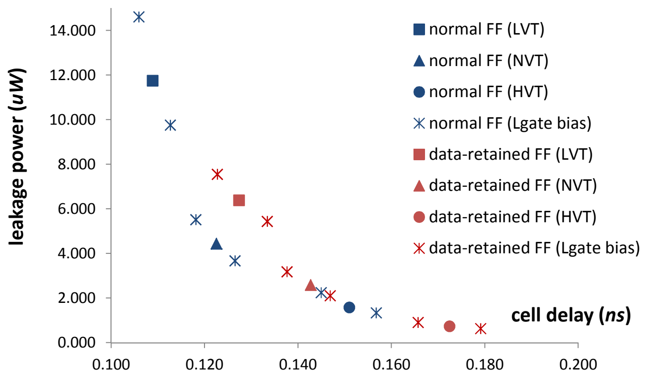


Figure 4.6: Delay and leakage power comparison for normal flip-flops and data-retained flip-flops (multi(8)-bit SDFQ flip-flop).

leakage and area information of the implemented flip-flops. Delay overheads, leakage reductions and area overheads are compared with those of the conventional versions of the flip-flops. The area overheads include the sleep and retention switches as well as the level-shifter circuit.

We measure the data for both the single flip-flop case and the multi(8)-bit case in which eight flip-flops share a sleep and retention switch together. We consider the multi(8)-bit case specifically since most data processing modules treat byte-based data. From the results, our proposed flip-flops can reduce active-mode leakage power by 36.5% with 15.9% area overhead on average with the data-retained power gating. When eight flip-flops are implemented in the same cluster (or multi-bit flip-flop is assumed), we can achieve further leakage reduction with smaller area overhead by sharing the sleep and retention switches. The area overhead has been measured with the sleep and retention switches as well as the shifter circuits, and compared with the area of conventional flip-flops. The clustered (or multi-bit) flip-flops show 48.7% leakage reduction with 8.5% area overhead, on average. Due to the level-shifter circuit, the proposed flip-flops have an average of 15.4% delay overhead over the corresponding conventional flip-flops.

Figure 4.6 shows the delay and leakage comparison for normal flip-flops and data-retained flip-flops. From the results, our data-retained flip-flop (HVT type) clearly extends the available tradeoff, and it hence provides more usable options for cell sizing and swapping optimizations. We explore gate-length (L_{gate}) biasing cases of +2nm and -2nm for each NVT, HVT and LVT cell. The results show that data-retained flip-flops offer more leakage-delay choices even when L_{gate} biasing is available as well. (The LVT data-retained flip-flop will never be used since it has no leakage-delay benefit over the normal NVT flip-flop.)

For correct operation during clock-enable periods, the wake-up latency when coming out of power gating should be less than the delay of the gated clock signal. In Figure 4.7(a), the sum of EN-to-Q delay in the CG (clock gating) cell and clock tree synthesis (CTS) buffer delays is typically larger than $200ps$. Figure 4.7(b) shows the waveform of the clock enable signal and virtual ground voltage from SPICE simulation. From the waveform, the voltage of virtual ground goes to zero within $30ps$. This means that the wake-up time of DRPG is sufficiently fast for correct flip-flop operation. On wake-up, the measured inrush current is $40.2uA$ (peak), which is 45% of peak current in the normal power gating case. Power overhead from the inrush current is compensated as shown in Figure 4.2.

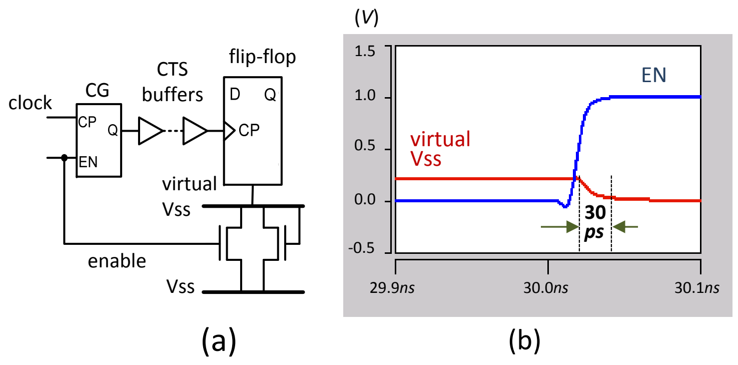


Figure 4.7: (a) Clock and enable signal connections for data-retained power gating; (b) waveforms of the clock enable signal and virtual ground voltage.

4.2.3 Comparison with Conventional Retention Flip-Flops

Conventional retention flip-flops retain data during power gating, and can also be used for DRPG. Figure 4.8 shows a schematic of the live-slave type of retention flip-flop, which provides power into a slave latch during power gating. If we replace the flip-flop in Figure 4.1 with the retention flip-flop, we do not need to use the retention switch. We can remove the clock-mask circuit (Figure 4.8(a)) since the clock is masked from the clock gating circuit. To preserve the proper voltage level at the output port, we should connect real (true) ground to the output inverter (Figure 4.8(b)).

We have implemented the live-slave type of retention flip-flop as shown in Figure 4.8, and used this flip-flop for DRPG. Figure 4.9 shows (a) virtual ground voltage and (b) current results for live-slave retention flip-flop (blue color) and retention switch (red color). From the results, the conventional retention flip-flop can achieve 25% active-mode leakage reduction without delay overhead compared with normal flip-flops. However, as discussed above in Section 4.1.1, the voltage of virtual ground goes to near high voltage (Vdd) during the power gating, and there is significant inrush current when the sleep (footer) switch is turned on. Because of the inrush current, the conventional retention flip-flop is not suitable for active-mode power gating.

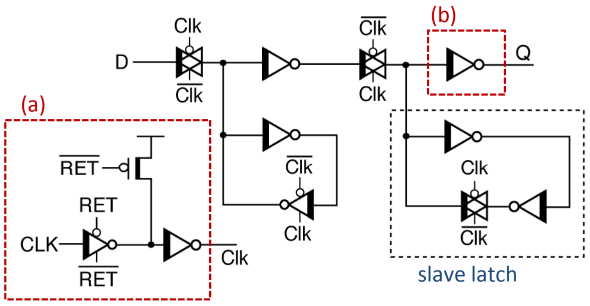


Figure 4.8: Live-slave retention flip-flop. To use the flip-flop for DRPG, (a) the clock-mask circuit can be removed, and (b) the output inverter should be connected to the real ground.

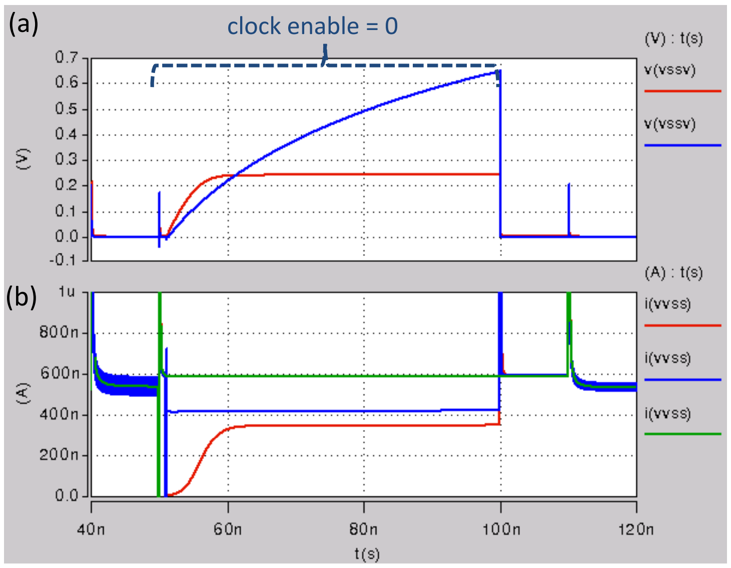


Figure 4.9: (a) Virtual ground voltage (vssv) and (b) leakage current for normal flip-flop (green color), live-slave retention flip-flop (blue color) and power gating with retention switch (red color).

4.2.4 Leakage Reduction for Implemented Designs

We implement 11 benchmark designs to assess the active-mode leakage reduction from our power gating approach. We use multi-V_t (HVT, NVT and LVT) standard library cells including data-retained flip-flops (N.B.: recall from Section 4.2.2 above that the LVT data-retained flip-flop is never instantiated). Three different timing constraints are used – (a) tight (maximum available frequency), (b) normal (20% longer clock period than the tight constraint), and (c) loose (50% longer clock period than the tight constraint). Figure 4.10 shows area breakdowns of combinational logic, non-clock-gated flip-flops, and clock-gated flip-flops for the implemented designs with the normal timing constraint. From the results, the portion of clock-gated flip-flops varies according to the designs. Some designs (e.g., *AES* and *CONMAX*) do not permit significant clock gating. However, we can see that most of the designs can use clock gating logic extensively.

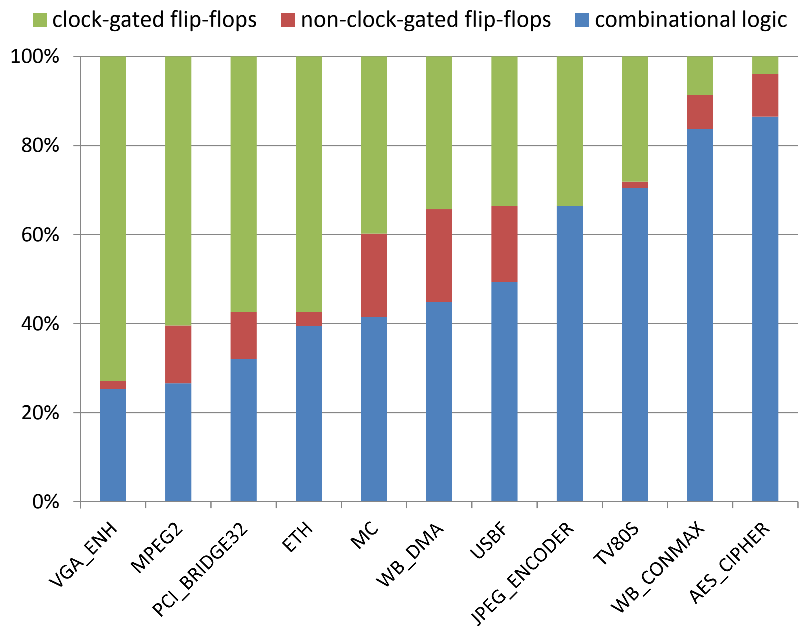


Figure 4.10: Breakdown of area for implemented designs (clock-gated flip-flops, non-clock-gated flip-flops and combinational logic).

We have applied our power gating technique to the implemented designs. Table 4.2 shows the implemented results and leakage power reduction over the conventional designs, which do not power gate during active mode. The amount of leakage reduction depends on (i) the portion of clock-gated flip-flops as shown in Figure 4.10 and (ii) the timing constraints, which are given in nanoseconds in the table. Designs with smaller proportions of clock-gated

Table 4.2: Leakage reduction achieved by data-retained flip-flops on benchmark designs [CG-FF: clock-gated flip-flops].

design	clock	# of CG-FFs		leakage power (*W*)		leakage reduction	
	period (ns)	normal	DRPG	flip-flops	total	flip-flops	total
AES	1.15	2	154	3.50E-05	4.55E-04	2.9%	6.5%
	1.38	0	156	3.32E-05	2.94E-04	6.4%	2.9%
	1.73	0	156	3.10E-05	1.94E-04	5.5%	-1.0%
ETH	1.15	140	9936	2.97E-04	1.51E-03	44.0%	2.4%
	1.38	64	10012	2.71E-04	7.05E-04	45.8%	12.3%
	1.73	21	10055	2.61E-04	4.05E-04	47.5%	23.9%
JPEG	1.25	133	4192	2.08E-04	1.38E-03	54.9%	13.1%
	1.50	69	4257	1.63E-04	8.59E-04	38.7%	7.5%
	1.88	17	4309	1.26E-04	5.81E-04	43.4%	10.2%
MC	1.25	18	705	4.39E-05	1.10E-04	24.2%	8.1%
	1.50	7	716	3.74E-05	7.32E-05	23.2%	11.3%
	1.88	3	720	3.02E-05	5.47E-05	49.4%	17.5%
MPEG2	0.95	165	2351	1.12E-04	2.43E-04	31.6%	12.9%
	1.14	90	2425	9.86E-05	1.87E-04	38.0%	12.3 %
	1.43	48	2468	8.12E-05	1.23E-04	43.5%	23.5 %
PCI	0.95	292	2480	1.32E-04	2.85E-04	33.0%	13.0%
	1.14	166	2606	1.17E-04	2.19E-04	30.1%	11.9%
	1.43	67	2705	9.29E-05	1.45E-04	35.4%	16.4%
TV80S	1.30	35	307	2.60E-05	1.16E-04	41.6%	9.1%
	1.56	28	314	2.32E-05	7.27E-05	35.4%	9.8%
	1.95	15	327	1.55E-05	4.78E-05	52.4%	15.0%
USBF	0.95	137	1011	8.49E-05	1.89E-04	25.8%	12.1%
	1.14	70	1078	6.21E-05	1.29E-04	27.3%	12.0%
	1.43	24	1132	5.06E-05	9.23E-05	29.4%	15.7%
VGA	1.30	0	16643	4.14E-04	1.27E-03	48.1%	11.1%
	1.56	1	16642	4.09E-04	6.96E-04	48.7%	21.8%
	1.95	4	16639	4.08E-04	5.71E-04	49.1%	25.7%
CONMAX	1.20	146	238	5.21E-05	2.94E-04	-5.9%	1.7%
	1.44	83	301	2.80E-05	1.76E-04	16.9%	5.0%
	1.80	1	383	2.54E-05	1.22E-04	15.7%	4.3%
DMA	0.75	81	236	3.72E-05	7.04E-05	16.0%	5.7%
	0.90	50	267	2.39E-05	4.23E-05	32.5%	17.9%
	1.13	9	308	1.63E-05	2.79E-05	39.5%	17.6%

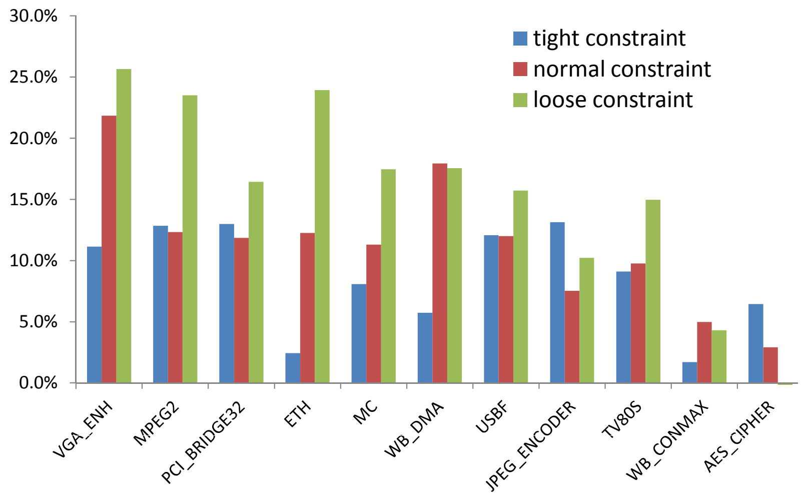


Figure 4.11: Leakage reduction for different timing constraints – tight constraint (maximum available frequency), normal constraint (tight constraint + 20% clock period) and loose constraint (tight constraint + 50% clock period).

flip-flops (e.g., *AES* and *CONMAX*) show small (or no) leakage reduction from our DRPG technique. As shown in Table 4.1, the proposed data-retained flip-flop has delay overhead. Therefore, we cannot replace normal flip-flops with the data-retained flip-flops if the timing slack is less than the delay overhead; we only exploit available slack, and do not permit performance (timing) degradation, i.e., DRPG is not applied to flip-flops in timing-critical paths. Table 4.2 shows that more data-retained flip-flops are used with looser timing constraints. With tight timing constraints, more flip-flops are in timing-critical paths, and hence fewer flip-flops can be replaced with the data-retained flip-flops. Moreover, with tight timing constraints, the leakage contribution of combinational cells increases more than that of flip-flops, since buffer insertion and gate sizing are mainly performed on the combinational cells. As a result, timing constraint effects on achievable leakage reduction vary across testcases, as shown in Figure 4.11. We estimate area overheads of the DRPG implementation based on Figure 4.5, and consider additional areas for DRPG flip-flops and sleep switches in this estimation. As shown in Table 4.2, our DRPG technique incurs 3.09% area overhead on average.

From the results, we conclude that our DRPG technique can reduce leakage power over conventional designs by up to 13.1% (average 8.7%), 21.8% (average 11.3%) and 25.7% (average 15.3%) with tight, normal and loose timing constraints, respectively. We note that these leakage reductions are for digital portions only. We expect that larger design cases will show similar leakage reductions as in our current experimental results.

4.3 Conclusions and Future Directions

In this chapter, we propose a new circuit-level technique which enables power gating of clock-gated flip-flops during active mode. We combine clock gating and power gating techniques together, such that the flip-flops are power gated during clock masked periods. We introduce a retention switch which retains data during the power gating. With the retention switch, correct logic states and functionalities are guaranteed without additional overheads. With small area and performance overheads, our proposed technique can achieve significant dynamic leakage reduction over conventional designs. Using $65nm$ libraries and 11 open-source designs, we demonstrate that the proposed power gating technique can achieve maximum and average leakage savings of 25.7% and 11.8% over conventional designs.

4.4 Acknowledgments

Chapter 4 is in part a reprint of "Active-Mode Leakage Reduction with Data-Retained Power Gating", *Proc. Design Automation and Test in Europe*, 2013.

I would like to thank my coauthors Professor Andrew B. Kahng and Dr. Bongil Park.

Chapter 5

Exploiting Error Resilience in Low-Power Design

Conventional hardware is designed and optimized using techniques that aim to ensure correct operation of the hardware under different conditions. Conservative design techniques seek to ensure correct hardware operation under worst-case conditions. Better-than-worst-case design techniques [41] save power by eliminating guardbands, but are still aimed at ensuring correct hardware operation under nominal conditions.

Our research has asked the question: *Should the availability of an error-resilience mechanism change the way we approach hardware design and optimization?* I.e., given that mechanisms exist to tolerate hardware errors, should hardware continue to be designed for correct operation or should it be optimized for a target error rate even during nominal operation? To address this question, we propose and evaluate a novel approach to hardware design, called *recovery-driven design*. Rather than optimizing for correct operation, a recovery-driven design deliberately allows timing errors [108] [83] to occur during nominal operation, while relying on an error-resilience mechanism to tolerate these errors. In other words, a recovery-driven design optimizes a circuit for a non-zero target error rate that can be gainfully tolerated by hardware- [83] or software-based [108] error resilience. The motivating expectation behind recovery-driven design is that the "underdesigned" hardware will have significantly lower power or higher performance than hardware optimized for correct operation. Also, because errors are now allowed, the design methodology can exploit workload-specific information (e.g., activity of timing paths, architecture-level criticality of timing errors, etc.) to further maximize the power/performance benefits of underdesign.

In the first part of this chapter, we show that optimizing power for a target timing error rate for voltage overscaling-induced errors indeed results in significant power savings for similar levels of performance. We show that this is true when errors are detected and corrected by a hardware error-tolerance mechanism [83] or allowed to propagate to an error-tolerant application [56] where the errors manifest themselves as reduced performance or output quality [108]. Increasing the target error rate for a processor module increases the potential for power savings, since the module can be operated at a lower voltage. In practice, the target error rate is chosen such that an error recovery mechanism can correct the resulting errors and still reduce energy (after considering the error recovery overhead) for an acceptable degradation in performance or output quality. The power benefits of exploiting error resilience are maximized by redistributing timing slack from infrequently-exercised paths that cause very few errors to frequently-exercised paths that have the potential to cause many errors. This reduces the error rate at a given voltage, and hence reduces the minimum supply voltage and power for a target error rate.

We present a detailed evaluation and analysis of a slack redistribution-based recovery-driven design methodology that minimizes the power of a processor module for a target error rate. Our cell sizing-based, design-level methodology has been used to create recovery-driven processors that are optimized for different target error rates or error-resilience mechanisms. Since some error-resilience mechanisms (e.g., error-tolerant applications) require adaptation to multiple reliability targets, we have also extended our recovery-driven design approach to create *gradual slack* designs – designs that are optimized not for a single error rate, but for a range of error rates. Such gradual slack designs (or *soft processors*) have the ability to trade performance or output quality for energy savings over a range of reliability targets.

The second part of this chapter focuses on the cost of error tolerance in the chip implementation context. Error-tolerant design can reduce design constraints by allowing timing errors, and obtain power and area benefits over conventional designs which always operate correctly. However, the error-tolerant design requires additional circuits to detect and correct timing errors. Figure 5.1 shows the structure of three error-tolerant register designs: (a) Razor flip-flop [83], (b) Razor-Lite flip-flop [143], and (c) TIMBER flip-flop [69]. The Razor flip-flop has an additional shadow latch and other error-tolerant circuits (comparator, multiplexer and *OR* gate). When compared to a conventional flip-flop, the total power overhead of the Razor flip-flop is 30% [77]. The overhead of additional circuits will diminish the benefit of error-tolerant design, and we must ensure that the benefit (in terms of area and power reduction from the error resilience) outweighs the additional cost of error-tolerant registers. With this in mind, the second part of this chapter develops a design flow that optimizes the mix of resilient and non-resilient circuits within a given implementation, so as to minimize the overhead of error resilience.

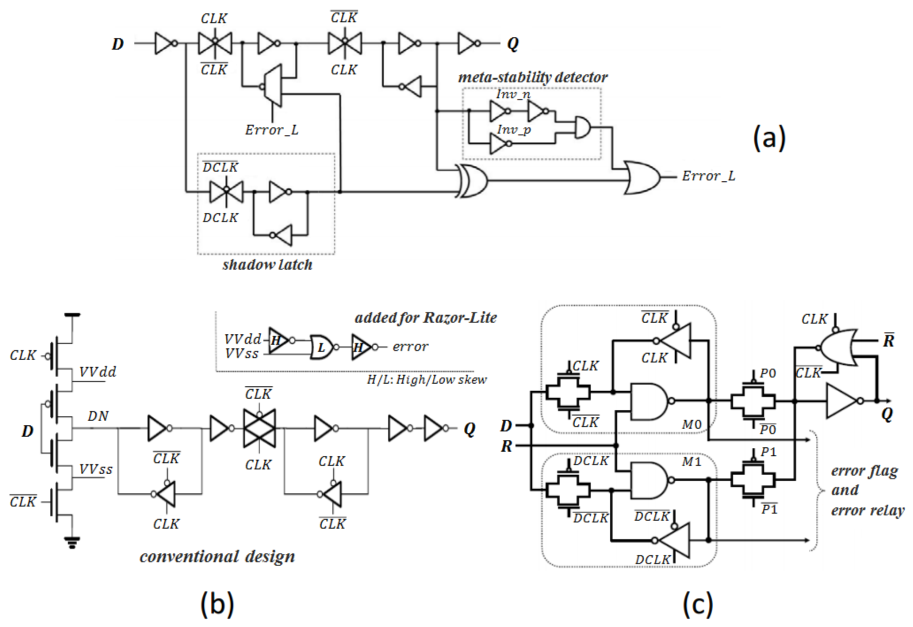


Figure 5.1: Structure of error-tolerant registersr: (a) Razor flip-flop [83], (b) Razor-Lite flip-flop [143], and (c) TIMBER flip-flop [69].

We make the following contributions in this chapter.

- To the best of our knowledge, we present the first design flow for power minimization that deliberately allows errors under nominal conditions. We demonstrate that such a design flow can result in power savings of 11.8%, on average, over all modules and error rate targets, and up to 29.1% for individual modules.

- We explore the heuristic choices and tradeoffs that are fundamental to the optimization quality of slack redistribution-based, recovery-driven designs. Within the context of the heuristic presented in Section 5.1.3, we evaluate choices for path priority and traversal during optimization, optimization radius, accuracy of path selection, error budget utilization, starting netlist, voltage step-size granularity, and iterative optimization in terms of their effects on the optimization result, heuristic runtime, and sensitivity to target error rate.

- To support the proposed recovery-driven design flow, we present a fast and accurate technique for post-layout activity and error rate estimation. We use collected functional information to redistribute slack efficiently in a circuit and significantly extend the range of voltage scaling for a target error rate.

- We extend our recovery-driven design methodology to create *recovery-driven processors* (processors that are optimized for different target error rates or error recovery mechanisms) and *soft processors* (processors that are optimized for efficiency over a range of target error rates). We demonstrate the power and energy benefits of such processor designs.

- We demonstrate that the power benefits of recovery-driven processors and soft processors increase when a hardware- or software-based error-resilience mechanism is used. We consider Razor [83] and application-level noise tolerance [234] as examples and show additional energy reductions of 19% and 20% with respect to the best correctness-optimized processors that exploit the same error-resilience mechanisms.

- We propose a selective-endpoint optimization in error-tolerant design that reduces timing-critical endpoints with small cost of timing optimization. With the optimization, we can replace the error-tolerant registers and minimize the overhead of error resilience.

- We propose a clock skew optimization which adds a guardband to normal registers without changing data path circuits. This optimization can improve reliability and robustness over process, voltage and temperature variations.

- We also analyze the effectiveness of our proposed optimizations across a range of error-resilience scenarios having different overheads and safety margins associated with the error-tolerant register.

5.1 Recovery-Driven Design

In this section, we propose a *recovery-driven design* approach that optimizes a processor module for a given target timing error rate instead of correct operation. We show that significant power benefits are possible from a recovery-driven design flow that deliberately allows errors caused by voltage overscaling to occur during nominal operation, while relying on an error recovery technique to tolerate these errors. We present a detailed evaluation and analysis of such a CAD methodology that minimizes the power of a processor module for a given target error rate.

5.1.1 Motivation

The goal of recovery-driven design in the context of voltage overscaling can be stated formally as follows. Given an initial netlist N_0, a set of cell libraries characterized for allowable operating voltages, toggle rates for the toggled paths in the netlist, and a target error rate ER_{target}, produce the optimized netlist $N_{V_{opt}}$ and operating voltage V_{opt} that minimize the total power consumption $W_{V_{opt}}$ of the circuit, such that the error rate of the optimized netlist does not exceed ER_{target}. Figure 5.2 demonstrates the goal. Our recovery-driven design optimization redistributes slack from infrequently-exercised paths to frequently-exercised paths and performs cell downsizing for average-case conditions. These optimizations reduce the power consumption of a circuit and extend the range that voltage can be scaled before a target error rate is exceeded. The combination of these factors produces a design with significantly reduced power consumption.

In the following, we present a cell sizing-based design methodology that relies on efficient redistribution of timing slack from infrequently-exercised critical paths to frequently-exercised paths to reduce the error rate at a given voltage, allowing a reduction in voltage for a given target error rate.

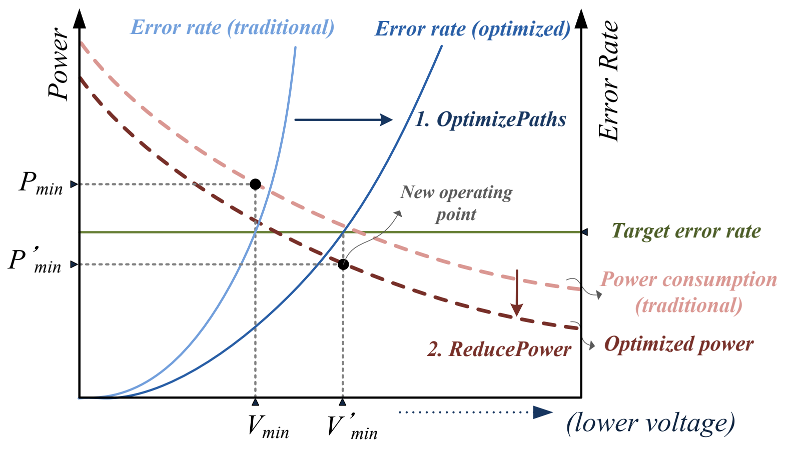


Figure 5.2: The goal of recovery-driven design.

5.1.2 An Abstract Heuristic for Power Minimization

Our heuristic for slack redistribution-based power minimization uses a two-pronged approach – extended voltage scaling through cell upsizing on critical and frequently-exercised circuit paths (*OptimizePaths*), and leakage power reduction achieved by downsizing cells in non-

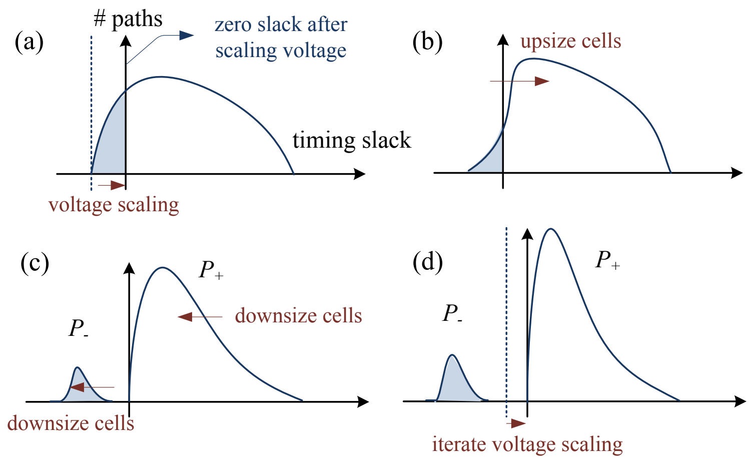


Figure 5.3: The slack distribution during the power minimization procedure.

critical and infrequently-exercised paths (*ReducePower*). The heuristic searches for the combination of the two techniques that results in the lowest total power consumption for the circuit, by performing path optimization and power reduction at each voltage step and then choosing the operating power at which minimum power is observed.

Figure 5.3 illustrates the evolution of the circuit path slack distribution throughout the stages of the power minimization procedure. Each iteration begins with voltage scaled down by one step (a). After partitioning the paths into sets containing positive- and negative-slack paths, *OptimizePaths* attempts to reduce the error rate by increasing timing slack on negative-slack paths (b). Next, the heuristic allocates the error rate budget by selecting paths to be added to the set of negative-slack paths, and downsizes cells to achieve area/power reduction (c). This cycle is repeated over the range of voltages to find the minimum power netlist and corresponding voltage (d). In Figure 5.3, P_+ is a set of paths that must have positive slack after power reduction, and P_- is a set of paths that are allowed to have negative slack. We ensure positive slack for P_+ paths by characterizing timing with worst-case libraries.

Figure 5.4 presents the algorithmic flow of our power minimization heuristic, which couples path optimization to extend the range of voltage scaling (*OptimizePaths*) with area minimization to achieve power reduction (*ReducePower*). In the figure, P_a is the set of all paths toggled during simulation. P_p is the set of all positive-slack paths. P_n is the set of all negative-slack paths in P_a. $\chi_{toggle}(p)$ is the set of cycles in which path p is toggled.

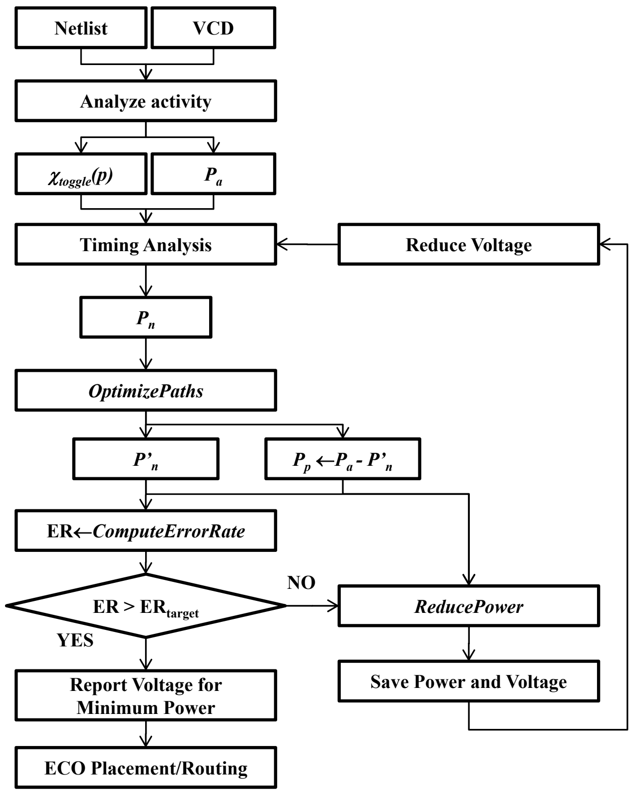


Figure 5.4: Algorithmic flow of a heuristic for minimizing power for a target error rate.

5.1.3 Heuristic Procedures

Path Optimization. The goal of the path optimization procedure ($OptimizePaths$) presented in Algorithm 6 is to minimize the error rate at a voltage level by transforming negative-slack paths into positive-slack paths. This is accomplished by performing cell swaps within the negative-slack paths to increase path slack. Negative-slack paths with maximum toggle rates are selected first during optimization, since they have the most potential to reduce the error rate if converted into positive-slack paths.

When a path is targeted for optimization, cell swaps are attempted on all cells in the path to increase slack as much as possible until positive path slack is achieved.[17] Once a cell has been visited during optimization, it is marked to prevent degradation of timing slack on any path containing the cell. Before accepting a cell swap, path slack is checked on all paths containing that cell or any visited fanin/fanout cell. If the swap has caused a decrease in slack for any such

[17]We consider only setup timing slack, since hold violations can typically be fixed by inserting hold buffers in a later step.

path, the move is rejected, and the original cell is restored. Previously optimized (visited) fanin and fanout cells are protected from slack decrease because they belong to paths that have higher toggle rates and, thus, higher priority of optimization. If cell swaps on a path fail to shift the path back into the set of positive-slack paths, then the path is ignored during subsequent iterations of path optimization.

Any cell swap that increases the error rate (by causing a path to switch from the set of positive-slack paths to the set of paths allowed to have negative slack) is rejected. Otherwise, we recompute the sensitivity of the swapped cell and all cells in its fanin / fanout network and select the next cell for downsizing.

Algorithm 6 Path optimization (*OptimizePaths*) procedure.

```
Procedure $OptimizePaths(P, N_{V_i}, V_i)$
Input : timing paths P, netlist $N_{V_i}$, operating voltage $V_i$
Output : optimized netlist $N_{V_i}$
1: Clear 'visited' mark in all cells in the netlist $N_{V_i}$;
2: while $P \neq \emptyset$ do
3:    Select path p from P with maximum toggle rate;
4:    for each cell $c$ in path $p$ do
5:       if $c.visited ==$ true then continue;
6:       $c.visited \leftarrow$ true;
7:       for each logically equivalent cell $m$ for the cell instance $c$ do
8:          Resize cell $c$ with logically equivalent cell $m$;
9:          $Q \leftarrow \{c\} \cup \{$visited fanin and fanout cells of $c\}$;
10:         for each path $q$ in $P$ that contains a cell in $Q$ do
11:            if $\Delta slack(q, c, m, V_i) < 0$ then restore cell change;
12:         end for
13:      end for
14:   end for
15:   $P \leftarrow P \setminus \{p\}$;
16: end while
```

Power Reduction. After path optimization, the error rate of the circuit is minimized at the present voltage. From this state, we proceed to minimize the power at the present voltage by utilizing the available error rate budget. Algorithm 7 (*ReducePower*) describes our power reduction procedure. The goal of the power reduction heuristic is to efficiently allocate the remaining error budget to infrequently-exercised paths in order to maximize power reduction achieved by cell downsizing. Typically, cells on P_{-} paths can exploit additional downsizing, because these paths are not bound by the normal timing constraints for the circuit.

The first step in power reduction is to choose additional paths to become negative-slack paths until the target error rate of the circuit is matched. Paths are selected in order to minimize the additional contribution to the error rate of the circuit. After defining the partition between

Algorithm 7 Power reduction (*ReducePower*) procedure.

```
Procedure $ReducePower(P_p, P_n, N_{V_i}, V_i, ER_{target})$
Input : positive-timing paths $P_p$, negative-timing paths $P_n$, netlist $N_{V_i}$, operating voltage $V_i$, target error rate $ER_{target}$
Output : optimized netlist $N_{V_i}$
1:  $P_+ \leftarrow P_p$ and $P_- \leftarrow P_n$;
2:  while $P_+ \neq \emptyset$ do
3:      Select path p from $P_+$ with minimum $\Delta ER(p)$;
4:      $ER \leftarrow ComputeErrorRate(P_-)$;
5:      if $ER \leq ER_{target}$ then
6:          $P_- \leftarrow P_- \cup \{p\}$;   $P_+ \leftarrow P_+ \cup \{p\}$;
7:      else
8:          break;
9:      end if
10: end while
11: Insert all downsizable cells into set $C$;
12: $ComputeSensitivity(C, N_{V_i}, V_i, -1)$;
13: while $C \neq \emptyset$ do
14:     Downsize cell $c$ from $C$ with minimum $sensitivity(c)$;
15:     $Q \leftarrow \{c\} \cup$ {fanin and fanout cells of $c$};
16:     for each path $p$ in $P_+$ that contains a cell in $Q$ do
17:         if $slack(p, V_i) < 0$ then
18:             Restore cell change;
19:             $C \leftarrow C \setminus \{c\}$;
20:             continue while loop;
21:         end if
22:     end for
23:     $ComputeSensitivity(Q, N_{V_i}, V_i, -1)$;
24:     if cell $c$ is not downsizable then
25:         $C \leftarrow C \setminus \{c\}$;
26:     end if
27: end while
```

negative- and positive-slack paths, cell downsizing is performed for all cells in the circuit in order of minimum sensitivity. We define the sensitivity of a cell in Equation (5.4) as the change in cell slack (Δs_c) divided by the change in cell power (Δw_c) when the cell c is downsized by one size. The slack of cell c is defined as the minimum slack on any timing arc containing c. The power of cell c is the sum of static power ($w_{stat}(c)$) and dynamic power ($w_{dyn}(c)$) for the cell. This formulation of sensitivity is similar to those proposed by previous works targeting leakage power reduction [100] [102].

$$sensitivity(c) = \frac{s_c - s_{c'}}{w_c - w_{c'}} \quad , where \;\; w_c = w_{stat}(c) + w_{dyn}(c) \tag{5.1}$$

5.1.4 Path Extraction and Error Rate Estimation

Path Extraction. Our heuristic has many path-based procedures – *OptimizePaths*, *ReducePower*, and *ComputeErrorRate* – and it is impractical to consider all of the topological paths in these procedures. Therefore, we reduce the number of paths that we consider by extracting only paths toggled during functional simulation. The value change dump (VCD) file can be used to extract toggled paths. To produce a VCD file, we perform gate-level simulation with *Cadence NC-Verilog* [5] at a frequency slow enough to capture all possible signal transitions. Figure 5.5 shows an example VCD file and the path extraction method. The VCD file contains a list of toggled nets at each time when a transition occurs, as well as their new values. We can use this information to extract toggled paths in each cycle. Glitched or toggled nets in each cycle are marked, and these nets are traversed to find toggled paths. We detect a toggled path when toggled nets compose a connected path of toggled cells from a primary input or flip-flop input to a primary output or flip-flop output. In Figure 5.5, nets a, x, and y have toggled in the first and third cycles (#1, #3), and nets b and y have toggled in the second and fourth cycles (#2, #4). We extract two paths: $a \longrightarrow x \longrightarrow y$ and $b \longrightarrow y$.

Toggle Rate and Error Rate Estimation. In order to accurately minimize power for a target error rate, we must be able to produce accurate estimates for error rate during our optimization flow. Thus, we propose a novel approach to error rate estimation that enables design for a target error rate.

We calculate the toggle rate of an extracted path using the number of cycles in which the path toggles. $\chi_{toggle}(p)$ represents the set of cycles in which path p has toggled during the simulation. $TR(p)$ represents the toggle rate of path p and is defined as

$$TR(p) = \frac{|\chi_{toggle}(p)|}{X_{tot}} \tag{5.2}$$

where $|\chi_{toggle}(p)|$ is the number of cycles in which path p has toggled, and X_{tot} is the total number of cycles in the simulation. Using the toggled cycle information of negative-slack paths, we can calculate the error rate precisely. The error rate (ER) of the design is calculated as

$$ER = \frac{|\bigcup_{p \in P_n} \chi_{toggle}(p)|}{X_{tot}} \tag{5.3}$$

where P_n is the set of negative-slack paths in the set of all toggled paths. In Figure 5.5, if paths $a \longrightarrow x \longrightarrow y$ and $b \longrightarrow y$ both have a toggle rate of 0.4 (number of toggled cycles is 2, and total number of cycles is 5), and if path $a \longrightarrow x \longrightarrow y$ has negative slack, then timing errors will occur in cycles #1 and #3. Therefore, the error rate is 0.4 for this example.

Our novel technique for error rate estimation has proven to be much faster than functional simulation and more accurate than previous estimation techniques. Results comparing our VCD-based technique to functional simulation and previous estimation approaches can be found in [128].

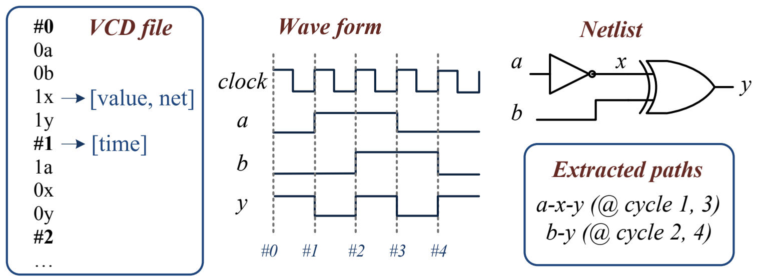


Figure 5.5: VCD file format and path extraction.

5.1.5 Heuristic Design Choices

In this section, we discuss heuristic design choices.

Experiment 1: Path Ordering During Optimization. The order in which we select paths for optimization affects the optimization result, since we prevent cells from being visited multiple times during optimization. The order also matters because we protect previously optimized paths from slack degradation due to other attempted cell swaps, as previously optimized paths have a higher optimization priority. We evaluate two prioritization functions for path selection during optimization. The first ranks paths in order of decreasing toggle rate $TR(p)$. Paths with the highest toggle rates have the greatest potential to decrease error rate when optimized. We compare against a function that ranks paths in order of decreasing value of $TR(p)/|slack(p)|$. In this alternative, we prefer paths with smaller negative slack, since the least effort is required to convert these paths into positive-slack paths.

Experiment 2: Optimization Radius. The goal of optimization is to maximize the slack of a targeted path through cell swaps. We evaluate two alternatives for the radius of optimization. In one case, we only swap cells on the target path. In the second case, we target both the cells on the path as well as cells in their fanin/fanout networks, since swaps in the fanin/fanout network can also affect cell slack.

Experiment 3: Path Traversal During Optimization. When optimizing a path, the order in which cells are visited can have an effect on the optimization result, since cell swaps affect input slew and output load. We consider two options – traversal from front to back, and traversal from back to front. We iterate over the cells in a path and make swaps until there is no further increase in the path slack.

Experiment 4: Accuracy of Path Selection During Power Reduction. During power reduction, positive-slack paths are selected to be added to the set of paths allowed to have negative slack, thus utilizing the available error rate budget. Paths are prioritized in order of increasing incremental contribution to error rate, $\Delta ER(p)$. However, after moving a path from P_+ to P_-, $\Delta ER(p)$ can change for paths that shared error cycles with the moved path.

To obtain precise ordering in terms of error rate contribution, we can update $\Delta ER(p)$ dynamically after each path selection. However, this introduces a runtime overhead, since we must continuously update $\Delta ER(p)$ for all remaining P_+ paths. We compare such *precise prioritization* against the alternative case where $\Delta ER(p)$ is calculated only once for all P_+ paths before path partitioning.

Experiment 5: Error Rate Budget Utilization. During power reduction, the final error rate after cell downsizing could be less than the target error rate, ER_{target}, since some paths in P_- might still have positive slack, even after maximum downsizing on the path cells. In this case, we might continue to reduce the power of the design by selecting more paths to add to P_- and downsizing cells again. We evaluate two cases – one where a single pass is performed for path selection and cell downsizing, and one where the *ReducePower* procedure is repeated until there is no further reduction in power (i.e., repeat *ReducePower* whenever some paths added to P_- still have positive slack after cell downsizing).

Experiment 6: Starting Netlist. Here, we evaluate heuristic performance for different starting netlists corresponding to loose (clock period increased by 10%) and tight (clock period reduced by 40%) timing constraints. This can significantly affect the final voltage reached, the dependence on ECO, and the amount of power savings afforded by the power minimization algorithm.

Experiment 7: Voltage Step Size. In each iteration of the power minimization heuristic, we step down the voltage by a value V_{step} and run the *OptimizePaths* and *ReducePower* procedures to produce a netlist for the present level of voltage scaling. The size of V_{step} can influence the optimization result and runtime of the heuristic. Thus, we compare two values of V_{step} – $0.01V$ and $0.05V$ – and compare the characteristics of the final netlist as well as the heuristic runtime.

Experiment 8: Iterative (Incremental) Optimization. In each iteration of the heuristic, we perform optimization of negative-slack paths at that voltage level. At the next iteration, we have a choice between starting from the latest optimized netlist ($N_{V_{i-1}}$) or the original netlist (N_0). We compare the results produced with each alternative to expose any systematic differences in power and runtime characteristics.

5.1.6 Gradual Slack Design

We extend our design methodology to implement another form of recovery-driven design called *gradual slack design* [129], which reshapes the slack distribution of a processor to create a gradual failure characteristic, rather than the typical critical wall. While error rate-optimized, recovery-driven designs achieve better energy efficiency at a single target error rate, gradual slack designs have the ability to trade reliability, throughput, or output quality for energy savings over a range of error rates. Figure 5.6 illustrates the optimization approach for gradual slack design. The goal of the 'gradual slope' slack optimization is to transform a slack distribution having a critical 'wall' into one with a more gradual failure characteristic. This allows performance–power tradeoffs over a range of error rates.

To achieve a gradual slack distribution with our recovery-driven design flow, we do not optimize for a single target error rate by selecting P_{-} paths. Instead, we select the maximum target error rate corresponding to the desired range of scalability, and optimize only the negative-slack paths in the scaling range with the highest switching activity, in order to maximize the range of voltage scalability for target range of error rates. We downsize only cells that have negligible activity so that the slack distribution for the active paths and the error rate of the processor are not affected. In this way, we maintain the desired gradual sloping slack distribution rather than creating a critical wall distribution with a cluster of active paths in the permanent negative-slack region.

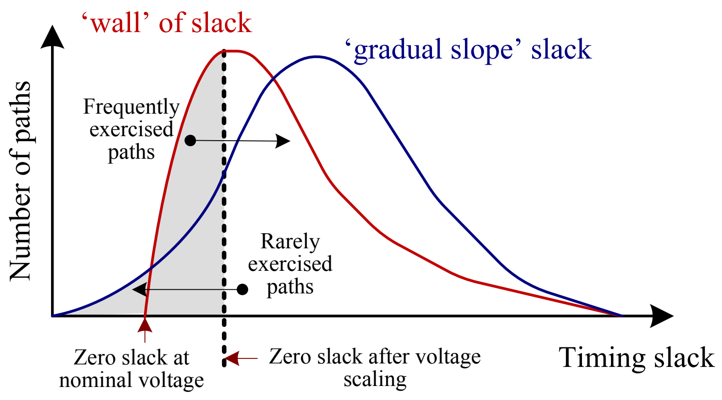

'wall' of slack
'gradual slope' slack
Number of paths
Frequently exercised paths
Rarely exercised paths
Zero slack at nominal voltage
Zero slack after voltage scaling
Timing slack

Figure 5.6: The 'gradual slope' slack optimization.

5.1.7 Processor Power Reduction

Algorithm 8 Processor-level design heuristic.

```
Procedure OptimizeProcessor ($ER_{target}$, MODULES, DOMAINS)
Input : target error rate $ER_{target}$, design modules, voltage domains
Output : optimized design modules
1: for each module $m$ in the optimization list of MODULES do
2:     for each error rate $ER < ER_{target}$ do
3:         $PowerOptimizer(N(m), ER)$;
4:     end for
5:     Use the results from $PowerOptimizer$ to characterize $P_m(V, ER)$;
6: end for
7: for each voltage $V \in V_{range}$ do
8:     Minimize $P_{core}(V) = \Sigma(P_m(V, ER))$ s.t. $ER_{core}(ER_{module_1}, ..., ER_{module_M}) \leq ER_{target}$;
9:     Record minimum power $P_{core}^{min}(V)$ and module error rate assignment
       $S(V) = [ER_{module_1}, ..., ER_{module_M}]$;
10: end for
11: Select the voltage $V_{opt}$ at which power $P_{core}^{min}$ is minimized;
12: Let $V^*(S(V)[m])$ be the voltage that minimizes power for module $m$ at $ER = S(V)[m]$;
13: Locate the $DOMAINS$ neighbors $\{V_1, ..., V_{DOMAINS}\}$ nearest to the set of voltages $V^*(S(V_{opt}))$;
14: Assign each module $m$ to the voltage domain $V_D[m] \in \{V_1, ..., V_{DOMAINS}\}$ that minimizes power
    $P_m(V_D[m], S(V_{opt})[m])$;
15: Layout the processor, selecting for each module $m \in$ MODULES
```

Algorithm 8 is our heuristic for minimizing the power of a processor core for a target error rate. The first step of the heuristic involves characterizing the modules of the processor core in terms of their power consumption at different error rate and voltage targets. These data are provided by $PowerOptimizer$ and are used to select the optimal operating voltage(s) for the processor core, as well as the error rate targets to assign to the processor modules.

The next step in the heuristic is to use the data from $PowerOptimizer$ to solve an optimization problem. The optimization objective is to minimize the power of the processor core subject to the constraint that the processor error rate must be less than the chosen target rate. Using the data from $PowerOptimizer$, we can formulate expressions for the power and error rate of the processor core in terms of the module error rates and the operating voltage. Thus, the goal of the optimization problem for a particular voltage is to find the assignment of error rate targets to modules that satisfies the optimization objective. We use a disjunctively-constrained knapsack-based [241] approach to solve the optimization problem. The knapsack solver selects the voltage and error rate assignment for which the power of the processor core is minimized and uses the selected error rate-optimized netlist of each module to lay out the processor.

For multiple voltage domain designs ($DOMAINS > 1$), the heuristic selects the voltage level of each domain, and the partitioning of modules to voltage domains, so as to minimize core power. This involves first selecting the error rate targets for the modules based on a minimum-power global assignment, then selecting the levels for the voltage domains and module-to-level assignments such that the power of the modules is minimized. The latter step is performed using a "Nearest-Neighbor" search to identify the neighbors nearest to the optimal module voltages that respectively correspond to the module error rate assignments in the space of voltages.

5.1.8 Recovery-Driven Processors

The proposed design methodology enables *recovery-driven processors* – processors that are optimized to deliberately produce timing errors at a rate that can be gainfully tolerated by an error recovery mechanism. Below, we describe two recovery-driven processor designs – one targeting hardware-based error resilience and another targeting software-based error resilience.

Case Study: Circuit-level Timing Speculation. Recall that a popular hardware-based scheme for error detection and correction is circuit-level timing speculation [83] [226]. Circuit-level timing speculation-based techniques detect errors by sampling the same computation twice – once using the regular clock and again using a delayed clock. If there is a mismatch, an error is signaled. Correction involves treating the delayed clock output as the correct output. Razor [83] and EDS [226] provide good examples of circuit-level timing speculation.

A recovery-driven processor design targeted for Razor takes into account the frequency of errors that can be gainfully tolerated by Razor (determined by the dynamic error recovery overhead) as well as the number of latches in which an error may occur (which determines the cost of making the circuit robust to errors). For the design-level heuristic, this means that when we define the partition between paths that are allowed have errors (P_-) and paths that are error-free (P_+), we must consider the error rate contribution of each path, which adds to the dynamic recovery overhead of Razor. We must also account for the cost of using a Razor flip-flop at the endpoint of any path that may potentially cause a timing error, and buffering for any short paths terminating at that endpoint. If downsizing a path during $ReducePower$ requires that we must replace a regular flip-flop with a Razor flip-flop, then we should ensure that the energy benefit (in terms of power reduction for additional cell downsizing) outweighs the additional cost of the Razor flip-flop and any short-path hold buffering. Since Razor assumes a maximum delay constraint on all paths [203], in addition to checking P_+ paths for negative slack (Line 16 of $ReducePower$) we must also ensure that all P_- paths respect the delay constraint after a downsizing move.

Case Study: Application Noise Tolerance. Error-tolerant applications [234] represent an opportunity to save power and increase performance by allowing errors to propagate to the application level rather than expending power to detect and correct them at the hardware level. For several such applications, data errors simply result in reduced output quality, instead of program failure. Designing a recovery-driven processor for error-tolerant applications requires several considerations. First, the set of processor modules is partitioned into two subsets – one containing modules that produce errors that the applications can tolerate and another containing modules that should not allow errors to propagate to the application level. For the class of error-tolerant applications that we consider in this work, errors in the arithmetic units can be tolerated. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

In addition to the list of modules to optimize, the $OptimizeProcessor$ procedure requires a target error rate. The error rate is chosen such that all applications in the class have acceptable quality for the target error rate. For the modules that produce errors that the application cannot tolerate, one of two approaches can be followed. One option is to operate those modules on the same voltage rail as the modules in which faults are allowed (single rail design). In this case, we feed these modules to the optimization heuristic targeting some hardware recovery mechanism that guarantees correctness, such as Razor. The two groups must agree on a common voltage that minimizes power consumption for the entire processor, and the optimal voltage reported by the optimization heuristic can be used as a constraint for the second optimization. Alternatively, the two groups can operate in separate voltage domains (dual rail design), in which case each optimization can select a different optimal voltage.

Soft processor design can also be used to adapt the reliability of the processor for reliability-diverse workloads, with more power savings available as the error rate target decreases. To create a soft processor design, the gradual slack module-level heuristic is used, and the optimal voltage and error rate targets of the modules are chosen based on the range of error rate targets that the processor should support.

5.1.9 Design-Level Methodology

Our methodology for demonstrating the benefits of recovery-driven design has two parts – a design-level methodology to characterize the power and reliability of circuit modules optimized for different voltage and error rate targets, and an architecture-level methodology to estimate processor power and performance when the proposed design-level techniques are applied at the processor-level.

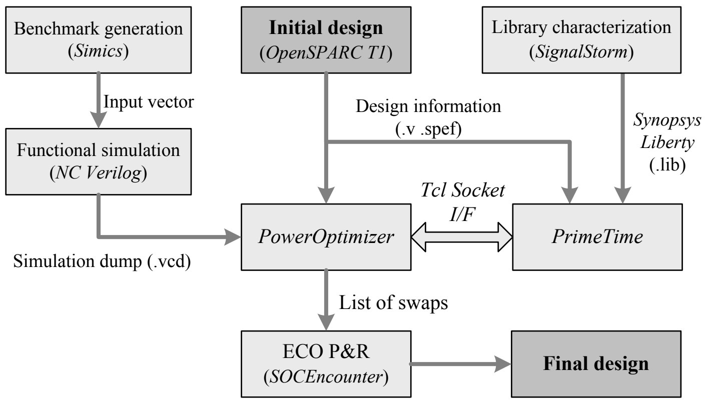


Figure 5.7: CAD flow incorporating the power optimization heuristic to minimize the power of a design for a given error-tolerance technique.

We use the *OpenSPARC T1* processor [25] to test our optimization framework. Table 5.1 describes the selected modules and provides characterization in terms of cell count and area. Module designs are implemented in TSMC 65GP technology using a standard flow of synthesis with *Synopsys Design Compiler vY-2006.06-SP5* [26] and place-and-route with *Cadence SoC Encounter v8.1* [6]. Runtime is reduced by adopting a restricted library of 66 commonly-used cells[18] (62 combinational and 4 sequential). Conventionally constrained designs are synthesized for the target operating frequency (0.8GHz), and tightly constrained designs are synthesized with a 40% smaller clock period (guardband) to have enough timing slack.

Figure 5.7 illustrates our recovery-driven design flow. We perform gate-level simulation to produce a VCD file[19] using *Cadence NC-Verilog v6.1* [5]. To find timing slack and power values at specific voltages, we prepare *Synopsys Liberty* (.lib) files for each voltage from 1.00V to 0.50V in 0.01V increments, using *Cadence Library Characterizer v9.1* [3]. Complete characterization for 51 voltage points takes a couple of days, but this is a one-time cost.

Timing information is continually available from *Synopsys PrimeTime c2009.06* [29] static timing tool through the *Tcl* socket interface [32], during the optimization process. After our optimization, all netlist changes are realized using *Cadence SoC Encounter* in ECO (engineering change order) mode.

Gate-level simulation is performed using test vectors obtained from full-system RTL simulation of a benchmark suite consisting of integer and floating point SPEC benchmarks [23]. These benchmarks are each fast-forwarded to their early SimPoints using the OpenSPARC T1

[18]Heuristic efficiency depends on the number of available logically equivalent cells. Since we use all available cell sizes for different drive strengths, our heuristic will also be effective with a full set of library cells.

[19]Gate-level simulation is performed for one million cycles, and the size of the VCD file is about 500MB for our testcases. To implement larger designs, a compressed VCD file could be used – e.g., *Synopsys VCD Plus* format [31].

system simulator, Simics Niagara [171]. After fast-forwarding in Simics, the architectural state is transferred to the *OpenSPARC* RTL using *CMU Transplant* [72].

Our recovery-driven design techniques optimize for average activity. To ensure that the activity profiles used during optimization (training) are representative and adequate, we use mutually exclusive training and test workloads. We optimize based on the average activity of half of our benchmarks and test using the other half. Training and test sets are chosen randomly and contain half integer and half floating point benchmarks. Table 5.2 shows the benchmarks in the training and test sets.

When characterizing Razor-based designs, we use worst-case timing libraries to determine any path that might have negative slack under worst-case PVT variations. We assign a Razor flip-flop to the endpoint of any such path, add a maximum delay constraint of 1.5 cycles to the path, and add a minimum delay constraint of 0.5 cycle to all paths ending at that flip-flop. We add buffers to any path that does not meet the minimum delay constraint. An error triggers a recovery period during which the pipeline recovers to a correct state. During this time, we assume that no progress is made, but we do account for the power and time consumed during recovery when reporting processor throughput and energy. We assume a counterflow pipeline-based Razor implementation [83] with a recovery penalty proportional to the depth of the pipeline (i.e., nine cycles for our nine-stage pipeline). We use the error rate, in conjunction with the rates of power consumption during normal operation and error recovery, as well as the recovery time overhead of Razor to calculate the energy overhead of error recovery [83]. Figure 5.8 compares the energy and area overheads of Razor for each design style that we evaluate. The fraction of Razor flip-flops ranges from 2.6% for a tightly constrained design to 5% for a Razor-optimized recovery-driven design.[20] Our Razor-optimized recovery-driven design heuristic directly accounts for the overheads of adding a Razor flip-flop to ensure increased energy savings, even if additional Razor flip-flops are required.

5.1.10 Architecture-Level Methodology

We use SMTSIM [228] integrated with Wattch [52] to simulate processors whose single core parameters are in Table 5.3. The simulator reports performance and power numbers at different voltages. Our evaluations are performed using the benchmarks listed in Table 5.2; these are chosen to maximize diversity in terms of performance and reliability requirements. We base our out-of-order processor microarchitecture model on the MIPS R10000 [242].

[20]In our previous work [129] [127], all flip-flops were Razor flip-flops, leading to different absolute power and area numbers.

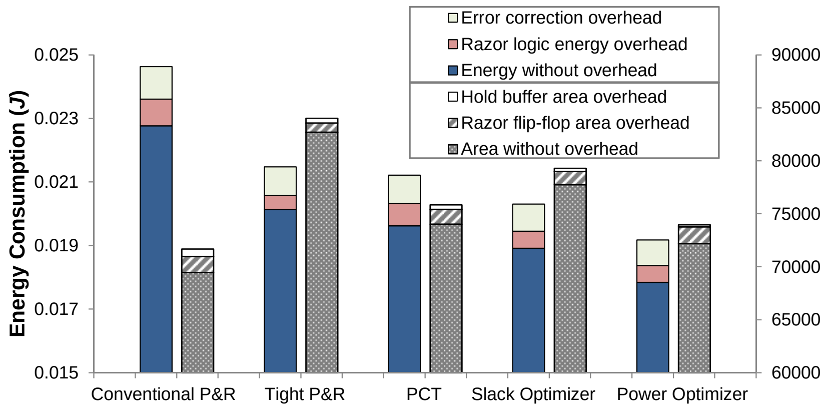


Figure 5.8: Energy and area overheads for Razor-based design.

Table 5.1: Target modules for experiments.

module	stage	description	# of cells	area (um^2)
lsu_dctl	MEM	L1 Dcache control	4537	13850
lsu_qctl1	MEM	LDST queue control	2485	7964
lsu_stb_ctl	MEM	ST buffer control	854	2453
sparc_exu_ecl	EX	execution unit control	2302	7089
sparc_ifu_dec	FD	instruction decode	802	1737
sparc_ifu_errdp	FD	error data path	4184	12972
sparc_ifu_fcl	FD	L1 Icache and PC control	2431	6457
spu_ctl	SPU	stream processing control	3341	9853
tlu_mmu_ctl	MEM	MMU control	1701	5113

To obtain a processor-wide error rate at a given frequency and voltage, we first sum the error rates from all the sampled OpenSPARC modules and then scale up the sum based on area, such that it includes all modules that we target for optimization. The error rate of a module that has not been characterized is assumed to be proportional to the module's area. We target only logic modules with our recovery-driven design methodology. On-chip memories are assumed to operate on a separate voltage rail [12] at the lowest error-free voltage for a given operating frequency. At 45nm and below, such "split rail" designs are common. While we provision for error-free SRAMs, logic that interfaces with SRAM structures, such as register read and writeback logic, may still produce errors. For designs that rely on error-tolerant applications, we scale the error rates of each module group separately, according to an error rate characterization of sampled modules in the group. Once the processor core-wide error rate is calculated, we can

Table 5.2: Benchmarks in the training and test sets.

benchmarks for design optimization (training set)	
art	image recognition and neural nets
bzip2	compression
mcf	combinatorial optimization
mesa	3D graphics library
benchmarks for design evaluation (test set)	
equake	seismic wave propagation
gzip	compression
twolf	place and route simulator
sort	sorting
additional benchmarks for processor-level evaluation	
AMMP, APPLU, MGRID, PARSER, SWIM, CRAFTY, EON, WUPWISE VPR, VORTEX-2, FACEDETECT†, CG†, LSQ† († *error-tolerant application*)	

Table 5.3: Processor specifications.

property	value	property	value
L1 cache	16*KB*, 4-way, one cycle	regfile	72 (int), 72 (FP)
L2 cache	2*MB*, 8-way, eight cycles	branch predict	gshare (8*K* entries)
execution	2-way out-of-order	memory access	315 cycles

use performance and power numbers reported by our simulators to estimate the throughput and power impact of errors for a given error recovery overhead.

We use a similar methodology to obtain processor-wide power estimates. To obtain a dynamic power estimate, we scale the dynamic power numbers reported by Wattch [52] for the optimizable components by the ratio of (total module power for a given optimization technique) over (total module power for the baseline design), as reported by *Synopsys PrimeTime* [29]. For designs that exploit application-based error resilience, we scale the power of the module groups independently, as we did for error rate. For the non-optimizable components, the Wattch numbers are scaled based on the minimum voltage at which those components can run without producing timing errors. For static power estimation, we use the ratio of dynamic and static module power for a given optimization technique, as reported by PrimeTime, to calculate static power for a dynamic power value obtained using the above methodology.

When a processor designed for application-level reliability runs an application that requires correctness, we scale down the frequency of the processor so that no timing violations occur. The safe clock frequency of the design is determined by the most negative-slack timing path in the processor plus a safety margin. All of our application simulations are executed for one billion cycles after fast-forwarding to the early SimPoints [210].

5.2 Minimization of Overheads for Error-Tolerant Designs

To reduce the overhead of error resilience, an intuitive strategy is to minimize the number of timing-critical endpoints, so that some error-tolerant registers may be replaced with conventional ones if their timing constraints are satisfied with sufficient margin. Figure 5.9 shows a circuit with Razor flip-flops and a plot for area (or power) of fanin-cone logic. If we upsize gates in the fanin cones until the corresponding endpoints have enough timing slack, the area (power) of the fanin cones increase. However, if we replace the error-tolerant register with a normal one, we save the overhead of error resilience. If the increased area (power) in fanin cones is smaller than the error-tolerance overhead, the replacement will be beneficial to total area (power).

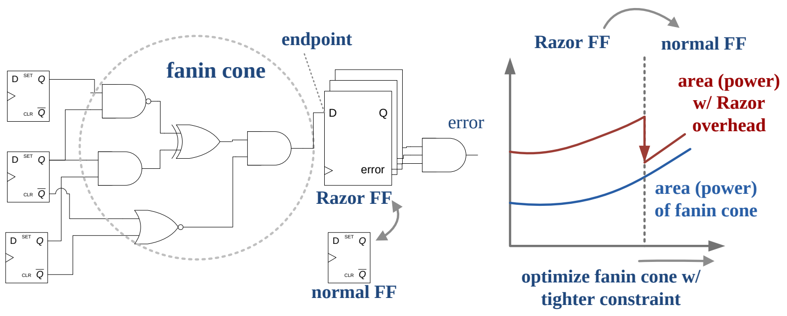

endpoint
fanin cone
error
Razor FF
normal FF
Razor FF
normal FF
area (power) w/ Razor overhead
area (power) of fanin cone
optimize fanin cone w/ tighter constraint

Figure 5.9: Tradeoffs between area and power of the fanin cone and overhead of a resilience mechanism (e.g., Razor [83], Razor-Lite [143] and TIMBER [69]).

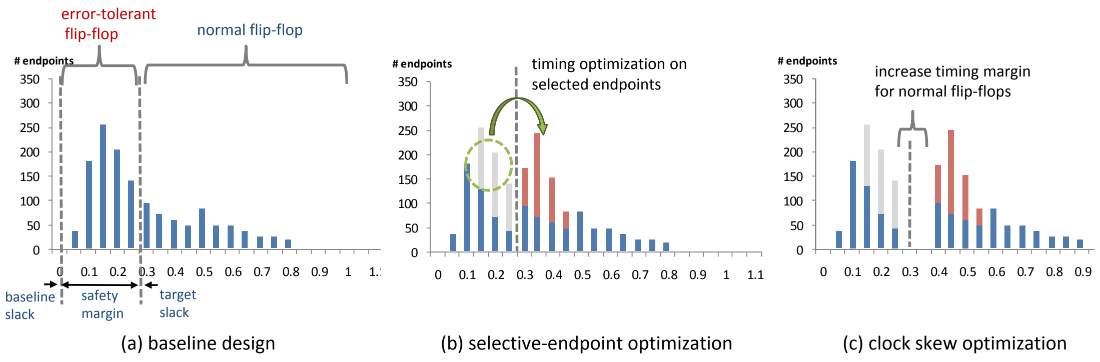


Figure 5.10: Slack distribution of endpoints in each stage of our optimization; (a) baseline design, (b) selective-endpoint optimization and (c) clock skew optimization.

In this section, we provide two design optimization techniques for error-tolerant design, (i) selective-endpoint optimization and (ii) clock skew optimization. Figure 5.10 shows the slack distribution of timing endpoints with respect to our optimization flows. In the baseline error-tolerant design (Figure 5.10(a)), several endpoints have timing violations, and error-tolerant registers are used for those endpoints. In our selective-endpoint optimization (Figure 5.10(b)), we tightly optimize a set of selected endpoints to reduce error-tolerance overhead. In the clock skew optimization (c), we increase timing margin for normal flip-flops by optimizing the clock arrival time to individual endpoints.

5.2.1 Selective-Endpoint Optimization

To reduce the overhead of error-tolerant registers, we propose a selective-endpoint optimization which minimizes a cost function (area, power) using tradeoffs between error-tolerance overhead and data path optimization. In this approach, we determine (i) '*how many endpoints should be optimized?*' and (ii) '*which endpoints should be optimized?*'

For Question (i), the more endpoints are optimized, the less error-tolerant registers can be used, but the cost of optimization increases as well. To minimize a given cost function, we check the cost function (area, power) impact of additional optimized endpoints, and select endpoints for optimization such that the resultant target number of design has minimum cost.

For Question (ii), each endpoint has a different cost of optimization. Figure 5.11 shows the cost of optimization for each endpoint: we see that area and power cost increase as the specified maximum delay constraints are reduced. Furthermore, the optimization cost increases significantly for the endpoint which has a large number of timing-critical fanin cells (i.e., negative-slack cells in the fanin cone of the endpoint). Therefore, we prioritize the optimization of endpoints which have small magnitude of timing slack and few timing-critical fanin cells.

According to the results of Figure 5.11, we define a new sensitivity measure for endpoints p in Equation (5.4). $slack(p)$ is the (worst negative) timing slack of endpoint p. $fanin(p)$ represents the number of timing-critical cells which are in the fanin cone of the endpoint p. In our optimization, we choose endpoints with small sensitivity values.

$$sensitivity(p) = |slack(p)| \times fanin(p) \tag{5.4}$$

Algorithm 9 describes our selective-endpoint optimization to reduce the error-resilience overhead. The procedure takes a netlist N which has error-tolerant registers (Razor flip-flops)

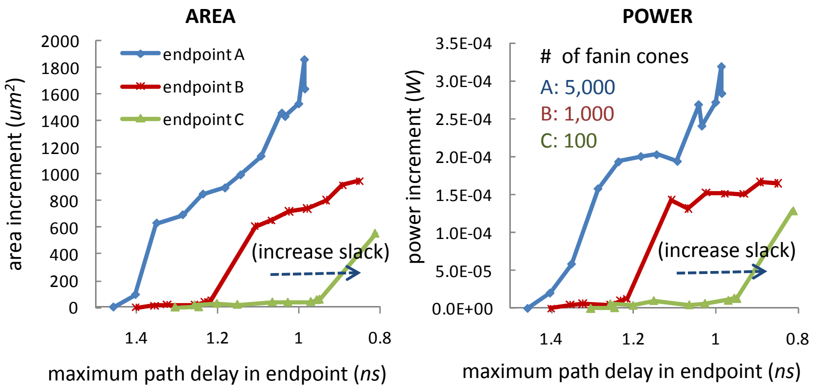


Figure 5.11: Cost (area and power increment) of endpoint optimization. Three endpoints with different numbers of timing-critical fanin cells (A: 5000, B: 1000, C:100) are optimized, and different delay constraints (from 1.4ns to 0.8ns) are applied for each endpoint.

on negative-slack endpoints. The procedure runs static timing analysis (STA) and computes a sensitivity for each endpoint p (Lines 1–8). The sensitivity function of Equation (5.4) is used. The procedure finds all fanin cells by tracing backward from the endpoint register using depth-first search (DFS). During the fanin-cone tracing, we only count timing-critical fanin cells since non-critical fanin cells will have little effect on the cost of endpoint optimization. The procedure optimizes top-k endpoints according to the sensitivity in each iteration (Lines 11–26). $TimingOpt(N, P)$ (Line 13) represents a timing optimization on the set of endpoints P in netlist N. $ISTA(N, P)$ is an incremental static timing analysis (STA) after the optimization. If the timing slack of endpoint p becomes positive, the procedure replaces the flip-flop of p with a normal flip-flop. Then, the relevant cost (area or total power) of netlist ($COST(N)$) is updated. After the iterations of endpoint optimization, the procedure returns a netlist (N_{min}) which has a heuristic minimum in area or power consumption cost.

5.2.2 Clock Skew Optimization

After selective-endpoint optimization, positive-slack endpoints are converted to normal flip-flops. If PVT variations do not go beyond the worst-case bound that is implicit in the corner-based timing library, timing yield is maintained. However, for many reasons, if any slight variation at the worst-case corner occurs, endpoints having near-zero slack values that are implemented as normal flip-flops can result in timing error. To maximize the timing tolerance for those endpoints, we propose to use clock skew optimization as a post-processing step following our previously-described selective-endpoint optimization.

Algorithm 9 Selective-Endpoint Optimization (SEOpt).

```
   Procedure $SEOpt(N)$
   Input: initial netlist $N$
   Output: optimized netlist $N_{min}$
 1: Run $STA$ to initialize slack values for the netlist $N$;
 2: $P \leftarrow \emptyset$;
 3: for all timing endpoint $p$ in the netlist $N$ do
 4:     if $slack(p) < 0$ then
 5:         $p.sensitivity \leftarrow |slack(p)| \times fanin(p)$;
 6:         $P \leftarrow P \cup \{p\}$;
 7:     end if
 8: end for
 9: $m \leftarrow |P|/k$;
10: $C_{min} \leftarrow \infty$;
11: for $i = 1$ ; $i \leq m$ ; $i \leftarrow i + 1$ do
12:     Pick the top-$k$ endpoints $P_i$ with minimum $sensitivity$ in $P$;
13:     $N_i \leftarrow TimingOpt(N_{i-1}, P_i)$;
14:     $ISTA(N_i, P_i)$;
15:     for all endpoint $p$ in $P_i$ do
16:         if $slack(p) \geq 0$ then
17:             Replace $Razor$ flip-flop by normal flip-flop at $p$;
18:         end if
19:     end for
20:     $C_i \leftarrow COST(N_i)$;
21:     if $C_i < C_{min}$ then
22:         $C_{min} \leftarrow C_i$;
23:         $N_{min} \leftarrow N_i$;
24:     end if
25:     $P \leftarrow P \setminus P_i$;
26:     Update $sensitivity$ of all endpoints in $P$;
27: end for
```

Clock skew scheduling or *useful skew* optimizes the arrival times of the clock signal to individual sinks. Fishburn [85] presents two linear programs for clock skew optimization to improve (i) minimum cycle time (*LP_SPEED*) or (ii) timing tolerance (*LP_SAFETY*), respectively. In our work, the target clock is fixed, and we maximize timing tolerance for the positive-slack endpoints, similar to Fishburn's *LP_SAFETY*. At the worst corner, to avoid double-clocking between flip-flops FF_i and FF_j, the clock arrival times to each flip-flop satisfy the following inequality.

$$x_i + T_{i,j} + T_{setup,j} \leq x_j + T_{clock} \tag{5.5}$$

where x_i and x_j are tunable clock arrival times for flip-flops FF_i and FF_j respectively, $T_{i,j}$ is the maximum signal delay of the timing paths from the clock pin of FF_i to the data pin of FF_j,

$T_{setup,j}$ is the setup time for FF_j, and T_{clock} is the given clock cycle time. From Equation (5.5), timing slack between two flip-flops FF_i and FF_j is defined as

$$slack(i,j) = (x_j + T_{clock}) - (x_i + T_{i,j} + T_{setup,j}) \tag{5.6}$$

The *LP_SAFETY* introduces a new variable M for timing tolerance. M is added to each endpoint that satisfies Equation (5.5) to determine clock arrival time x_i for each endpoint, so that the timing tolerance M for all endpoints is maximized. However, in the design with error-tolerant flip-flops, *LP_SAFETY* cannot determine the timing tolerance M, since the inequality in Equation (5.5) is already broken due to the negative-slack endpoints. We have extended the idea of *LP_SAFETY* to be applicable to a design with negative-slack endpoints. Specifically, we maximize M for only positive-slack endpoints while not worsening the slack of the negative-slack endpoints beyond the safety margin T_{safety} which is curable by the error-tolerant flip-flop. The proposed *Modified-LP_SAFETY* is formulated as

• *Maximize:* M
• *Subject to:* for all pairs of flip-flops FF_i and FF_j,

$$x_j - x_i - M \geq -T_{safety} - slack(i,j), \text{ if } slack(i,j) < 0 \tag{5.7}$$

$$x_j - x_i - M \geq T_{i,j} - T_{setup,j} - T_{clock}, \text{ if } slack(i,j) \geq 0 \tag{5.8}$$

Equation (5.8) is the same as in *LP_SAFETY*, but we add Equation (5.7) to prevent the slack of the negative-slack endpoints from being worsened beyond the safety margin T_{safety}. The maximum timing tolerance M can be found using linear programming (LP) solvers.

5.3 Experimental Results

We now evaluate our recovery-driven design implementations, which redistribute timing slack to reduce the error rate at a given voltage, allowing a reduction in voltage and energy for a given target error rate and operating frequency.

5.3.1 Evaluation of Heuristic Design Choices

Figure 5.12 shows power and runtime of the various heuristic design alternatives that we have evaluated, as described in Section 5.1.5. For **path ordering during optimization**, considering the slack in the prioritization function results in higher power than the case where only toggle rate is used. Runtime is somewhat smaller, but since our optimization iterates over

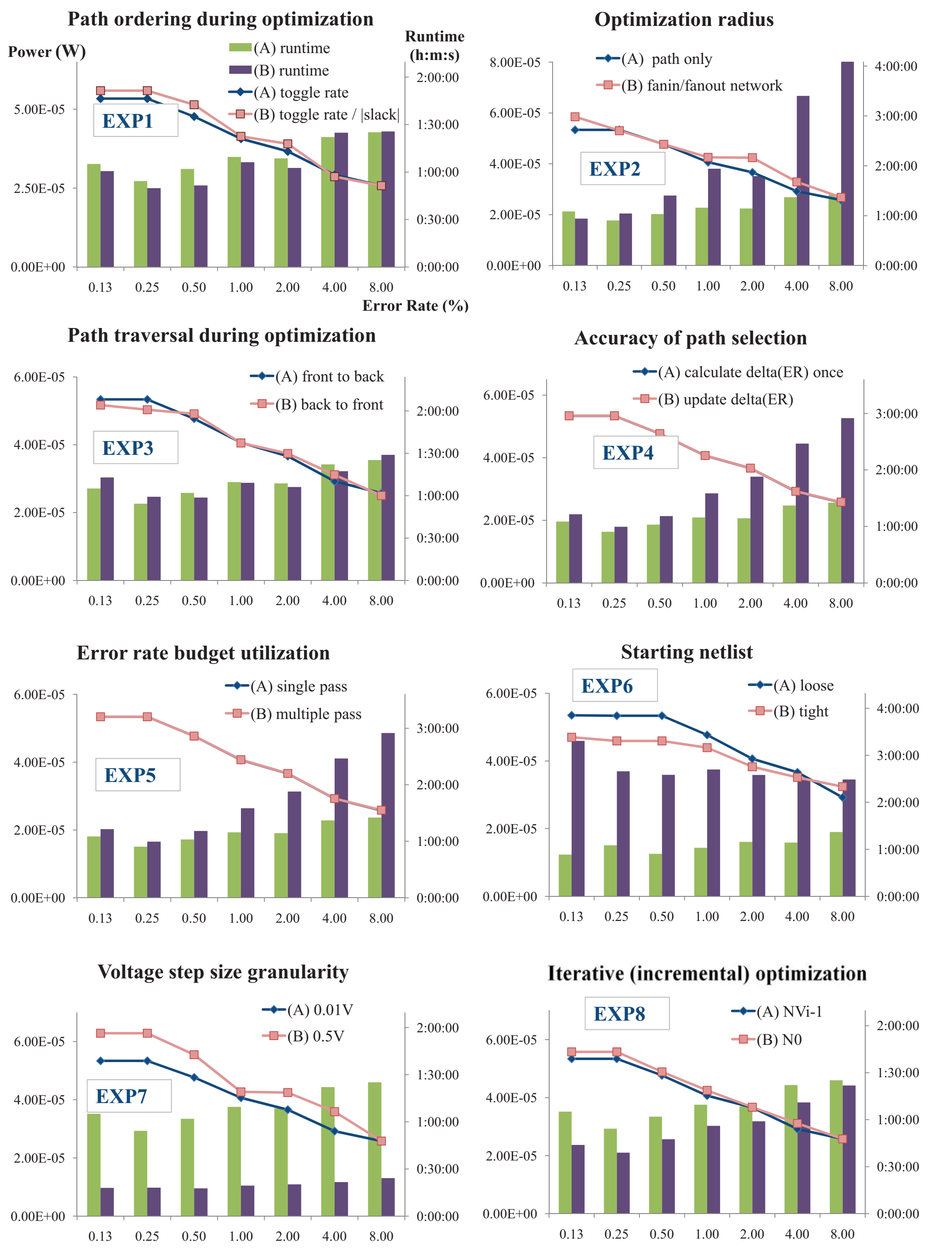


Figure 5.12: Evaluation of different heuristic design choices.

a path multiple times until no slack increase is observed, both results perform similarly. For the same reason, **path traversal order** has little effect on the optimization result. We choose the toggle rate priority function for its simplicity and lower power.

The results for **optimization radius** show that swapping cells in the fanin and fanout networks not only increases power at some error rates, but also greatly increases runtime due to the large amount of swaps that are performed. Thus, we choose to swap cells only on the optimized path. In the experiments on **accuracy of path selection** and **error rate budget utilization**, we observe no difference in power. Both updating the error rate contribution continuously during path selection and ensuring full utilization of the error rate budget increase runtime significantly without providing power benefits, and these techniques are not used in the final heuristic implementation.

The choices of **starting netlist** and **voltage step size** have a significant effect on power. Our recovery-driven design heuristic employs two main procedures – *OptimizePaths* (cell upsizing to reduce the error rate) and *ReducePower* (cell downsizing to reduce area and power). When starting the optimization flow from a loosely constrained design, path optimization provides the most substantial contribution to power reduction by reducing the error rate and extending voltage scaling. However, when starting from a tightly constrained design, much optimization has already been performed, and the power reduction stage of our heuristic is essential for power minimization. Although runtime increases due to evaluation of more downsizing moves, a tightly constrained netlist provides a better starting point, since it permits more voltage scaling. Voltage scaling has a stronger effect on power reduction and scales the power of all cells, while area reduction only affects the downsized cells. Also, starting from a tightly constrained design reduces the dependence on ECO, which improves the optimization efficiency. Using a coarser-granularity voltage step reduces runtime significantly, but comes at the cost of power, since the heuristic cannot hone in on the optimal voltage as easily. For higher error rates, a large step size can provide a near-optimal power result and a large reduction in runtime. Thus, an error rate-aware adaptive step sizing can be beneficial.

In terms of **iterative (incremental) optimization**, we observe that our heuristic is able to achieve the same result independent of the starting netlist. Thus, we choose the option that minimizes runtime.

5.3.2 Comparison with Alternative Flows

To demonstrate the benefits of our recovery-driven design flow, we compare five alternative design flows – traditional P&R implementations with conventional and tight timing constraints, a *BlueShift*-like path constraint tuning (PCT) approach, gradual slack design [129] [127], and our heuristic for error rate-optimized recovery-driven design. Figure 5.13 compares the power consumptions of the various design techniques at several target error rates.

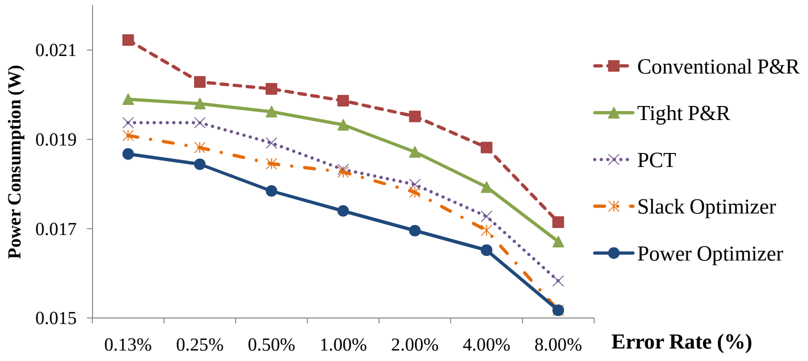


Figure 5.13: Power consumption of each design technique at various target error rates for target modules in Table 5.1.

Recovery-driven designs reduce power by enabling extended voltage scaling and keeping area overhead low with respect to other optimization techniques. Compared to a conventionally optimized design, a recovery-driven design operates at a much lower voltage for a given target error rate, due to the function-aware optimization approach that optimizes the paths that cause the most errors. Compared against a highly-optimized design that uses tightly-constrained P&R, a recovery-driven design reduces power by minimizing the amount of area spent on path optimization. Traditional tightly-constrained designs are functionally agnostic and optimize all paths heavily, incurring a large area overhead. Recovery-driven designs, on the other hand, use functional information to target only the paths that cause the most errors, thereby minimizing the area cost of additional voltage scaling. In scenarios where the cost of area is high, such as when leakage currents increase as has been forecasted for future technology nodes [18], the cost of functionally-agnostic optimizations will increase, and the benefits of recovery-driven design will increase. Table 5.4 shows power savings for recovery-driven design for each module with respect to traditional P&R at different target error rates.

In our power minimization heuristic, after deciding how to allocate the error rate budget, the $ReducePower$ stage performs aggressive cell downsizing to reduce circuit area and power. Table 5.5 compares the recovery-driven design against other design flows in terms of area overhead with respect to the baseline design. Designing for a target error rate has similar area overhead to PCT but still produces a design with lower power. The reason is that designing for a target error rate allows more aggressive voltage scaling before the target error rate is exceeded. At lower voltages, there are more negative-slack paths to be optimized during *OptimizePaths*, which increases area overhead. However, aggressive downsizing keeps area overhead low, and since the paths targeted by $PowerOptimizer$ are the paths that cause the most errors in the design, the area is well spent, and the additional voltage scaling contributes to a net benefit in terms of power savings. PCT, on the other hand, adds tighter timing constraints to the registers where the most errors are captured and optimizes all paths with endpoints at those registers. Since our heuristic targets paths individually, we can target the error-causing paths more efficiently, reduce overhead, and increase voltage scaling and power savings.

Compared to tightly constrained P&R and gradual slack design, designing for a target error rate incurs significantly less area overhead and reduces power. On one hand, tightly constrained P&R is functionally agnostic and fails to identify the set of paths that maximizes voltage overscaling per unit area overhead. Gradual slack design, on the other hand, optimizes the design to make tradeoffs between power, throughput, and reliability over a *range* of error rates. Thus, a gradual slack design is over-optimized for any single target error rate.

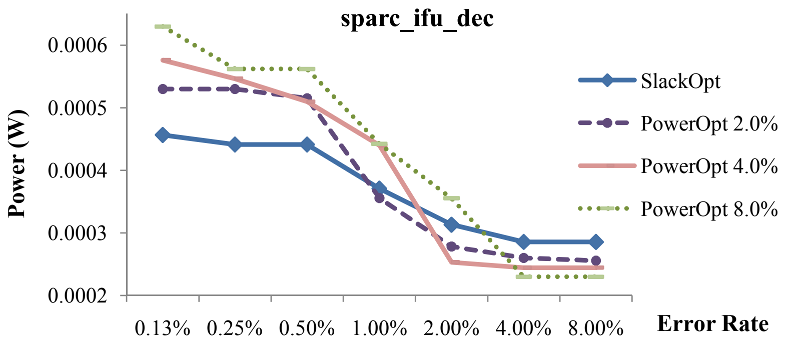


Figure 5.14: Comparison between recovery-driven design (*PowerOpt*) and gradual slack design (*SlackOpt*).

Figure 5.14 compares recovery-driven design for a target error rate against gradual slack design. The results show that designing for a target error rate minimizes power at the target error rate. However, since a recovery-driven design can have a non-zero error rate even under nominal conditions, power efficiency at error rates lower than the target may drop off steeply. Likewise, since designing for a target error rate creates a slack wall at the error-optimal voltage, additional benefits for error rates higher than the target are limited. A gradual slack design, on the other hand, is optimized for a *range* of error rates. Although this means that it is less efficient than an error rate-optimal design for any single error rate, it also means that performance or output quality can be efficiently traded for power savings over the entire range of error rates. Thus, whenever more errors can be tolerated, a gradual slack design can reduce power consumption. This may not be possible for an error rate-optimal design, since it forgoes scalability to achieve additional power savings at the target error rate.

Recovery-driven design optimizes for errors in the average operating behavior of a design. If the frequently exercised paths during operation are significantly different from those targeted during optimization, then too many errors may be produced, and voltage scaling may be limited for a target error rate. To evaluate the robustness of recovery-driven design when the workload changes, we compared the power reduction achieved when running the training (optimization) benchmarks against power reduction achieved for the test benchmarks. Figure 5.15 shows that power reduction is slightly higher for the benchmark set that the processor was optimized for, but the difference is only about 1% on average.

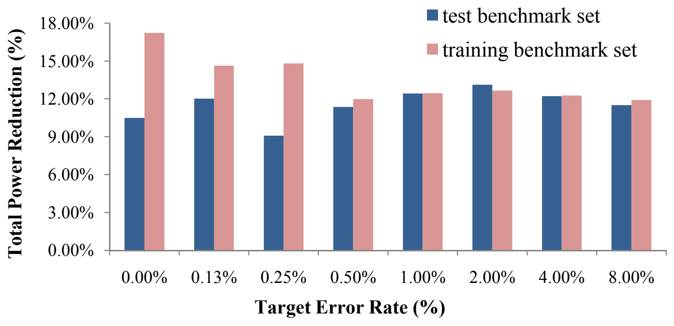


Figure 5.15: Total power reduction for training (optimization) and test benchmark sets.

5.3.3 Variation-Aware Analysis

Recovery-driven design increases energy efficiency by reshaping the slack distribution of a design, such that error rate is reduced at a particular voltage. Figure 5.16 shows activity-weighted slack distributions (sum of path toggle rate versus timing slack) from before and after optimization, confirming that the optimization increases slack for frequently exercised paths, which enables extended voltage scaling for a target error rate. However, due to random variations introduced in the physical circuit by sources of static and dynamic non-determinism, the actual slack distribution may be somewhat different than the designed distribution.

Table 5.4: Power savings (%) for error rate-optimized recovery-driven designs compared to traditional P&R.

module	target error rate (ER_{target})						
	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%	8.0%
lsu_dctl	29.1	16.8	16.8	16.8	16.8	16.8	21.6
lsu_qctl1	8.8	6.7	5.8	8.1	11.0	9.0	8.6
lsu_stb_ctl	17.9	17.9	18.1	15.4	9.6	19.2	2.9
sparc_exu_ecl	6.0	6.0	18.3	18.3	22.7	23.3	17.4
sparc_ifu_dec	13.7	10.1	8.6	14.3	15.9	18.5	15.1
sparc_ifu_errdp	2.2	2.8	5.7	5.7	5.7	9.3	9.3
sparc_ifu_fcl	14.5	15.4	16.5	19.2	19.2	19.2	19.2
spu_ctl	13.1	13.1	13.1	13.2	8.8	1.6	8.9
tlu_mmu_ctl	0.8	0.8	0.8	0.8	0.8	0.8	0.8

Table 5.5: Average area overhead with respect to the baseline.

tight P&R	PCT	SlackOpt	PowerOpt 0.125%	PowerOpt 0.25%
19.1%	5.0%	11.9%	3.9%	4.3%
PowerOpt 0.5%	PowerOpt 1%	PowerOpt 2%	PowerOpt 4%	PowerOpt 8%
4.8%	5.4%	5.8%	6.0%	5.3%

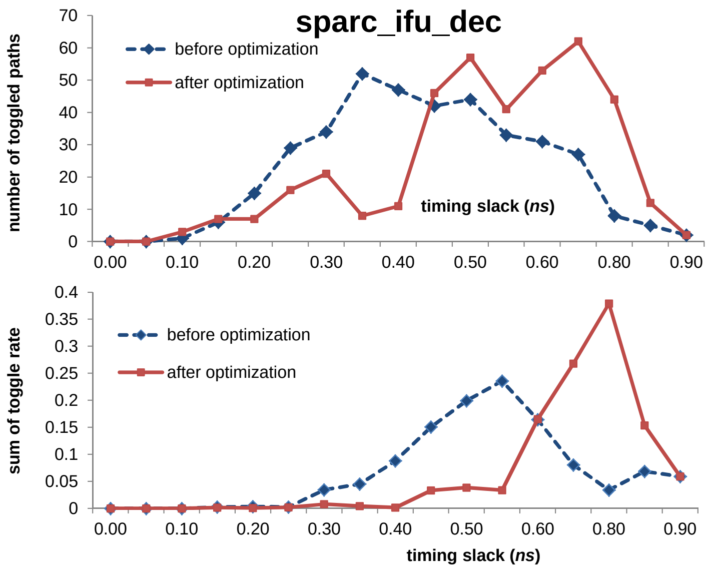


Figure 5.16: Slack distribution (above) and activity-weighted slack distribution (below) after recovery-driven design (testcase: *sparc_ifu_dec*).

To test the benefits of recovery-driven design in the presence of variations, we have implemented a model for inter-die and spatially correlated within-die variations based on the models in [61] [106]. We use an exponential model for correlation between different die locations, in which the correlation function decays exponentially as a function of distance, with parameters supplied by the authors of [106]. We extract standard deviations (σ) of cell delay at each operating voltage from SPICE simulations, and use our variation model to assign a random delay variation to each die and each gate within the die, based on its location. We then repeat error rate and power estimation with one hundred different random variation maps. From the Monte Carlo simulations, we report total power consumption of the target modules at each error rate in Table 5.6. Table 5.6 shows that even when variations are accounted for, recovery-driven design still achieves significant power savings over a conventional design. Furthermore, the average benefits do not noticeably change when variations are accounted for. (Power reduction in Table 5.6 is somewhat lower for error rates below 1% because the test design was optimized for a target error rate of 1%.) Random variations cause perturbations within a design but do not shift the average-case behavior. Since recovery-driven designs are optimized for and operate at the average-case operating point, they are naturally robust to random variations.

Table 5.6: Variation-aware analysis.

	target error rate (ER_{target})						
	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%	8.0%
power consumption (*W*) in baseline design							
minimum	0.0126	0.0126	0.0122	0.0113	0.0108	0.0106	0.0095
maximum	0.0202	0.0201	0.0199	0.0196	0.0191	0.0186	0.0167
average	0.0162	0.0160	0.0156	0.0153	0.0149	0.0141	0.0127
power consumption (*W*) in recovery-driven design							
minimum	0.0111	0.0106	0.0105	0.0096	0.0092	0.0088	0.0080
maximum	0.0187	0.0183	0.0175	0.0172	0.0165	0.0161	0.0151
average	0.0148	0.0144	0.0141	0.0134	0.0128	0.0123	0.0113
power reduction (%)							
average	8.28	9.71	9.43	12.61	13.80	13.03	11.18

5.3.4 Recovery-Driven Processors

We now demonstrate the benefit of designing processors for specific hardware and software error-resilience mechanisms, as described in Section 5.1.8.

Circuit-Level Timing Speculation. Figure 5.17 compares the energy consumption of a recovery-driven processor that has been designed and optimized for Razor against the power consumption of processors designed for other objectives, such as gradual slack or PCT, and against processors that have been designed for correctness but use the traditional Razor methodology to save energy. We assume a recovery overhead of nine cycles, proportional to the pipeline depth of the processor.

Figure 5.17 demonstrates that the minimum energy is indeed achieved by a processor that is designed to produce errors that can be gainfully tolerated by Razor. Designing the processor for the error rate target at which Razor operates most efficiently allows us to extend the range of voltage scaling from $0.84V$ for the best "designed for correct operation" processor to $0.71V$ for the processor designed for an error rate of 1%, affording an additional 19% energy reduction.

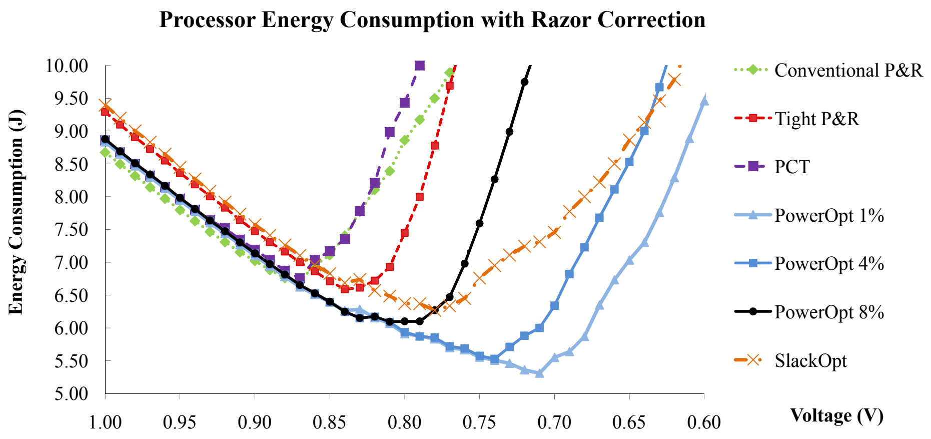


Figure 5.17: Comparisons of the energy consumption of a recovery-driven processor.

Error recovery with a circuit-level approach such as Razor imposes a throughput penalty, since error recovery requires feeding correct values back into the pipeline. Figure 5.18 shows the throughput reduction caused by error recovery for a correction overhead of five cycles. As can be seen, a recovery-driven processor minimizes the recovery overhead even at the target operating voltage.

Application Noise Tolerance. To demonstrate the benefits of recovery-driven design targeted at application-level noise tolerance, we use a face detection algorithm [234] as the example application. Face detection is naturally robust to errors in several processor modules and does not require strict computational correctness. Rather than causing program failure, errors may result in reduced output quality (false positive or negative detections) [204].

Face detection, as well as the other error-tolerant applications we consider, tolerates errors in the arithmetic units of the processor. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

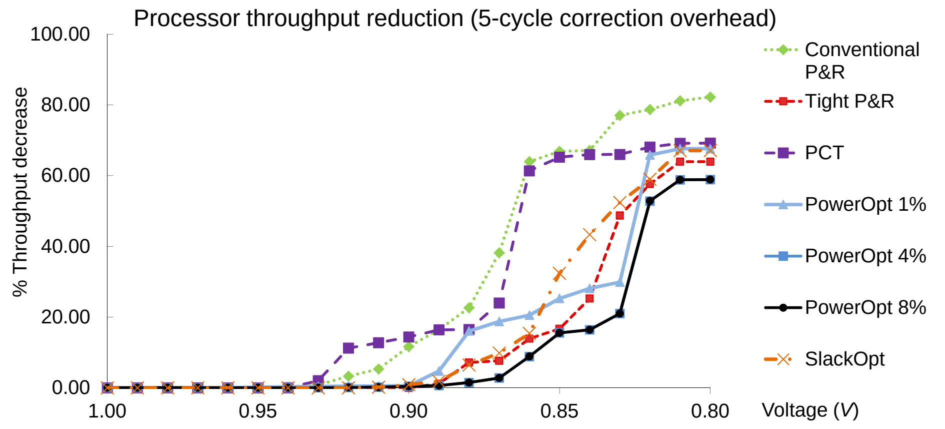


Figure 5.18: Throughput reduction at different voltages with an error recovery overhead of five cycles.

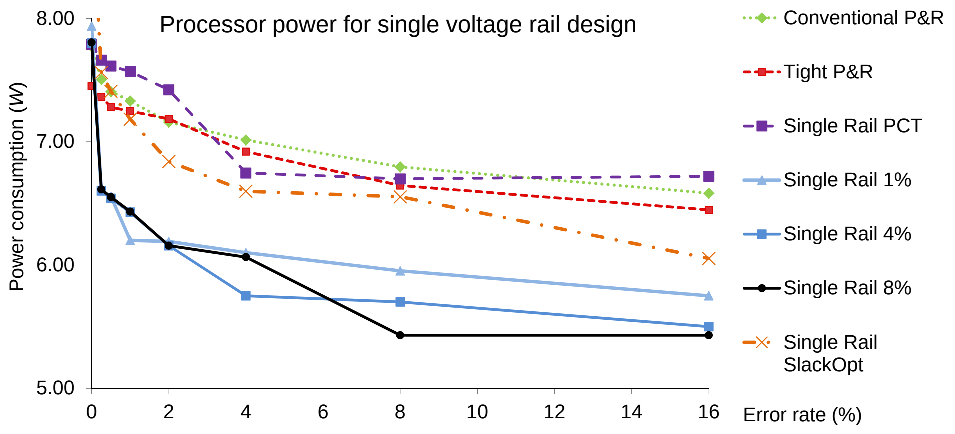


Figure 5.19: Power consumption of processors using a single voltage rail design.

Figures 5.19 and 5.20 compare the power consumption of processors designed for application-level error tolerance of arithmetic errors using single and dual voltage rail designs, as described in Section 5.1.8. In these figures, all processors achieve the same output quality at a given error rate, but processors designed to allow errors consume less power, and power is minimized for these designs at their respective error rate targets. For example, at an error rate

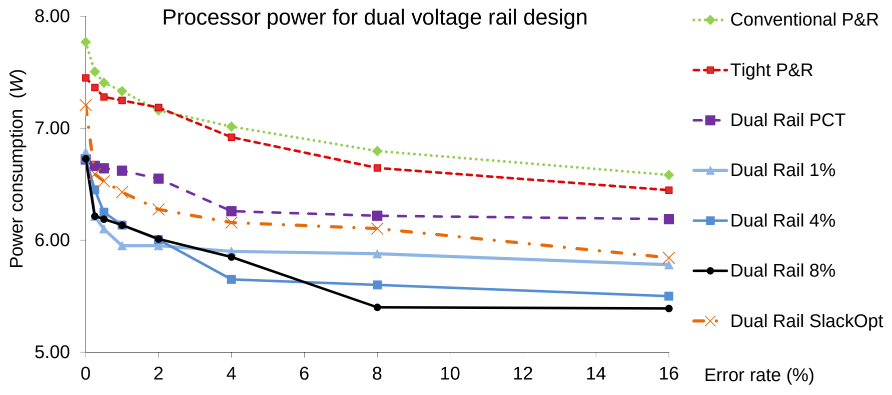


Figure 5.20: Power consumption of processors using a dual voltage rail design.

of 1%, where output quality is still maximized for the face detection application, the processor designed for an error rate target of 1% consumes 19% less power for dual-rail design and 15% less power for single-rail design than the baseline correctness-optimized processor. Benefits are even higher for larger error rates if some application output degradation is permissible.

Note that we can always perform error-free computation on a core designed for application-level noise tolerance by scaling down the frequency to the point where all paths have positive slack. However, this may represent a performance penalty when compared to relaxed-correctness operation. Further, trends in processor-level results may differ somewhat from trends in averaged module-level results. Whereas the power reduction of a recovery-driven design is limited by a module's critical paths, the power reduction of a recovery-driven processor is biased by the critical modules that begin causing errors first when voltage is scaled down. As we show in the following discussion, results can be improved by utilizing multiple voltage domains.

5.3.5 Supporting Multiple Voltage Domains

Given a target error rate, the module-level power minimization heuristic in [128] selects an optimal operating voltage for a given processor module. However, the proposed processor core-level methodology (Algorithm 1, $DOMAINS = 1$) selects a common voltage for all modules of a processor core. Table 5.7 shows that different modules vary (sometimes substantially) in their optimal voltage operating points due to a number of factors, including module area (number of paths/cells), slack distribution (fraction of paths that are critical), and activity factor (frequency of paths toggling). In addition, the table shows that the range of optimal module voltages increases when designing for a non-zero error rate target.

Because of the above module-level variations, there can be a substantial difference in power consumption between the locally and globally optimized module implementations. Figure 5.21 quantifies the difference between single and multiple voltage domain design for processor cores that tolerate different error rates. We compare designs with different numbers of voltage domains, targeting different processor error rates in terms of their power consumption relative to a processor optimized for a common operating voltage. The results show that the power efficiency of recovery-driven processors improves significantly with the number of supported voltage domains. In practice, the number of voltage domains should be chosen by carefully balancing the voltage overscaling benefits with the area and complexity overheads of supporting multiple power rails. The results of Figure 5.21 do not consider the overhead of level shifter circuitry.

Table 5.7: Optimal module voltages at different target error rates.

	target error rate (ER_{target})						
module	0.0%	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%
lsu_dctl	0.75	0.72	0.71	0.75	0.74	0.73	0.72
lsu_qctl1	0.88	0.87	0.86	0.85	0.84	0.83	0.80
lsu_stb_ctl	0.77	0.76	0.75	0.75	0.70	0.68	0.66
sparc_exu_ecl	0.75	0.74	0.73	0.70	0.70	0.69	0.70
sparc_ifu_dec	0.68	0.67	0.66	0.63	0.70	0.58	0.57
sparc_ifu_errdp	0.77	0.58	0.57	0.56	0.55	0.54	0.53
sparc_ifu_fcl	0.79	0.77	0.76	0.75	0.74	0.73	0.72
spu_ctl	0.78	0.65	0.64	0.63	0.62	0.63	0.58
tlu_mmu_ctl	0.85	0.52	0.51	0.51	0.51	0.51	0.51
range	0.20	0.35	0.35	0.34	0.33	0.32	0.29

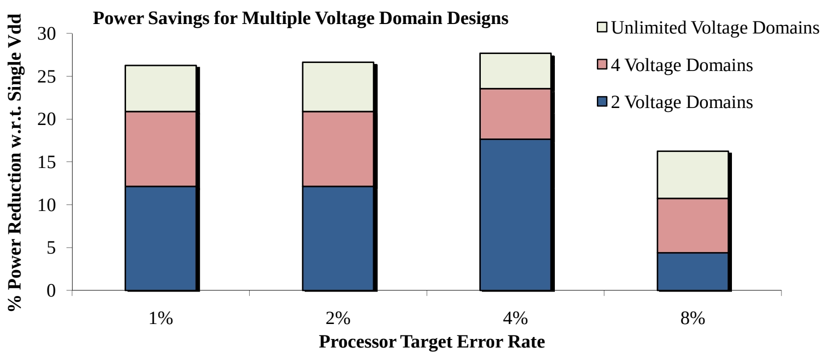


Figure 5.21: Benefits of multiple voltage domain design over single voltage domain design.

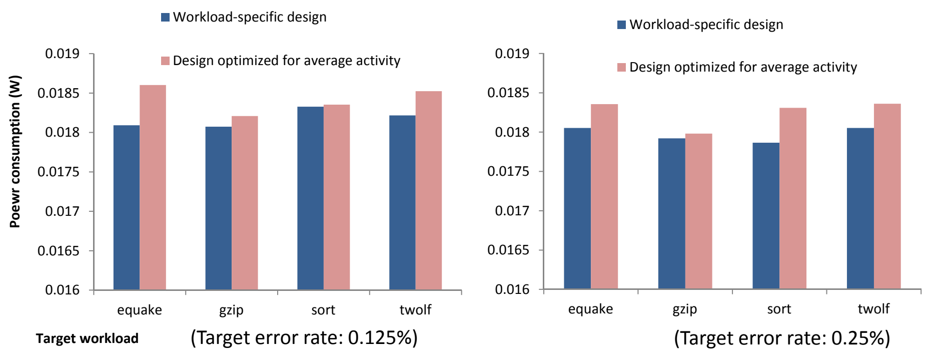


Figure 5.22: Comparisons between optimized designs; workload-specific and average cases.

5.3.6 Robustness to Application Diversity

Different workloads exercise the timing paths of a processor core differently. Thus, the sets of frequently-exercised and infrequently-exercised paths may change, depending on the workload. Since recovery-driven designs are optimized according to an average-case activity profile, it is important to ensure that power efficiency is not degraded significantly when the activity profile of a workload is not the same as the activity profile for which the processor was optimized.

To gauge the robustness of recovery-driven design to workload diversity, we create several recovery-driven designs, optimized for the activity profiles of each benchmark in the test set – *equake*, *gzip*, *sort*, and *twolf*. Then, we compare the power consumption of each benchmark in the test set, running on the design that was optimized for the average-case, against the design that was optimized specifically for that benchmark. Figure 5.22 compares the power consumption of average-case design against workload-specific designs for different target error rates.

On average, the difference is small – only 1.5% difference in power at an error rate of 0.125% and 0.9% difference at 0.25% – demonstrating the robustness of recovery-driven design to application diversity. The difference decreases as the target error rate increases. The reason for this robustness is that since some paths are allowed to cause errors, there is some "forgiveness" when the priority of path optimization deviates somewhat from the optimal. Our recovery-driven design heuristic bins paths into P_- paths that are allowed to cause errors and P_+ paths that should remain error free. As long as the difference in activity for a path is not so large as to make the path switch bins, the path dichotomy is preserved and power efficiency is not degraded. In the worst case, we observe only 3% degradation in power efficiency.

5.3.7 Selective-Endpoint Optimization

Table 5.8 shows area and total power results from the selective-endpoint optimization for *EXU* module. In the optimization, negative-slack endpoints are sorted according to the sensitivity, and 10% of the total endpoints are optimized in each iteration. Data in the first row (0% optimization) show the power and area of the initial implementation. We assume 50% power and 100% area overhead for the error-tolerant registers (i.e., Razor flip-flops). The design has large overhead for error-tolerant registers, since a large number of endpoints have timing violations at the target frequency. By optimizing each selected endpoint, the number of error-tolerant registers is reduced. However, power and area overhead from the path optimization increases. From the results, we observe 4% area reduction and 6% total power reduction as compared to the initial implementation, when 30% total endpoints are optimized.

Figures 5.23 and 5.24 compare results from endpoint optimization with two different sensitivities – (a) considering timing slack only, and (b) considering both timing slack and the number of timing-critical fanin cells. We observe that the overhead from path optimization increases slowly in Figure 5.23 (b) – up to 30% endpoint optimization gives only 0.7% area and 0.3% power overhead. Therefore, consideration of both the slack and the number of timing-critical fanin cells is beneficial for the endpoint selection. Figure 5.25 compares area and power consumption across conventional design, error-tolerant design, and endpoint-optimized design. In the endpoint-optimized design, total design area and power consumption are saved by reducing the number of error-tolerant registers.

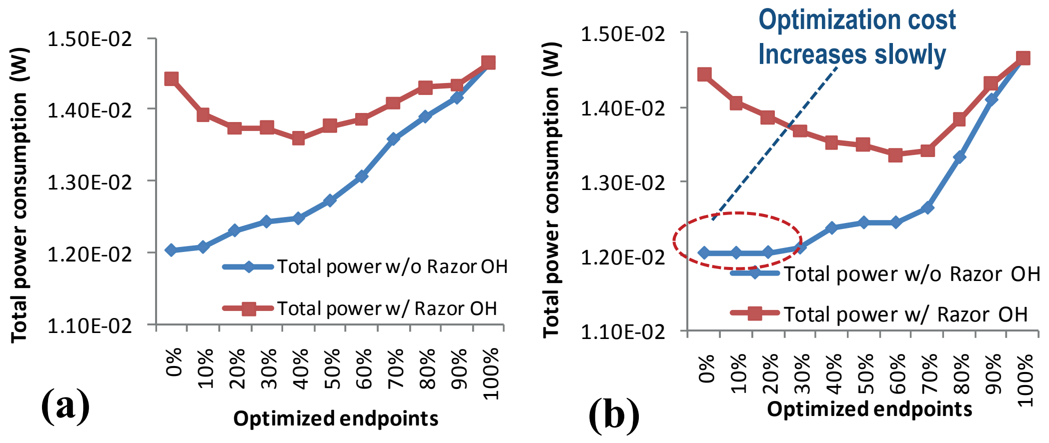


Figure 5.23: Total power results for selective-endpoint optimization – (a) considering timing slack only, and (b) considering both timing slack and the number of timing-critical fanin cells. (testcase: *MUL*)

Table 5.8: Area and power consumption with endpoint optimization (*MUL*).

optimized endpoint	without Razor flip-flop		with Razor flip-flop		
	area (um^2)	power (W)	# of Razor flip-flop	area (um^2)	power (W)
0%	40321	1.20E-02	840	48146	1.44E-02
10%	40349	1.20E-02	698	46891	1.41E-02
20%	40393	1.20E-02	611	46258	1.39E-02
30%	40641	1.21E-02	519	45676	1.37E-02
40%	41611	1.24E-02	371	45235	1.35E-02
50%	42028	1.24E-02	335	45307	1.35E-02
60%	42431	1.24E-02	290	45287	1.34E-02
70%	43418	1.26E-02	245	45850	1.34E-02
80%	47410	1.33E-02	161	49039	1.38E-02
90%	52685	1.41E-02	71	53412	1.43E-02
100%	55809	1.47E-02	0	55809	1.47E-02

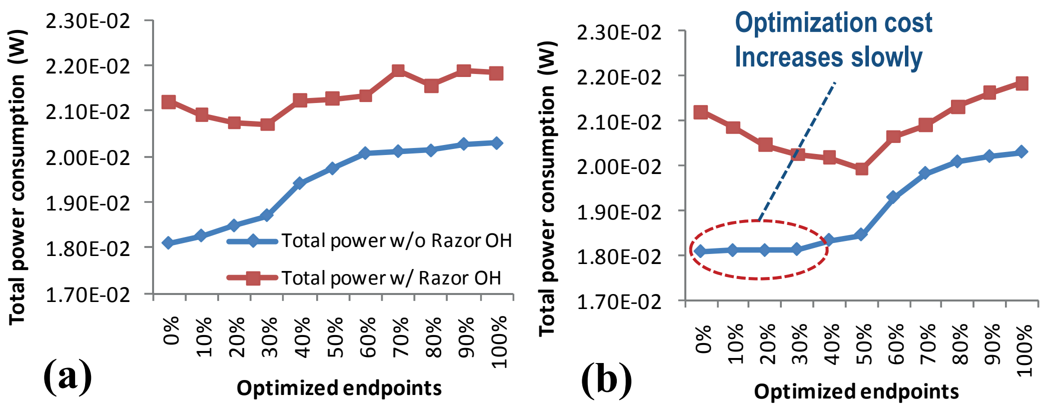


Figure 5.24: Total power results for selective-endpoint optimization – (a) considering timing slack only, and (b) considering both timing slack and the number of timing-critical fanin cells. (testcase: *EXU)*

To analyze the effect of overhead and safety margin of error-tolerant registers, we change the parameters and implement each testcase. In the experiments, three safety margins for the error-tolerant register are tested – 200ps, 300ps and 400ps. We also test with three cases of overheads for the error-tolerant register – 50% (L_{OH}), 100% (M_{OH}) and 200% (H_{OH}) in area, and 30% (L_{OH}), 50% (M_{OH}) and 100% (H_{OH}) in total power consumption. Table 5.9 shows the achieved power reduction and percentage of optimized endpoints, Table 5.10 shows the achieved area reduction. We see that endpoint-optimization does not give benefits with low error-tolerance

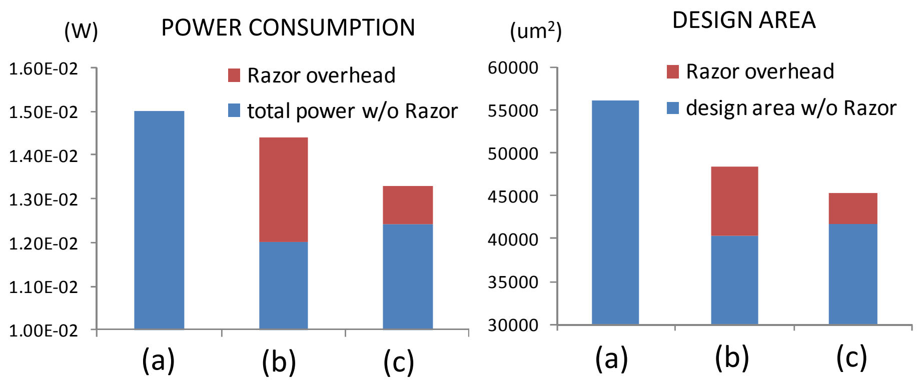


Figure 5.25: Area and power consumption in each design – (a) conventional design, (b) error-tolerant design, and (c) endpoint-optimized design. (testcase: *MUL)*

overhead; the error-tolerant registers are more beneficial in power and area than conventional flip-flops. When there is high overhead of error-tolerance (H_{OH} column), we observe more substantial reductions in power and area, as one would expect. It is important to realize that we cannot give a general trend with respect to the magnitude of safety margin, because for each combination of $\{200ps, 300ps, 400ps\} \times \{L_{OH}, M_{OH}, H_{OH}\}$ the baseline design is different, and we report the power reduction achieved with the best percentage of optimized endpoints.

Table 5.9: Power reduction and best percentage of optimized endpoints for each combination of Razor overhead and margin.

test	safety	power reduction			optimized endpoint		
case	margin	L_{OH}	M_{OH}	H_{OH}	L_{OH}	M_{OH}	H_{OH}
MUL	$200ps$	0.0%	2.8%	7.9%	0%	20%	60%
	$300ps$	1.5%	3.9%	9.5%	10%	20%	50%
	$400ps$	2.5%	5.4%	13.0%	30%	50%	70%
EXU	$200ps$	3.2%	5.0%	12.0%	20%	40%	80%
	$300ps$	3.0%	5.9%	11.9%	20%	50%	50%
	$400ps$	3.1%	5.9%	11.8%	10%	50%	50%
SPU	$200ps$	3.1%	6.4%	14.4%	70%	80%	90%
	$300ps$	3.3%	6.0%	11.8%	60%	60%	60%
	$400ps$	2.7%	5.4%	11.1%	40%	60%	60%
FPU	$200ps$	7.2%	13.0%	23.8%	70%	70%	70%
	$300ps$	10.2%	15.0%	24.0%	50%	50%	60%
	$400ps$	7.4%	12.1%	21.5%	50%	50%	70%

Table 5.10: Area reduction from the endpoint optimization.

test	safety	area reduction			optimized endpoint		
case	margin	L_{OH}	M_{OH}	H_{OH}	L_{OH}	M_{OH}	H_{OH}
	200ps	0.00%	0.97%	4.77%	0%	20%	20%
MUL	300ps	0.76%	3.16%	8.26%	10%	20%	20%
	400ps	1.39%	4.09%	11.18%	20%	30%	50%
	200ps	4.49%	5.13%	8.69%	20%	20%	40%
EXU	300ps	1.77%	3.92%	9.97%	30%	30%	50%
	400ps	1.62%	4.31%	10.41%	20%	50%	50%
	200ps	2.03%	4.19%	13.35%	30%	30%	90%
SPU	300ps	2.26%	4.70%	10.92%	30%	50%	60%
	400ps	1.95%	4.78%	10.62%	30%	40%	60%
	200ps	3.31%	7.44%	16.30%	30%	30%	70%
FPU	300ps	4.63%	9.99%	18.36%	40%	40%	60%
	400ps	2.76%	7.78%	15.59%	30%	40%	50%

5.3.8 Clock Skew Optimization

Figure 5.26 shows how *Modified-LP_SAFETY* modifies slack distribution profile for *MUL*. We can observe that most of near-zero positive-slack endpoints are clearly moved to the timing slack bin +0.18 – +0.20ns. Hence, higher tolerance to variation and thus higher timing yield are expected.

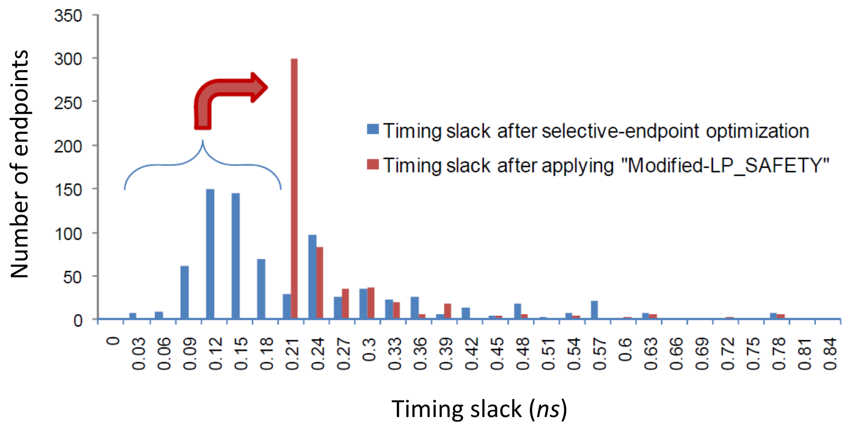


Figure 5.26: Slack histogram before and after applying *Modified-LP_SAFETY*. Only slack of the positive-slack endpoints is shown. Slack of the negative-slack endpoints does not exceed the given safety margin (T_{safety} = 300ps).

Table 5.11 summarizes the improved timing tolerance M, and the number of negative-slack endpoints from our selective-endpoint optimization (SEOpt) and after applying *Modified-*

LP_SAFETY (MLP). From the table, we observe that the timing tolerance is always improved, and can be improved by up to 187.37ps. However, when most near-zero positive-slack endpoints are related to each other, timing tolerance improvement can be small as in the *EXU* testcase.

A side benefit from the clock skew optimization is that slack of negative-slack endpoints can also be improved. Columns 3 and 4 in Table 5.11 show the number of negative-slack endpoints (i.e., the number of required Razor flip-flops.). The data suggest that we can further reduce the number of required Razor flip-flops if we do not require a timing tolerance guardband. For the four testcases (*MUL*, *EXU*, *SPU* and *FPU*) this implies respective further power reductions of 1.03, 0.75, 1.73 and 5.67%, and area reductions of 0.98, 0.73, 2.13 and 4.86%. We recognize that other combinations of selective-endpoint optimization and useful skew to find min-power and min-area Razor-based designs are possible. We leave this tuning of our methodology to follow-on work.

Table 5.11: Optimized timing tolerance M, and the number of negative-slack endpoints from selective-endpoint optimization and from *Modified-LP_SAFETY*.

Test	Timing	# of error-tolerant flip-flops		Imp. over SEOpt	
case	tolerance (ps)	SEOpt	MLP	Area	Power
MUL	187.37	332	282	0.98%	1.03%
EXU	0.21	496	444	0.73%	0.75%
SPU	68.24	78	5	2.13%	1.73%
FPU	30.55	1421	1172	4.86%	5.67%

5.4 Conclusions and Future Directions

In this chapter, we propose *recovery-driven design*, a design-level approach that optimizes a processor module for a target timing error rate instead of correct operation. We present a detailed evaluation and analysis of a recovery-driven design methodology that minimizes power for a target error rate. We extend our recovery-driven design flow to design recovery-driven processors – processors that are designed and optimized for a target error rate. We also present an extension of our recovery-driven design flow that creates a gradual slack design that is optimized for a range of error rates rather than a single target. The gradual slack technique is used to design soft processors that can trade throughput or output quality for energy savings over a range of reliability targets. While we have chosen to focus on improving the energy efficiency of error resilient designs, recovery-driven design can also be used to optimize other design characteristics, such as yield.

We also provide a new design flow for mixing of resilient and non-resilient circuits within a given implementation, which minimizes the overhead of error resilience. We have proposed a selective-endpoint optimization, which reduces timing-critical endpoints with small cost of timing optimization. We also propose a clock skew optimization in error-tolerant design, which improves robustness over process, voltage and temperature variations. From the proposed optimization techniques, we achieve 7.7% power and 5.4% area reduction on average, over conventional error-tolerant designs, and improve the timing variation tolerance by up to $187ps$.

In our selective-endpoint optimization, we currently do not consider error-recovery overheads, such as the additional cycles needed for rollback or instruction replay [83]. The recovery overheads increase with error rate, and this should be estimated from the activity information. Another current limitation of our work is that two approaches – endpoint optimization and clock-skew optimization – are performed independently of each other. are separated each other. By combining the two optimization knobs, we would likely further improve the power efficiency of error-tolerant designs. Our ongoing work seeks to (i) incorporate activity information into the sensitivity metric to minimize the impact of recovery on throughput, and (ii) build a unified framework for simultaneous data- and clock-path optimization.

5.5 Acknowledgments

Chapter 5 is in part a reprint of "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010; "Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs", *Proc. International Symposium on High-Performance Computer Architecture*, 2010; "Recovery-Driven Design: A Power Minimization Methodology for Error-Tolerant Processor Modules", *Proc. ACM/IEEE Design Automation Conference*, 2010; "Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012; and an unpublished 2010 draft coauthored with Dr. Kwangok Jeong.

I would like to thank my coauthors Dr. Kwangok Jeong, Professor Andrew B. Kahng, Professor Rakesh Kumar and Professor John Sartori.

Chapter 6

Enhancing the Efficiency of Energy-Constrained DVFS Designs

Given the energy overheads of DVFS designs produced by conventional multi-mode CAD flows, we explore the DVFS design space to identify the sources of DVFS inefficiency and scenarios in which DVFS designs are typically inefficient. Based on our insights, we propose a context-aware multi-mode design flow to enhance DVFS efficiency. Our context-aware approach takes into consideration the intrinsic characteristics of a design, the desired range of scalability, the relative amounts of time spent in different operating modes, and the desired energy efficiency metric to select appropriate design constraints and optimize the design for maximum efficiency over multiple modes of operation.

We also propose a selective replication-based methodology that identifies modules for which context-aware multi-mode designs are energy-inefficient; such modules are candidates for replication. Selective replication employs multiple replicas of modules that have been optimized for different performance targets, such that the appropriate replica is active for only a given operating scenario. Since replication adds an area overhead, we suggest replication for only the modules that are particularly inefficient in terms of scalability and energy efficiency.

Finally, our study of the sources of DVFS inefficiency allows us to identify microarchitectural features that affect DVFS efficiency. Thus, we are able to suggest microarchitectural changes that can improve DVFS efficiency in general or for a particular scenario.

The main contributions of our work are the following.

- We quantify DVFS inefficiency for different operating scenarios for conventional multi-mode CAD flows and identify the sources of inefficiency. We observe that the average power of a conventional multi-mode design may be up to 28% higher than the ideal average power.

- We propose a context-aware approach to multi-mode design that considers intrinsic characteristics of hardware, duty cycle and range of scalability to select energy-efficient design constraints. Our context-aware multi-mode design flow reduces average power by up to 20% with respect to a conventional multi-mode design flow.
- We propose a selective replication-based approach to improve the energy efficiency of DVFS in cases where context-aware design is inefficient. Our selective replication-based methodology reduces average power by up to 25% with respect to a conventionally optimized DVFS design, achieving average power consumption that is within 1% of ideal, on average.
- We show that optimizing the microarchitecture of a DVFS design has the potential to significantly improve DVFS energy efficiency. We observe that optimizing the microarchitecture reduces average power by up to 18% for a context-aware multi-mode design flow.

6.1 Understanding DVFS Inefficiency

Conventional single-mode CAD flows optimize a design for a single design constraint. Such a design may be inefficient at all operating points except the one for which the hardware was optimized. Consequently, conventional single-mode CAD flows may be inadequate for DVFS-based designs. Our present work points out that conventional multi-mode methodologies may also be inefficient for DVFS-based designs – especially energy-constrained designs. This is because the primary focus of a conventional multi-mode design methodology is to ensure timing closure over multiple, fully-constrained operating modes, not to optimize a design over multiple modes for a specific metric such as energy efficiency. As such, there exists no multi-mode design flow that considers the operating scenario (i.e., both the range of scalability (X) *and* the duty cycle (R)) during optimization. Range of scalability refers to the ratio of maximum and minimum operating frequencies for the design ($X = f_{hi}/f_{lo}$). Duty cycle refers to the fraction of time spent in an operating mode ($R_k = (time\ at\ f_k)/(total\ time)$). Likewise, there exists no multi-mode design flow to optimize a design for minimum energy when the exact optimal constraints (all operating frequencies *and* voltages) are not known at design time.

To quantify the inefficiency of conventional CAD flows, Table 6.1 compares the power consumption and characteristics of several designs, each with different timing constraints.[21] Each design operates at its minimum safe voltage. Table 6.1 demonstrates that the design for which power is minimized depends on the dominant operating frequency. At high frequency, dynamic power (CV^2f) dominates total power, and tightly constrained designs that allow more voltage scaling have lower power consumption. These designs do well in scenarios that favor high performance (high duty cycle (R), low range of scalability (X)). However, these designs also have higher leakage and area due to their topology and cell composition (reduced fanout and increased buffering, higher drive strength cells, more low threshold voltage (LVT) cells, fewer high threshold voltage (HVT) cells). These characteristics reduce efficiency for scenarios that favor low performance (low R, high X).

As the operating frequency is reduced, leakage power begins to dominate total power, and designs that are optimized to reduce voltage are inefficient due to large leakage overheads. In this regime, designs that favor low performance (low R, high X) do better. Thus, a conventional single-mode design may be inefficient for DVFS, since it suffers from either area and leakage overhead or higher voltage at the point for which it was not optimized. Table 6.1 also suggests that even a multi-mode design may be inefficient if it is duty cycle-agnostic, because energy efficiency requires that the design be constrained according to its dominant operating mode.

Figure 6.1 shows the single-constraint netlist that consumes minimum energy for a given operating scenario, confirming that the minimum-energy netlist indeed depends on the amount of time spent in each operating mode (R), the range of scalability between high and low performance (X), and the tightness of the timing constraint. We quantify energy in terms of average power, which accounts for the power consumption and fraction of time spent in each mode: $E = R \times Pwr(f_{hi}) + (1 - R) \times Pwr(f_{lo})$.

Figure 6.1 also shows the average power overhead of the netlists compared to the ideal average power. Ideal average power is calculated using the power consumption of the minimum power implementation for each mode. In general, energy inefficiency is greater when duty cycle is low and range of scalability is high. This can result in significant energy overheads for energy-constrained designs that attempt to reduce energy by aggressively scaling down voltage and frequency and spending more time in a low-power mode.

One reason for inefficiency compared to the ideal case is that the delays of different paths scale differently with voltage, and thus, the set of critical paths changes as voltage scales. For

[21]In total, six netlists have been implemented at $1.0V$ with different setup timing constraints: $NET_i = 1.00ns + i \times 0.05ns$.

Table 6.1: Implementation statistics for the OpenSPARC multiplier.

case	area (um^2)	total power @ 1GHz	total power @ 100MHz	% of LVT cells	Avg. fanout	Avg. cell drive strength
NET_0	60509	37.4mW	0.923mW	28%	2.26	1.30
NET_1	59277	38.1mW	0.882mW	19%	2.28	1.29
NET_2	55412	38.2mW	0.872mW	16%	2.31	1.18
NET_3	52383	38.3mW	0.877mW	14%	2.32	1.08
NET_4	49594	38.5mW	0.858mW	14%	2.35	1.00
NET_5	48944	38.9mW	0.853mW	11%	2.34	0.97

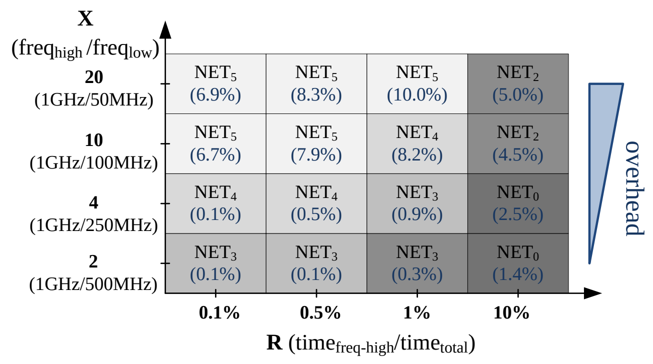


Figure 6.1: Single-constraint netlist with minimum average power in each (R, X) scenario and average power overhead compared to ideal average power (testcase: OpenSPARC multiplier).

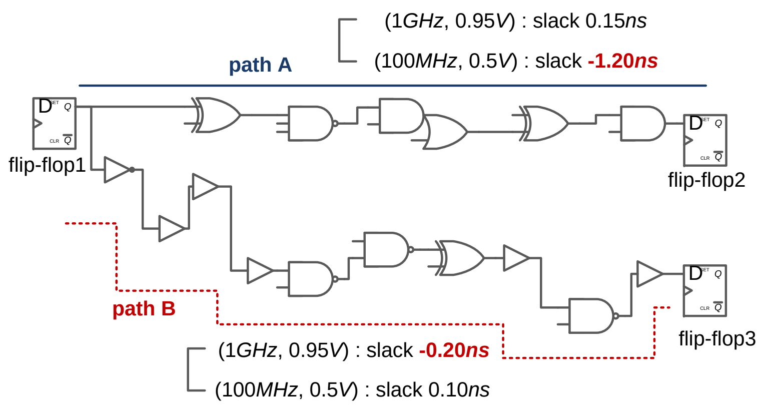

(1GHz, 0.95V) : slack 0.15ns
(100MHz, 0.5V) : slack -1.20ns
path A
flip-flop1
flip-flop2
path B
flip-flop3
(1GHz, 0.95V) : slack -0.20ns
(100MHz, 0.5V) : slack 0.10ns

Figure 6.2: Voltage scaling versus path slack: as voltage scales, the set of critical paths for a design can change.

example, Figure 6.2 shows two paths from the integer multiplier of the OpenSPARC core, along with their timing slack at different minimum-energy operating points. At high performance ($1GHz$, $0.95V$), path B is a critical path, while path A is not. However, at low performance ($100MHz$, $0.5V$), path A is a critical path, while path B is not. The reason for this reversal is that different paths have different delay sensitivities to voltage scaling. In high-performance mode, paths with large logic depths tend to be critical, but such paths are also optimized with cells that tend to be less sensitive to voltage scaling (e.g., cells with lower V_t, shorter interconnect, etc.). Paths that have high V_t cells or long interconnects are *more sensitive* to voltage scaling. While these conditions do not significantly affect delay near nominal voltage, they amplify the increase of path delay when voltage is scaled down. In the example, voltage scaling causes delay to increase faster for path A so that the delay of path A overtakes the delay of path B. This means that path A, which was not critical at high performance, limits voltage scaling at low performance if not properly optimized. Random logic structures may be naturally more susceptible to reversal of critical paths under voltage scaling than regular memory structures, since path characteristics in random logic vary more.

A conventional multi-mode design methodology spends resources to optimize the critical paths in all modes and is therefore over-optimized at any single operating mode. A context-aware multi-mode design methodology increases DVFS efficiency over conventional multi-mode design approaches by accounting for the operating scenario during constraint selection. However, in some scenarios, there can still be significant overhead, especially when the range of scalability is large. For such scenarios, microarchitectural techniques may be useful for increasing DVFS efficiency, as described in Sections 6.2.2 and 6.4.

Another factor that can influence DVFS efficiency is the amount of combinational logic in a design. As we show in Section 6.1 (Table 6.1), combinational logic area can vary substantially as the timing constraint changes, because the entire topology of the optimal implementation can change (re-clustering, buffering, cell sizes and types, etc.). For sequential logic, changes are limited to cell threshold voltages and drive strengths, so there is less variation between tightly and loosely constrained designs. Designs heavy on combinational logic (e.g., ALUs) tend to have worse DVFS efficiency, especially when the range of frequency scaling is large.

6.2 Maximizing DVFS Efficiency

In Section 6.1, we have discussed several reasons why modern DVFS designs may be inefficient. We now propose a context-aware multi-mode design approach that considers operat-

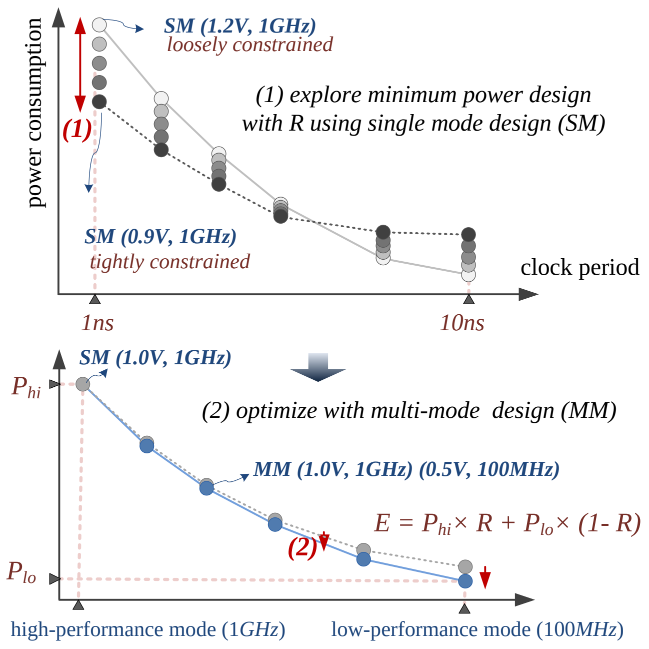

power consumption
SM (1.2V, 1GHz)
loosely constrained
(1) explore minimum power design with R using single mode design (SM)
(1)
SM (0.9V, 1GHz)
tightly constrained
clock period
1ns
10ns
SM (1.0V, 1GHz)
P_{hi}
(2) optimize with multi-mode design (MM)
MM (1.0V, 1GHz) (0.5V, 100MHz)
$E = P_{hi} \times R + P_{lo} \times (1- R)$
(2)
P_{lo}
high-performance mode (1GHz)
low-performance mode (100MHz)

Figure 6.3: Context-aware multi-mode design flow – (1) Algorithm 10: Lines 1–14, (2) Algorithm 10: Lines 15–25.

ing conditions, performance metric, and constraints to maximize energy efficiency over multiple modes of operation.

6.2.1 Context-Aware Multi-Mode Design

Before implementing a multi-mode design, conventional EDA tools require all operating modes to be completely constrained (frequency and voltage). However, as we have shown in Section 6.1, multi-mode designs that are oblivious to the operating scenario can be energy-inefficient. For this reason, we propose a design flow that postpones constraint finalization and uses information about the design and operating scenario to select the most appropriate set of constraints.

Figure 6.3 and Algorithm 10 describe our context-aware multi-mode design flow. The figure shows the total power consumption (y-axis) of a design for each clock period (x-axis). Circles with the same color indicate the same netlist.

The procedure takes as input the high-performance frequency target (f_{hi}), the operational duty cycle (R), the range of scalability (X), and an initial high-performance voltage target from which to start the optimization ($v_{hi}(0)$). This flow implements energy-efficient netlist N_{min}

Algorithm 10 Context-aware multi-mode design.

```
Procedure MultiModeDesign ($f_{hi}$, $v_{hi}(0)$, $R$, $X$)
Input : high-performance frequency $f_{hi}$, initial high-performance voltage $v_{hi}(0)$, duty cycle $R$, frequency ratio $X$
Output : netlist $N_{min}$, high-performance voltage $v_{hi}$, low-performance voltage $v_{lo}$
1: $f_{lo} \leftarrow f_{hi}/X$;  $v_{hi} \leftarrow v_{hi}(0)$;
2: while true do
3:    $E_{min} \leftarrow \infty$;
4:    for $i = 1$ ; $i \leq 2\delta$ ; $i \leftarrow i + 1$ do
5:       $v_i \leftarrow v_{hi} + (i - \delta)v_{step}$;
6:       $N_i \leftarrow SM(v_i, f_{hi})$;
7:       $E_i \leftarrow R \times Pwr(N_i, f_{hi}) + (1 - R) \times Pwr(N_i, f_{lo})$;
8:       if $E_i < E_{min}$ then
9:          $v_{min} \leftarrow v_i$;  $E_{min} \leftarrow E_i$;  $N_{min} \leftarrow N_i$;
10:      end if
11:   end for
12:   if $v_{min} == v_{hi}$ then break;
13:   else $v_{hi} \leftarrow v_{min}$;
14: end while
15: $v_{lo} \leftarrow$ min. safe operating voltage of netlist $N_{min}$ at frequency $f_{lo}$;
16: while true do
17:   $N \leftarrow MM(N_{min}, f_{hi}, v_{hi}, f_{lo}, v_{lo})$;
18:   $E \leftarrow R \times Pwr(N, f_{hi}) + (1 - R) \times Pwr(N, f_{lo})$;
19:   if $E < E_{min}$ then
20:      $E_{min} \leftarrow E$;  $N_{min} \leftarrow N$;  $v_{lo} \leftarrow v_{lo} - v_{step}$;
21:   else
22:      $v_{lo} \leftarrow v_{lo} + v_{step}$;  break;
23:   end if
24: end while
```

according to the scenario (R, X) and determines the voltage constraints that minimize average power across high and low-performance modes (v_{hi}, v_{lo}). The first **while** loop in Algorithm 10 (corresponding to Figure 6.3 (1)) selects a design with minimum average power among single-mode implementations with different timing constraints. The constraint is varied by changing the target voltage (using different timing libraries characterized for each voltage). $SM(v_i, f_{hi})$ (Line 6) represents a physical design implementation that has been optimized for a single design point – voltage v_i and frequency f_{hi}. E is the average power consumption, and $Pwr(N, f)$ is the power consumption of netlist N at frequency f and the minimum safe voltage available through dynamic voltage scaling. δ defines the radius of a local comparison window to avoid selecting a local optimum because of EDA tool noise.

While optimizing constraint specification encompasses a two-dimensional space (v_{hi}, v_{lo}), performing a full evaluation of the constraint space would require too much computational effort. We rely on the following observations to accelerate constraint selection. First, the tightest single constraint dominates the others in determining the amount of area spent to reduce voltage. By considering the energy efficiency of v_{hi} as varied, we can effectively constrain v_{hi} to balance leakage/area reduction and voltage scaling in an operating scenario (R, X), independent of v_{lo}. We hone in on the minimum-energy constraints by varying v_{hi} first because delay sensitivity to voltage is greater at low voltages. Small voltage changes around v_{lo} cause large timing changes and result in large netlist changes. Thus, initially tuning the constraint with the same precision at v_{lo} would require approximately an order of magnitude finer characterization of voltage libraries.

The second **while** loop of Algorithm 10 (corresponding to Figure 6.3 (2)) optimizes the selected design for multiple operating modes. $MM(N,\ f_{hi},\ v_{hi},\ f_{lo},\ v_{lo})$ performs a multi-mode (incremental) optimization for high-performance mode (f_{hi}, v_{hi}) and low-performance mode (f_{lo}, v_{lo}). We continue to reduce v_{lo} (tighten the constraint on low-performance mode) until average power E is minimized. Since multi-mode optimization considers high and low-performance constraints, the second optimization stage optimizes critical paths in all modes and enables further voltage scaling in low-performance mode. To further accelerate constraint selection for v_{hi}, we evaluate the effectiveness and runtime efficiency of using adaptive step sizing for v_{step}. We add an optional pre-processing loop (Algorithm 11) and initially use a large value of v_{step} to select an appropriate range for fine-tuning the constraint. The search space for the pre-processing stage spans a radius of $\gamma \times v_{step}$, centered around an initial estimate of the high-performance constraint (v_{hi}). For example, if $\gamma = 3$ and $v_{step} = 0.05V$, the pre-processing stage will locate an efficient starting point for Algorithm 10 ($v_{hi}(0)$) in the range of $v_{hi} \pm 0.15V$. We observe the shape of the average power versus v_i curve to locate the voltage range that contains the minimum energy design point. Algorithm 11 describes the range selection algorithm that feeds the selected initial constraint value ($v_{hi}(0)$) to Algorithm 10. With the coarse-grained search, our heuristic reduces the number of implementation steps required to fine-tune the constraint, thus reducing runtime. We compare the runtime and average power reduction of our context-aware multi-mode design heuristic with and without adaptive step sizing in Section 6.4.

Although we focus on enhancing DVFS efficiency over two modes that define the boundaries of the range of scalability, the concepts presented here can be generalized for an arbitrary number of modes. In Algorithm 12, we present a generalized, K-mode version of our context-aware multi-mode design heuristic. Rather than a single (R, X) pair, we specify K frequency constraints ($f_1, ..., f_K$) and the fraction of time spent in each mode ($R_{1,2}, ..., R_K$). The K-mode design flow is similar to the previous two-mode flow (Algorithm 10). The formulation of average power (Lines 7, 19) accounts for the contributions of each mode, and the second **while** loop (Lines 17–26) is repeated $K - 1$ times to successively refine the multi-mode design and find the voltage constraints for each mode that minimize average power. We demonstrate the generalized context-aware multi-mode design for the case of three modes in Section 6.4.

Algorithm 11 Constraint selection pre-processing stage.

```
  **Procedure** $RangeSelection(f_{hi}, v_{hi}, R, X)$
  Input : high-performance frequency $f_{hi}$, high-performance voltage $v_{hi}$, duty cycle $R$, frequency ratio $X$
  Output : initial constraint $v_{hi}0$
1: $f_{lo} \leftarrow f_{hi}/X;\ \ E_{min} \leftarrow \infty;$
2: **for** $i = 1\,;\, i \leq 2\gamma\,;\, i \leftarrow i + 1$ **do**
3:     $v_i \leftarrow v_{hi} + (i - \gamma)v_{step};$
4:     $N_i \leftarrow SM(v_i, f_{hi});$
5:     $E_i \leftarrow R \times Pwr(N_i, f_{hi}) + (1 - R) \times Pwr(N_{sm}, f_{lo});$
6:     **if** $E_i < E_{min}$ **then**
7:         $v_{hi}(0) \leftarrow v_i;\ \ E_{min} \leftarrow E_i;$
8:     **end if**
9: **end for**
```

6.2.2 Replication-Based DVFS Design

As noted in Section 6.1, there are scenarios in which multi-mode designs exhibit significant inefficiency with respect to the ideal, due to the area and power overheads of operating in modes for which the design was not specifically optimized. In cases when the overhead is substantial, replication-based design can be used to target each mode individually. We propose a selective replication technique that identifies the modules that cause the most inefficiency and suggests replication for only those modules. Other modules are optimized with context-aware multi-mode design.

Figure 6.4(a) offers a high-level representation of replication and power gating circuitry. A circuit module is replicated, and each replica is optimized for a different performance mode. The control signal, $mode$, selects the active operating mode by power gating the appropriate replica and selecting the correct MUX input. Figure 6.4(b) shows an example of selective replication-based design.

The benefits of replication come at the cost of substantial area overhead due to replicated circuitry, especially when replication is performed at a coarse granularity (i.e., large replicated blocks). Replication overheads due to power gating and MUX circuitry can also be high when replicating at a fine granularity. We evaluate replication decisions at the granularity of RTL

Algorithm 12 Context-aware multi-mode design (K-mode version).

```
Procedure $K\text{-}ModeDesign(v(0), f_1, f_2, ..., f_K, R_1, R_2, ..., R_K)$
Input : initial voltage $v(0)$, frequency constraint $f_i$, duty cycle $R_i$ ($i = 1, 2, ..., K$)
Output : netlist $N_{min}$, voltage $v_i$ ($i = 1, 2, ..., K$)
1: $v_1 \leftarrow v(0)$;
2: while true do
3:   $E_{min} \leftarrow \infty$;
4:   for $i = 1 ; i \leq 2\delta ; i \leftarrow i + 1$ do
5:     $v_i \leftarrow v_1 + (i - \delta)v_{step}$;
6:     $N_i \leftarrow SM(v_i, f_1)$;
7:     $E_i \leftarrow R_1 \times Pwr(N_i, f_1) + R_2 \times Pwr(N_i, f_2) + ... + R_K \times Pwr(N_i, f_K)$;
8:     if $E_i < E_{min}$ then
9:       $v_{min} \leftarrow v_i$;  $E_{min} \leftarrow E_i$;  $N_{min} \leftarrow N_i$;
10:    end if
11:  end for
12:  if $v_{min} == v_1$ then break;
13:  else $v_1 \leftarrow v_{min}$;
14: end while
15: for $j = 2 ; j \leq K ; j \leftarrow j + 1$ do
16:   $v_j \leftarrow$ min. safe operating voltage of netlist $N_{min}$ at frequency $f_j$;
17:   while true do
18:     $N \leftarrow MM(N_{min}, f_1, v_1, ..., f_j, v_j, ..., f_K, v_K)$;
19:     $E_i \leftarrow R_1 \times Pwr(N_i, f_1) + R_2 \times Pwr(N_i, f_2) + ... + R_K \times Pwr(N_i, f_K)$;
20:     if $E < E_{min}$ then
21:       $E_{min} \leftarrow E$;  $N_{min} \leftarrow N$;  $v_j \leftarrow v_j - v_{step}$;
22:     else
23:       $v_j \leftarrow v_j + v_{step}$;
24:       break;
25:     end if
26:   end while
27: end for
```

modules, and we analyze replication at different granularities in this section. Partitioning the design for optimal-granularity replication requires rewriting the RTL and is beyond the scope of this work. We focus on finding the best way to optimize a given DVFS design.

Though replication area overheads can be substantial, most modules do not warrant replication. Modules that are loosely constrained or that have a significant fraction of sequential cells do not require many high-leakage cells or significant topology re-structuring to meet performance constraints. Consequently, they do not impose significant power overheads at scaled frequencies and voltages, and do not necessitate replication.

In addition, implementations of large structures, such as caches, that are optimized for the lowest safe operating voltage or on a separate voltage rail are not significantly affected by scaling. These structures do not necessitate replication. This is beneficial for consistency (no state copying on mode switch), rapid switching between power-gated replicas, and reduction of the area overhead of replication. If replicas share access to a memory structure, the interface and interconnect might require modification to accommodate the multiple replicas, possibly affecting the access time for the structure. If this is the case, access time for the high-performance replica can be minimized at the expense of the low-performance replica. This strategy avoids performance degradation, since timing constraints in the low-performance mode are considerably relaxed compared to the high-performance mode.

To accommodate aggressive voltage scaling, memory structures are typically optimized for the lowest safe voltage or placed on a separate voltage rail. Low-voltage SRAMs [54] [88] can safely operate at voltages below $400mV$, which is the minimum voltage we consider in our study. At $45nm$ and below, split-rail power distribution [12] is common. Except where noted otherwise, our results assume split-rail power distribution with SRAMs on a separate voltage rail.

In order to choose the most energy-efficient partitioning of modules between replication and context-aware multi-mode design, we solve a disjunctively-constrained 0-1 knapsack problem [193] in which the knapsack items are the replication-based and multi-mode module implementations. For each module, one implementation is selected. The profit for an item is the average power savings afforded by selecting a certain optimization strategy for the module, and the weight is the area of the implementation. For replicated modules, average power savings must account for the energy consumed by the active replica, the power-gated replica, MUX logic, and power gating cells. Area must account for both replicas, power gating cells, and MUXes. The capacity of the knapsack is the area budget for the core. Thus, solving the knapsack problem corresponds to choosing the partitioning of replicated and multi-mode modules that maximizes energy savings while fitting within the core's area budget.

Figure 6.5 shows a replication-based DVFS architecture at different levels of abstraction. Subfigure (a) shows that replication for different performance targets can be performed at the core level to create a heterogeneous multi-core architecture in which tasks with different power and performance requirements are scheduled to different cores. Subfigure (b) zooms in on an individual core, showing coarse-grained replication at the module level. The instruction fetch unit (*IFU*) and load store unit (*LSU*) have been replicated. Subfigure (c) zooms further into the floating point frontend unit (*FFU*) module to show fine-grained replication of the *FFU* control unit.

Our replication strategy uses power gating to turn off the inactive replica. The overhead of power gating depends on the number of gating cells required. Equation (6.1) calculates the

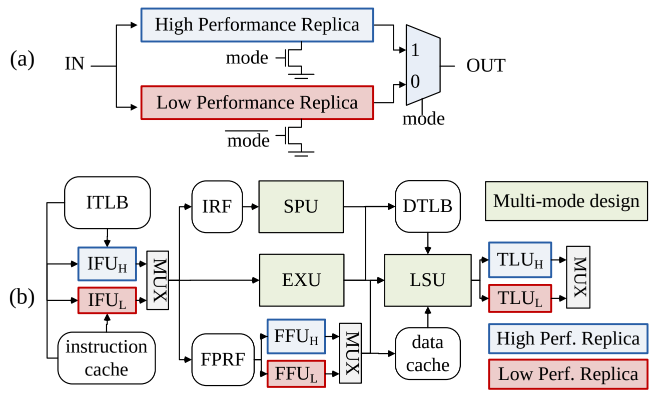


Figure 6.4: (a) A high-level representation of a replication-based design, and (b) selective replication-based processor design for the OpenSPARC T1.

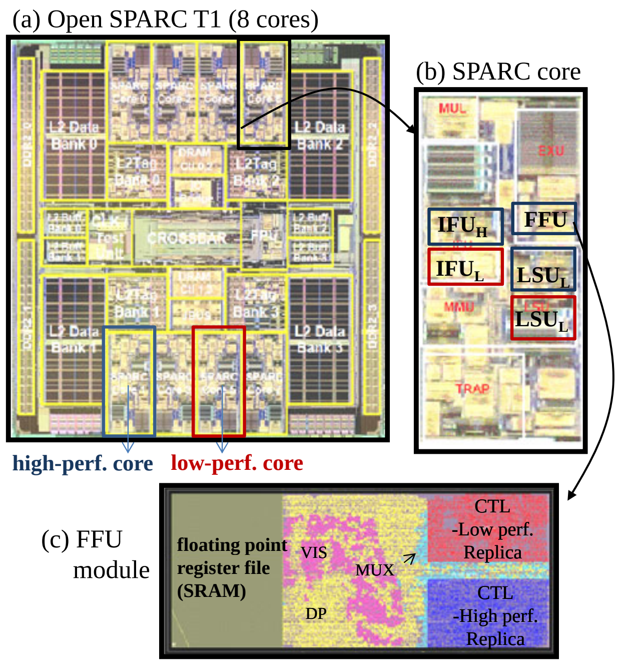


Figure 6.5: Replication-based design at different levels of abstraction can lead to (a) heterogeneous CMP, (b) coarse-grained, and (c) fine-grained selective replication designs.

required number of gating cells for a circuit module with maximum current I_{total}.

$$\delta V_{dd} = I_{total} \times (R_{sw}/N_g) < margin \times V_{dd} \tag{6.1}$$

In this equation, R_{sw} is the resistance of a power gating cell, N_g is the required number of power gating cells, and $margin$ is chosen to be 1% of V_{dd}. We use SPICE simulation to obtain

R_{sw} and gate leakage power at each voltage. We also account for the delay overhead from IR drop. In addition to the overhead of power gates, replication requires that we place MUXes at the ports of a replicated module. This incurs an additional power and area cost, and also adds some latency to the timing paths of the module.

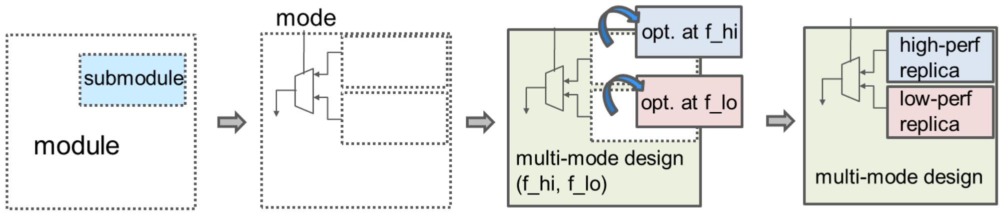


Figure 6.6: Implementation flow for selective replication.

Figure 6.6 shows the implementation flow of our selective replication procedure. The proposed design approach replicates only the modules that maximize energy savings per area, and performs multi-mode design for the rest. We first implement a target module and select submodules to be replicated, using the knapsack optimization. We then change the implemented netlist and make a floorplan for the top module. In the netlist modification, the selected module is re-synthesized for high- and low-performance modes, and MUXes are connected to the output ports of the replicas. Third, we partition each replica from the top module and implement (place and route) them separately. The high-performance replica is optimized at high frequency, and the low-performance replica at low frequency. Finally, we merge each partition and report timing and power for the entire module.

6.3 Methodology

For our experiments, we use all modules (Table 6.2) that comprise the OpenSPARC T1 processor [25]. In Table 6.2, %seq is the percentage of sequential cell area in the module. As we will show, %seq impacts DVFS efficiency. We also evaluate the Issue and Load Store Unit (LSU) stages of the FabScalar [68] processor in different microarchitectural configurations to understand the microarchitectural dependence of DVFS efficiency. We evaluate pipelining and superscalar width in the Issue stage, from 1-wide, 1-deep to 4-wide, 3-deep, and we evaluate the impact of changing the queue sizes and superscalar widths in both the Issue and LSU stages. Since the maximum operating frequency of FabScalar is less than that of OpenSPARC, we consider a smaller range of scalability (X) for FabScalar experiments. Consequently, we increase the range of R for these experiments to ensure adequate coverage of the interesting points in the (R, X) space. Designs are implemented with TSMC 65GP ($65nm$) libraries, characterized by *Cadence Library Characterizer v9.1* [3] for low, nominal, and high threshold voltages over a range of operating voltages. The initial netlists are synthesized with *Synopsys Design Compiler C-2009.06* [26], and layout is performed in *Cadence SoC Encounter v8.1* [6]. As described in Section 6.2.1, we perform implementation at various voltages to find a minimum-energy solution. To mitigate 'inherent noise' in EDA tools [132] [121], we implement each design three times with a small variation in the timing constraint (+0.5ps, -0.5ps and no variation) and choose the design with minimum average power. When evaluating conventional multi-mode design, we choose the voltage constraint for each mode as the voltage that minimizes power for a single-mode design at that mode's operating frequency.

To estimate power consumption, we perform gate-level simulations for one million clock cycles using real workloads. Test vectors for gate-level simulations are gathered from full-system RTL simulations of SPEC benchmarks (*art*, *bzip2*, *equake*, *gzip*, *mcf*, *mesa*, *twolf*) on the OpenSPARC and FabScalar processors. Leakage and dynamic power consumption are reported by *Synopsys PrimeTime-PX c2009.06* [29]. Note that our results assume the same workloads for high and low-performance modes, whereas the activity characteristics may vary significantly between a workload that runs at high frequency and one that runs at low frequency. Though this may be the case, the duty cycle (R) can act as a catch-all to adjust the weighting between the two modes during optimization, not only in terms of time spent in each mode, as originally described, but also to account for factors such as disparity in average circuit activity.

Table 6.2: Target modules from OpenSPARC T1.

module	stage	description	%seq	area (um^2)
EXU	EX	integer execution	20%	81902
FFU	EX	floating point front-end	30%	28584
IFU	F/D	instruction fetch & decode	40%	121458
LSU	MEM/WB	load store	45%	125725
MUL	EX	integer multiplier	15%	61978
SPU	EX	stream processing	45%	33580
TLU	MEM/WB	trap logic	47%	111902

6.4 Experimental Results

In this section, we analyze our heuristic design choices, quantitatively demonstrate the benefits of context-aware multi-mode and selective replication-based designs, and explore the implications of microarchitecture on the energy efficiency of DVFS design.

6.4.1 Heuristic Design

We begin with an evaluation of the design choices that led to our chosen heuristic implementation for multi-mode design. Table 6.3 compares the average power and runtime efficiency of heuristic implementations that vary in voltage step size (v_{step}) and whether or not the pre-processing stage (Algorithm 11 in Section 6.2.1) is used. The pre-processing stage reduces runtime significantly by reducing the number of single-mode implementations (row (a) in Table 6.3) required for minimum-energy constraint selection. For a step size of $0.01V$, pre-processing reduces the constraint search space by 48% while achieving the same energy efficiency. This reduces runtime by 43%.

Increasing the size of v_{step} also reduces runtime, but with a potential cost in energy efficiency, due to the coarser granularity of constraint tuning. Without pre-processing, doubling v_{step} reduces runtime by 24% and increases average power by 3.7%. When the pre-processing stage is also used, doubling v_{step} reduces runtime by 15% and increases average power by 6.1%. Thus, the marginal benefit of increasing the step size is higher when pre-processing is not used. Based on our analysis, we apply pre-processing with $\gamma = 2$, $v_{hi} = 0.80V$, and $v_{step} = 0.05V$. A step size of $v_{step} = 0.01V$ is used for the main heuristic procedure.

Table 6.3: Analysis of design choices for multi-mode design heuristic implementation, for a submodule of *LSU*.

	without pre-processing		with pre-processing	
step	$0.01V$	$0.02V$	$0.01V$	$0.02V$
(a) # of implementations	23	18	12	11
(b) # of optimizations	7	4	7	4
(c) runtime (*sec*)	4199	3196	2395	2048
(d) average power (*W*)	3.26E-05	3.38E-05	3.26E-05	3.46E-05

6.4.2 Context-Aware Multi-Mode Design

To gauge the effectiveness of context-aware DVFS design, we first present average power results for the modules of the OpenSPARC processor. Table 6.4 shows the average power reduction achieved by context-aware multi-mode and replication-based designs,[22] compared to conventional multi-mode design (high-performance clock frequency $f_{hi} = 1GHz$). Table 6.5 shows average power overhead for context-aware DVFS with respect to the ideal. The "total" rows represent processor-wide results.

From Table 6.4, we observe that context-aware multi-mode design improves DVFS efficiency by up to 19.5% as compared to conventional multi-mode design. In general, benefits are higher for low %seq (e.g., *EXU*, *MUL*) and low R, as is typical in energy-constrained designs. Processors that have a higher fraction of area devoted to execution units (e.g., DSP processors) or spend more time in a low-power mode (e.g., mobile devices), should see more benefits from context-aware design. For combinational logic, area and leakage can change significantly with different constraints, due to topological adaptations. Therefore, targeted designs for different performance constraints can differ substantially. As such, any multi-mode design is necessarily inefficient at one or more performance targets, especially when the range of scalability (X) is high.

To analyze context-aware multi-mode design for more than two modes, we have evaluated designs with more modes. Table 6.6 compares three-mode context-aware designs (f_1, f_2, f_3 $= 1GHz$, $500MHz$, $100MHz$) against high-performance targeted designs ($f = 1GHz$) in terms of power consumption in each operating mode. We have chosen the duty cycles (R_1, R_2 and R_3) such that each mode contributes roughly an equivalent fraction of average power. We also evaluate the impact of split-rail design, where SRAMs are on a separate voltage rail versus single-rail design, where the minimum operating voltage of the chip is limited to $0.6V$. Although the minimum operating voltage of typical SRAMs is around $0.7V$, some special-purpose SRAMs can operate well at lower voltages with design overheads; $0.6V$ represents a voltage within the operating range of many low-power SRAMs (e.g., [54] [88] [194]). On average, context-aware multi-mode design reduces average power by 12.4% for a design with low %seq (*MUL*) and 10.3% for a design with high %seq. For the single-rail case, benefits for *MUL* decrease only slightly, since limiting the minimum voltage reduces the disparity between optimal high-performance and low-performance designs, such that the optimal design point shifts more toward high-performance

[22]In the replication-based results, power, area, and delay overheads from power gating cells, MUXes, and IR drop have been included. We do not consider wakeup energy.

Table 6.4: Average power reduction for context-aware multi-mode and replication-based designs against conventional multi-mode design (higher is better).

		context-aware multi-mode			replication-based design		
test case	X	R = 0.5%	R = 1%	R = 5%	R = 0.5%	R = 1%	R = 5%
EXU	5	1.5%	1.4%	2.3%	8.7%	8.3%	6.6%
	10	4.1%	4.9%	4.4%	9.6%	9.1%	7.4%
	20	13.0%	9.4%	2.6%	15.6%	12.1%	4.1%
FFU	5	3.8%	4.3%	6.9%	5.8%	6.2%	8.1%
	10	4.6%	4.9%	3.3%	6.1%	5.5%	3.4%
	20	4.0%	2.1%	1.8%	3.6%	2.1%	-1.7%
IFU	5	2.7%	2.7%	9.0%	0.4%	1.1%	4.1%
	10	2.7%	3.3%	7.1%	3.6%	3.5%	3.4%
	20	1.5%	2.4%	2.6%	0.3%	0.8%	1.8%
LSU	5	4.4%	5.4%	9.8%	6.2%	7.3%	12.1%
	10	1.6%	1.8%	9.7%	4.8%	6.2%	10.2%
	20	3.6%	2.3%	5.4%	5.5%	6.1%	7.2%
MUL	5	19.5%	12.4%	8.7%	25.4%	24.0%	16.6%
	10	4.0%	2.2%	5.4%	15.1%	14.2%	11.5%
	20	16.2%	6.5%	5.7%	23.1%	19.7%	11.5%
SPU	5	4.0%	3.9%	4.7%	1.4%	1.7%	3.0%
	10	2.6%	3.0%	4.4%	4.2%	3.9%	2.9%
	20	5.9%	4.6%	1.8%	8.0%	5.9%	1.0%
TLU	5	5.3%	5.6%	7.0%	3.4%	3.9%	6.3%
	10	7.0%	7.8%	3.6%	3.9%	4.7%	7.1%
	20	6.5%	5.1%	3.6%	8.8%	8.6%	8.2%

mode. So, although power reduction decreases for low-performance mode (mode 3), it increases for high-performance mode (mode 1). Average power reduction decreases more significantly for *SPU*. Since *SPU* has high %seq, the difference between high-performance and low-performance designs is not as significant. Thus, the primary source of power savings is power reduction in low-performance mode (mode 3), and limiting the minimum voltage significantly cuts into those savings; in the single-rail case, *SPU* shows only 3.66% average power reduction in mode 3, due to the minimum voltage limitation.

To demonstrate the potential benefit of context-aware design in the three-mode case, we implement context-aware multi-mode designs targeting seven different scenarios. Each scenario has different duty cycles (R), as described in Table 6.7. The optimized netlist for scenario S_i is net_i. Table 6.8 shows the average power consumption (normalized to the power of net_i for S_i) of

Table 6.5: Average power overhead for context-aware multi-mode and replication-based designs as compared to the ideal case (lower is better).

		context-aware multi-mode			replication-based design		
test case	X	R = 0.5%	R = 1%	R = 5%	R = 0.5%	R = 1%	R = 5%
EXU	5	8.1%	8.0%	5.8%	0.2%	0.3%	1.0%
	10	6.5%	5.3%	4.7%	0.4%	0.6%	1.5%
	20	3.8%	4.2%	3.3%	0.7%	1.0%	1.7%
FFU	5	2.4%	2.5%	2.6%	0.2%	0.4%	1.3%
	10	2.1%	1.4%	2.0%	0.4%	0.8%	1.9%
	20	3.9%	3.9%	3.4%	4.3%	4.0%	3.3%
IFU	5	1.5%	2.1%	1.7%	3.9%	3.9%	3.6%
	10	1.4%	1.2%	1.8%	0.6%	1.0%	2.2%
	20	0.2%	0.1%	1.7%	1.0%	1.5%	2.6%
LSU	5	2.1%	2.4%	3.5%	0.2%	0.3%	0.9%
	10	7.1%	5.3%	1.9%	0.4%	0.6%	1.3%
	20	2.7%	5.0%	3.6%	0.6%	0.9%	1.6%
MUL	5	8.1%	15.7%	10.7%	0.2%	0.4%	1.1%
	10	13.5%	14.8%	8.6%	0.4%	0.7%	1.6%
	20	9.8%	17.7%	8.6%	0.8%	1.1%	2.0%
SPU	5	1.8%	2.2%	3.1%	4.6%	4.6%	4.9%
	10	2.5%	2.4%	2.0%	0.9%	1.5%	3.6%
	20	7.7%	6.8%	4.6%	5.3%	5.3%	5.4%
TLU	5	2.0%	1.8%	0.8%	1.6%	1.8%	2.0%
	10	3.2%	3.3%	3.8%	2.1%	2.4%	2.7%
	20	2.5%	3.8%	5.0%	1.8%	1.1%	1.4%

each netlist in each scenario. The context-aware design targeting a specific scenario minimizes average power for that scenario. Since our heuristic pinpoints the minimum energy constraints for a design by performing small explorations in the constraint space, the optimization result can be improved by searching more design points or searching at a finer granularity (smaller step size, V_{step}). Our experiments use $V_{step} = 0.01V$, and some netlists show only small benefits over the netlist optimized for a different scenario. For example, net_2 and net_3 are nearly identical, since the two scenarios are similar (mode 1 is dominant for S_2 and S_3). Similarly, net_4 and net_6 have similar average power in all scenarios, because mode 2 is dominant for S_4 and S_6.

Figure 6.7 quantifies runtime and average power savings for context-aware designs as the number of modes increases from one to four. Increasing the number of modes causes runtime to increase approximately linearly, since the complexity of Algorithm 12 is linear with respect

Table 6.6: Comparison of average power consumption between single-mode design and context-aware multi-mode design for three modes.

mode	freq.	R	single-mode (mode 1)		context-aware multi-mode		
			voltage	total power	voltage	total power	reduction
	(*GHz*)		(*V*)	(*W*)	(*V*)	(*W*)	(%)
module: *MUL*							
mode 1	1.0	0.02	0.92	2.950E-2	0.99	3.181E-2	-7.83%
mode 2	0.5	0.28	0.69	8.892E-3	0.69	7.755E-3	12.78%
mode 3	0.1	0.70	0.48	1.140E-3	0.46	8.405E-4	24.55%
average				3.877E-3		3.396E-3	12.41%
module: *SPU*							
mode 1	1.0	0.02	0.86	1.050E-2	0.87	1.108E-2	-5.47%
mode 2	0.5	0.28	0.67	3.231E-3	0.63	2.875E-4	11.03%
mode 3	0.1	0.70	0.47	3.731E-4	0.41	2.968E-4	20.44%
average				1.376E-3		1.234E-4	10.30%

Table 6.7: Different scenarios for three-mode implementation.

scenario	energy consumption			duty cycle (R)		
	mode 1	mode 2	mode 3	mode 1	mode 2	mode 3
S_1	100%	0%	0%	1.000	0.000	0.000
S_2	65%	30%	5%	0.100	0.660	0.240
S_3	65%	5%	30%	0.060	0.070	0.870
S_4	30%	65%	5%	0.020	0.830	0.150
S_5	30%	5%	65%	0.020	0.030	0.950
S_6	5%	65%	30%	0.002	0.498	0.500
S_7	5%	30%	65%	0.002	0.172	0.826

to the number of modes. As the number of modes increases, average power decreases, since context-aware design (i) enables a lower voltage for a given mode by optimizing the critical paths in each mode and (ii) reduces area and leakage for the design by accounting for the duty cycles and range of scalability.

6.4.3 Selective Replication-Based Design

Even though context-aware multi-mode design has significant benefits over conventional multi-mode design, when %seq is low, it can still exhibit considerable inefficiency (up to 17.7%) with respect to the ideal. This is because the ideal area and power consumption of combinational

Table 6.8: Average power consumption in each scenario (testcase: *MUL*).

scenario	net_1	net_2	net_3	net_4	net_5	net_6	net_7
S_1	1.000	1.043	1.043	1.092	1.084	1.126	1.136
S_2	1.088	1.000	1.000	1.009	1.036	1.021	1.048
S_3	1.067	1.000	1.000	1.033	1.005	1.049	1.026
S_4	1.154	1.009	1.009	1.000	1.046	1.003	1.042
S_5	1.171	1.031	1.031	1.064	1.000	1.070	1.003
S_6	1.178	1.010	1.010	1.001	1.035	1.000	1.026
S_7	1.215	1.023	1.023	1.027	1.013	1.024	1.000

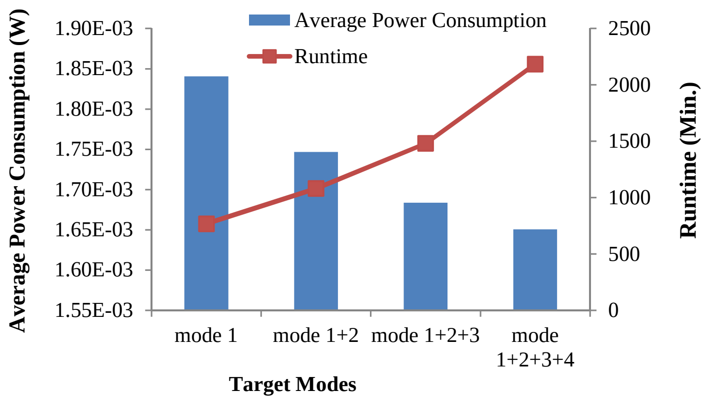
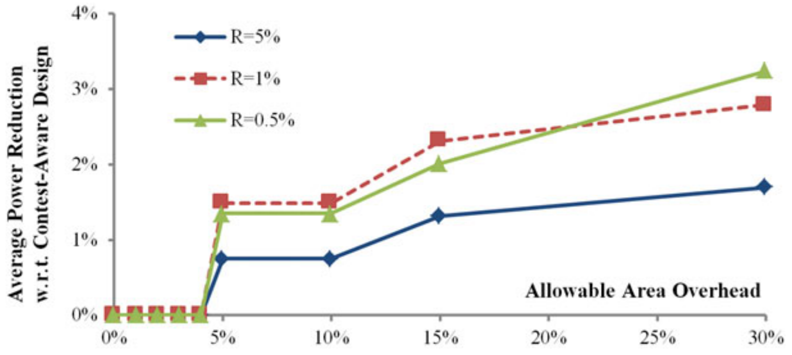
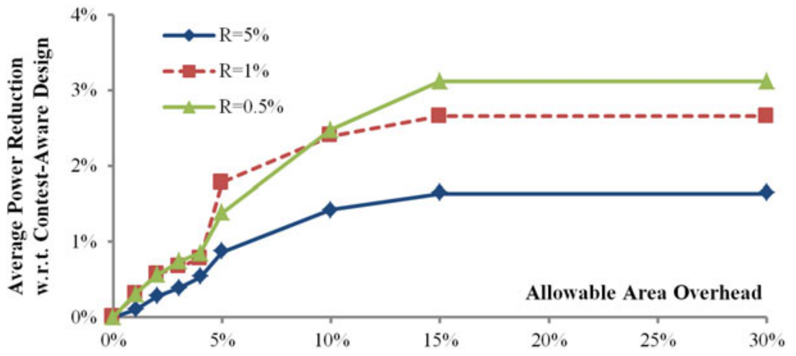


Figure 6.7: Results for context-aware implementations with different numbers of modes (testcase: *MUL*). Frequencies for each mode are $1GHz$, $500MHz$, $250MHz$ and $100MHz$, and duty cycle values (R_k) are selected so that each mode consumes roughly an equal share of the total power.

logic change considerably with the timing constraint. Fortunately, selective replication minimizes average power in these cases, coming within 1% of the ideal average power, on average, at the expense of area overhead. This suggests that an appropriate combination of context-aware and replication-based approaches may be nearly ideal for maximizing DVFS efficiency. Note that replication does not achieve significant benefits over multi-mode design for modules such as *SPU* and *IFU*. Most of the benefits of replication come from reducing leakage and area overheads from the combinational logic. However, these modules have high %seq, so that the extent of achievable benefits is considerably reduced. In fact, in a few scenarios, context-aware multi-mode design even does better than replication, due to the timing and power overheads of replication. In many cases, the significant area overheads of module replication (94% on average) are not justified. For a given area budget, we use our knapsack-based selective replication technique



(a) Coarse-grained selective replication.



(b) Fine-grained selective replication.

Figure 6.8: Selective replication achieves additional (average) power reduction over context-aware multi-mode design for the OpenSPARC processor, coming within 1% of the ideal average power for only 5% area overhead.

to identify the processor modules to replicate, such that the average power of the processor is minimized.

Figure 6.8 shows average power reduction achieved by selective replication for the OpenSPARC processor with a given area budget. Note that the benefits shown in Figure 6.8 are in addition to those achieved by the best context-aware multi-mode design for the particular scenario. The results demonstrate that in most cases, context-aware multi-mode design achieves close to ideal average power, and that replicating only a small number of modules closes the gap between context-aware design and ideal average power. The final result with both context-aware and selective replication-based designs is a DVFS processor with *average power that is within 1% of ideal*, on average, with only 5% area overhead. Note that increasing the area budget beyond 5% yields only minimal incremental benefits, confirming that only a few modules need to be replicated.

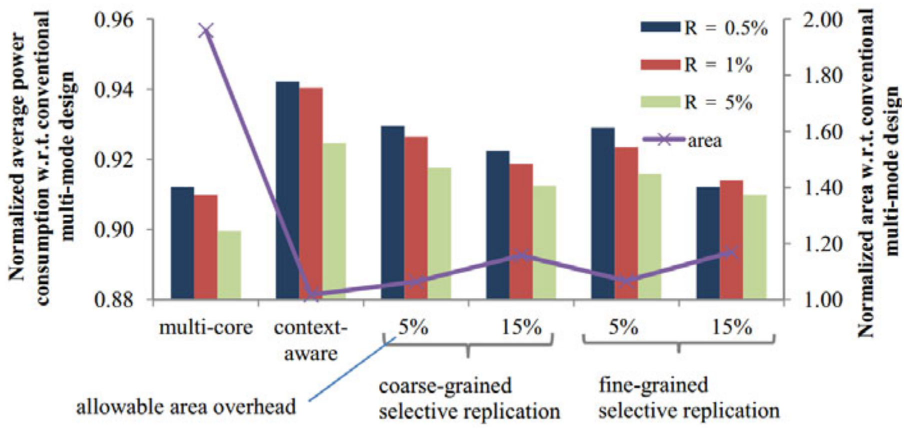


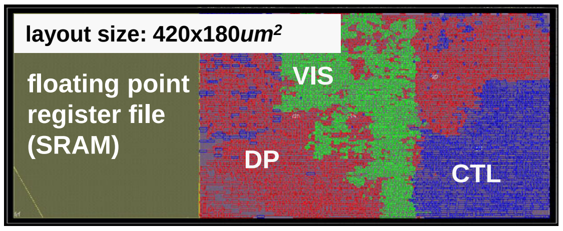
Figure 6.9: Average power consumption and area comparison among multi-core, context-aware, and selective replication-based design.

Figure 6.8 also compares replication at the module granularity (Table 6.2 modules) to replication at the fine granularity of leaf modules. Typically, replication at the finer granularity achieves larger average power reduction for a given area budget, as inefficient submodules can be targeted more precisely without replicating the entire encompassing module. Also, fine-grain replication can provide average power reduction even for small allowable area overheads, whereas coarse-grain replication must overcome an initial area hurdle before any module can be replicated. One disadvantage of fine-grain replication is increased overhead for power gating and MUX logic, which can prevent several small leaf modules from achieving replication benefits.

As discussed in Section 2.2.2, a heterogeneous multi-core processor can also target multiple power and performance points. In Figure 6.9, we compare power reduction and area overhead among heterogeneous multi-core, context-aware, and selective replication-based designs. All designs target high-performance and low-power modes. Selective-replication-based design achieves energy efficiency benefits within 2% of the heterogeneous multi-core with a significantly smaller area overhead.

While area, power, and timing overheads of replication are carefully modeled in the above studies, replication also affects the physical layout of the design, particularly in the neighborhood of the replicated module. Figure 6.10 shows layouts for conventional multi-mode design and selective replication-based design for the *FFU* module. In the selective replication-based implementation, the CTL module has been replicated, affording 12% average power savings with 14% area overhead. This result demonstrates the feasibility of the proposed replication-based approach described in Figure 6.6, in spite of the potential layout complications.

(a)


layout size: 420x180um^2
floating point register file (SRAM)
VIS
DP
CTL

(b)

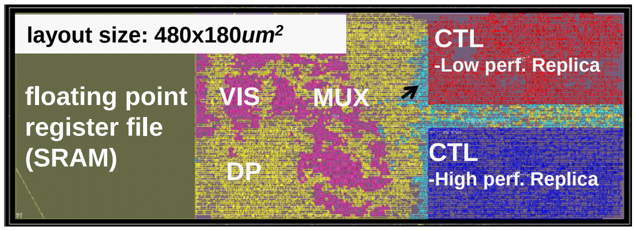

layout size: 480x180um^2
floating point register file (SRAM)
VIS
MUX
DP
CTL -Low perf. Replica
CTL -High perf. Replica

Figure 6.10: Layout of *FFU* (Floating point Front-end Unit) module (a) without replication and (b) with replication.

6.4.4 Variation Analysis

In general, it may not be possible to estimate duty cycle (R) precisely for all users of a particular DVFS product. Figure 6.11 shows how average power savings may be affected when R_{hi} differs from the average value of R_{hi} targeted at design time. In these experiments, average power savings are reduced by at most 3% in the case where target R_{hi} is maximum (5%) and duty cycle is 50 times lower. This is because when actual duty cycle is lower than the target, the resulting design is over-optimized for the user scenario and exhibits area and leakage overhead. Note that duty cycle variation does not affect the efficiency of a selective replication-based design.

We also evaluate the impact of variations, including systematic and random within-die (WID) variations on our DVFS designs. Since our DVFS designs scale between fixed operating points, and these operating points must be defined based on worst-case corners, variations do not affect the voltage or frequency at which our designs operate. Variations can, however, affect power consumption, primarily in terms of leakage power (e.g., due to changes in threshold voltage and gate length). While we present results for 65nm process technology, it is well known that in recent and future technology nodes (e.g., 32nm, 22nm), static power constitutes a larger fraction of total power, and can vary more substantially (e.g., in response to temperature changes). Analyzing the impact of leakage variations may be particularly relevant for context-aware design, where design optimization accounts for the relative contribution of leakage in different operating modes. The impact of variations may be felt most prominently for low R_{hi}, when leakage accounts for a larger fraction of total power. Following the methodology of [55], we model WID variations, including lithography-induced systematic WID variations. We use standard deviation (σ) of leakage for each standard cell following [55] and repeat power analysis for 1000 different variation maps, recording the average power in each trial. Note that to model lithography-induced, pattern-dependent WID variations, instances of the same standard cell have the same systematic WID variations for a given Monte Carlo trial. Figure 6.12 compares average power consumption for the processor observed during variation analysis for different multi-mode design styles. Error bars show the min and max average power observed during Monte Carlo analysis. For designs with low R_{hi}, where leakage power variations impact total power more significantly, we see that variations impact average power savings by less than 2%.

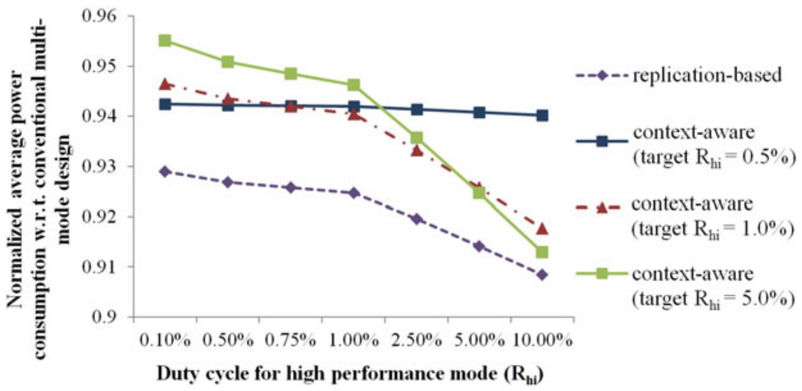


Figure 6.11: Normalized average power consumption when the duty cycle (R_{hi}) varies from the value targeted at design time (target R_{hi}).

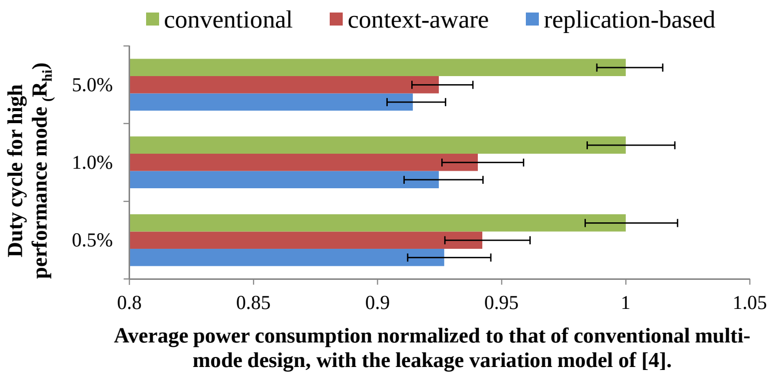


Figure 6.12: Normalized average power consumption with leakage variations (X = 5, $R_{hi} \in \{0.5\%, 1\%, 5\%\}$).

6.4.5 Impact of Microarchitecture on DVFS Efficiency

In Section 6.1, and in the results above (Table 6.4), we observe that the benefits of context-aware multi-mode design depend on factors such as the relative amount of sequential logic in a design and the tightness of the timing constraints. Since microarchitecture can influence these factors, DVFS energy efficiency can potentially be enhanced through microarchitectural adaptations. To gauge the potential impact of microarchitecture on DVFS efficiency, we evaluate the effectiveness of context-aware multi-mode design for different microarchitectural adaptations, namely, pipeline depth, superscalar width, and queue sizes in the Issue and LSU stages.

Pipeline Depth and Logic Complexity. To evaluate the effects of changing the pipeline depth and superscalar width, we use FabScalar to generate six versions of the Issue stage of a superscalar processor, with pipeline depths $\in \{1,\ 3\}$ and superscalar widths $\in \{1,\ 2,\ 4\}$. Table 6.9 shows how increasing the pipeline depth from 1 to 3 affects the average power of various context-aware multi-mode designs with different superscalar widths in different scenarios.

Table 6.9: Percent reduction in average power from increasing pipeline depth from 1 to 3 for different scenarios and superscalar widths.

width	X	R = 0.5%	R = 1%	R = 10%	R = 50%
	2	1.6	1.5	3.3	8.1
1	4	0.8	0.4	2.8	10.1
	10	5.8	4.0	4.6	11.3
	2	6.9	6.8	6.3	5.1
2	4	0.4	1.1	7.1	6.1
	10	1.0	1.6	7.0	5.5
	2	4.2	4.3	5.1	11.8
4	4	5.3	5.1	4.6	13.0
	10	7.7	7.4	8.4	14.0

Table 6.9 shows that deeper pipelining in a context-aware multi-mode design improves energy efficiency more for designs with higher R, X, and logic complexity (e.g., superscalar width). Higher R means that high-performance mode is weighted more in the energy metric. Deeper pipelining allows context-aware multi-mode design to reduce voltage significantly (10%) at high frequency, where the impact of voltage (dynamic power) reduction is more pronounced, providing benefits for designs with high R. At low frequency and voltage, delay sensitivity to voltage is much higher, and pipelining does not allow our DVFS design flow to achieve signif-

icant voltage reduction. Thus, the energy efficiency of context-aware multi-mode design does not improve much with pipelining in scenarios with low R.

Designs that have more complexity (e.g., higher superscalar width) or shallower pipelines have deeper logic depth and higher fanout. For such designs, the difference between high and low performance targeted netlists is considerable, especially for larger X, since paths with deep logic, high fanouts, and tight timing constraints require larger, more leaky cells and more buffering. to meet tight timing constraints. Deeper pipelining provides more benefits in such scenarios, as it relaxes tight constraints, allowing a multi-mode design to meet timing in multiple modes with considerably less overhead, especially when the range of scalability (X) is large.

In typical circuits, such as the FabScalar testcases discussed above, perfect pipelining is not possible due to logic complexity. To investigate the potential influence of pipelining on DVFS efficiency, we create a generic test circuit that can be subdivided cleanly. Figure 6.13 shows the different implementations of the pipelining test circuit. The basic, single-stage design (a) is created by cascading four 8-bit multipliers end-to-end between a pair of latches. Two-stage (b) and four-stage (c) versions of the circuit are created by placing pipeline latches at the appropriate junctions between multipliers.

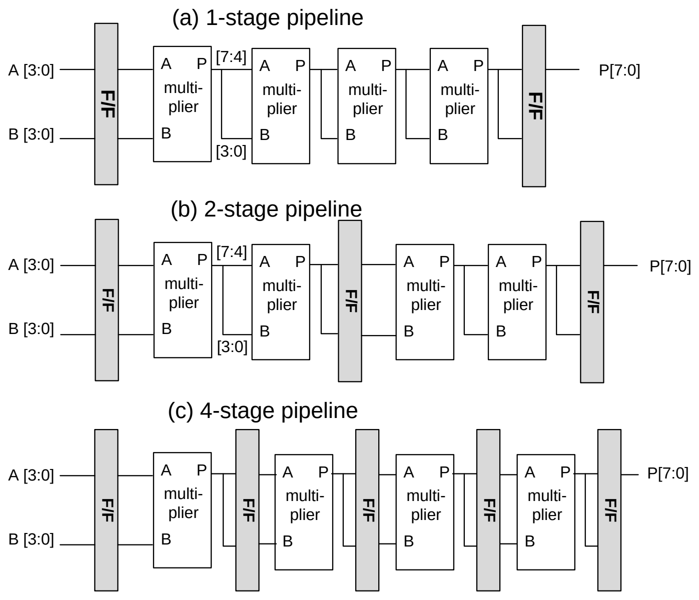


Figure 6.13: Pipeline test circuit composed of cascaded multiplier blocks for pipeline depths of 1, 2, and 4.

Table 6.10 shows the DVFS power versus performance tradeoff for the pipelined multiplier testcase. Interestingly, each pipeline implementation is the minimum energy design over

Table 6.10: Power comparison for DVFS in each pipelined multiplier implementation.

pipeline	operating frequency (*MHz*)	operating voltage (*V*)	total power (*W*)	leakage power (%)
1-stage	1000	N/A	N/A	N/A
	500	N/A	N/A	N/A
	200	0.96	4.53E-05	4.9
	100	0.74	1.37E-05	7.7
	50	0.62	5.04E-06	13.5
2-stage	1000	1.16	5.15E-04	3.4
	500	0.76	1.09E-04	4.7
	200	0.58	2.40E-05	9.3
	100	0.51	1.06E-05	15.3
	50	0.45	5.46E-06	21.8
4-stage	1000	0.84	3.52E-04	3.0
	500	0.65	1.07E-04	5.2
	200	0.53	2.99E-05	11.3
	100	0.47	1.29E-05	19.9
	50	0.41	6.10E-06	31.0

some portion of the frequency scaling range. At high frequency, the deepest pipeline (4-stage) consumes the least average power, since a large fraction of total power is dynamic power. While the 1-stage design cannot operate at 1*GHz* for any available voltage, the deeper pipelining of the 4-stage design enables a voltage reduction of 28% with respect to the 2-stage design, significantly reducing dynamic power consumption. The 4-stage design remains the minimum power design as frequency is scaled down to 500*MHz*. At the same time, the voltage differential between the 4-stage and 2-stage designs shrinks due to increasing delay sensitivity to voltage. Also, as frequency is scaled down, leakage power accounts for a larger fraction of total power. Leakage increases faster for deeper pipelines due to increased latch area. By 200*MHz*, the voltage advantage of the 4-stage design is only 9%, and reduced area and leakage due to fewer pipeline latches make the 2-stage design more efficient. Scaling down to 50*MHz*, the 1-stage design takes over as the most energy-efficient.

As demonstrated in Table 6.9 and Table 6.10, DVFS efficiency depends on microarchitectural parameters, including pipeline depth. Thus, pipeline depths for the modules in a selective replication-based DVFS design should be chosen appropriately to minimize average power. For a design that weights high-performance mode more heavily (high R), a deeper pipeline is beneficial, since it allows more voltage scaling for reduced dynamic power. For a design that weights low performance more heavily (low R), a shallower pipeline is beneficial for area and leakage reduction. Consequently, duty cycle (R) plays a strong role in determining the optimal pipeline depth of a multi-mode design. As R increases, optimal pipeline depth also increases. For a replicated module in a selective replication-based design, high and low-performance replicas may have different energy-optimal pipeline depths. Thus, module replicas, though identical in functionality, may not have identical implementations. In addition to optimizing the design level implementation of each replica for the performance target, the microarchitecture – including pipeline depth – should also be optimized. For the multiplier at $R = 1\%$, $X = 10$, a replication-based design that optimizes the pipeline depths of each replica has 14% lower average power than a replication-based design without microarchitectural optimization. Average power reduction is 9.5% with respect to a context-aware multi-mode design. Figure 6.14 shows normalized average power consumption for each pipelined multiplier implementation, demonstrating that the optimal pipeline depth for a DVFS design depends on the dominant performance mode.

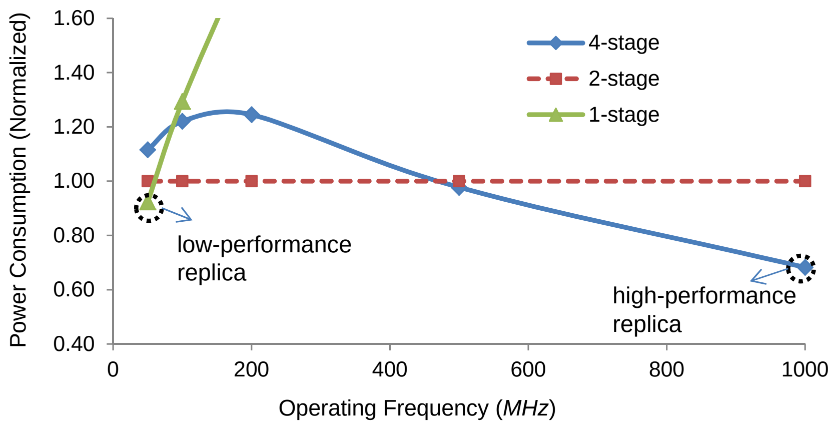


Figure 6.14: Power consumption of each pipelined multiplier (normalized to the 2-stage implementation). The minimum-energy implementation depends on the dominant performance mode.

Microarchitectural Structure Sizing. Besides pipeline depth, we also investigate the effect on DVFS efficiency of changing the sizes of microarchitectural structures in the processor. We evaluate the Issue stage of the FabScalar processor for issue queue (IQ) lengths $\in \{16, 32\}$. We also evaluate the Load Store Unit (*LSU*) of the processor for load and store queue lengths $\in \{8, 16\}$. Note that the total length of the load store queue (LSQ) is the length of the load queue plus the length of the store queue $\{16, 32\}$. In both cases, we observe the effect of changing the superscalar width for widths $\in \{1,\ 2,\ 4\}$. Superscalar width and IQ and LSQ lengths have a significant impact on processor complexity [187] and, consequently, affect several design characteristics that influence DVFS efficiency, as outlined in Section 6.1.

In order to isolate the effect of these microarchitectural optimizations on DVFS efficiency, we compare the average power consumption of the context-aware multi-mode design for each scenario against the average power of an ideal design. Ideal average power in each scenario is computed using the power of the minimum power targeted design in each mode.

Table 6.11: Percent average power overhead with respect to ideal for the LSU stage for different scenarios, LSQ lengths, and superscalar widths.

		LSQ length = 16				LSQ length = 32			
width	X	R = 0.5%	R = 1%	R = 10%	R = 50%	R = 0.5%	R = 1%	R = 10%	R = 50%
1	2	0.3	0.7	4.0	7.9	6.0	6.0	6.9	0.7
	4	2.8	3.7	11.9	9.0	6.0	6.1	7.4	2.6
	10	7.1	9.7	14.8	11.1	8.8	6.8	4.9	2.4
2	2	3.7	4.4	8.7	13.7	7.5	7.4	5.6	2.6
	4	5.4	6.8	20.2	31.7	7.3	7.2	2.6	0.7
	10	10.0	13.1	34.9	39.4	10.2	10.0	1.4	0.3
4	2	6.0	6.5	13.7	1.9	1.0	1.0	2.5	0.7
	4	8.4	10.4	19.5	1.3	8.0	8.1	2.5	1.1
	10	12.2	16.7	28.0	0.7	10.1	10.5	3.8	0.5

Table 6.12: Percent average power overhead with respect to ideal for the Issue stage for different scenarios, IQ lengths, and superscalar widths.

		IQ length = 16				IQ length = 32			
width	X	R = 0.5%	R = 1%	R = 10%	R = 50%	R = 0.5%	R = 1%	R = 10%	R = 50%
1	2	1.0	1.2	4.0	9.5	1.4	1.7	4.8	3.7
	4	8.4	8.8	7.4	2.2	4.7	5.2	10.2	2.3
	10	6.1	7.3	11.7	3.9	13.2	13.6	13.5	2.1
2	2	1.3	1.5	3.7	3.8	0.2	0.4	3.2	0.3
	4	6.3	6.8	10.6	2.4	11.0	11.2	4.9	1.3
	10	6.7	7.6	6.7	1.1	16.1	18.3	7.0	1.2
4	2	0.1	0.2	2.4	0.1	0.1	0.2	1.6	0.2
	4	11.3	10.4	8.5	1.9	5.8	5.9	7.3	1.7
	10	14.5	14.0	6.2	1.5	11.4	10.9	6.0	1.0

Table 6.11 shows the average power overhead with respect to ideal average power in different scenarios for the LSU stage with different LSQ lengths and superscalar widths. Averaged over all scenarios and superscalar widths, the larger LSQ has 5% overhead with respect to ideal, compared to 11% for the smaller LSQ. This is partially because the LSU has a large %seq (40% – 50%), and increasing the size of the LSQ further increases %seq. Due to this and the increased logic complexity incurred by the larger LSQ, the difference between high and low-performance designs is less for the larger LSQ. Thus, DVFS energy efficiency does not suffer considerably when the LSQ size increases. With the smaller LSQ, there is more variation between targeted designs for each performance mode, resulting in more DVFS inefficiency, especially for larger X, where the difference is stressed. Figure 6.15 demonstrates that the inefficiency of the designs with the smaller LSQ increases significantly with X, while the inefficiency of the designs with the larger LSQ stays fairly constant, around 5%.

Table 6.12 shows the average power overhead with respect to ideal in different scenarios for the Issue stage with different IQ lengths and superscalar widths. The Issue stage has a significantly larger fraction of combinational logic (lower %seq) than the LSU. Thus, increasing the size of the IQ can result in a significant difference between the optimal designs for high and low performance, especially when superscalar width is low (since this exaggerates the impact of IQ length on design complexity). This also means that the effect of increasing X is amplified when the IQ is larger, as shown in Figure 6.16. Based on the above, we observe that reducing the sizes of microarchitectural structures that have low %seq (e.g., IQ) improves DVFS energy efficiency when range of scalability is large.

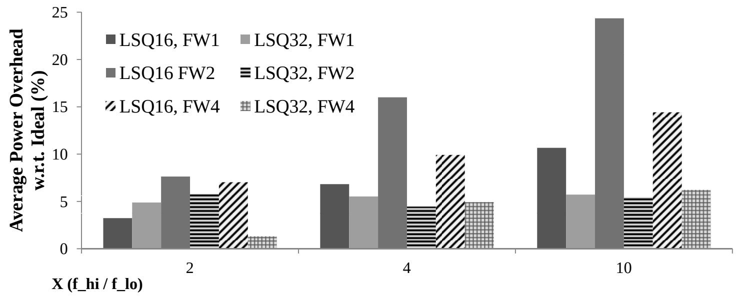


Figure 6.15: DVFS inefficiency increases more significantly with X for the LSU with a smaller LSQ.

The results above demonstrate that adapting the microarchitecture of a DVFS design can potentially improve DVFS efficiency. Note that we explore only a few microarchitectural features and primarily consider the design-level efficiency implications of microarchitectural

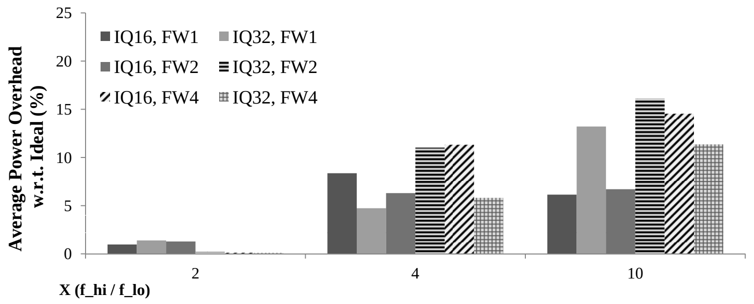


Figure 6.16: DVFS inefficiency increases more significantly with X for the Issue stage with a larger IQ, especially when superscalar width is low.

decisions; other considerations are beyond the scope of this work. For example, our results do not reflect the potentially increased cost of hazard recovery due to increased pipeline depth. A more thorough investigation of the effects of microarchitecture on DVFS efficiency at the system level is the subject of ongoing work.

6.5 Conclusions and Future Directions

DVFS is a popular technique for reducing power and energy consumption under dynamic operating conditions by targeting multiple power and performance modes in a single design. In this chapter, we demonstrate that DVFS-based designs obtained with conventional CAD methodologies can be energy-inefficient. This may be especially true for energy-constrained designs that spend a large fraction of time in a low-power mode. We identify the different factors that impact DVFS efficiency. Based on our insights, we propose a new approach to optimize for DVFS – *context-aware multi-mode design* – that considers the operating scenario to constrain and optimize a multi-mode design for improved energy efficiency. We also identify operating scenarios in which even an efficient multi-mode design exhibits substantial energy overhead with respect to the ideal energy consumption. For such operating scenarios, we propose a selective replication-based approach that maximizes energy efficiency while minimizing area overhead. We demonstrate that context-aware and selective replication-based design can provide up to 25% average power reduction with respect to conventional multi-mode design. Average power for the optimized design is within 1% of ideal, on average. Finally, we show that microarchitectural optimizations influence DVFS efficiency and demonstrate up to 18% average power reduction by optimizing processor microarchitecture for DVFS efficiency. Ongoing work includes continuing to investigate the connection between microarchitecture and DVFS efficiency. Future work includes exploring the implications of our techniques in design for yield – e.g., by improving the energy efficiency of processor designs that are binned according to their post-manufacturing characteristics and operate at a voltage or frequency that is determined only after manufacturing.

6.6 Acknowledgments

Chapter 6 is in part a reprint of “Enhancing the Efficiency of Energy-Constrained DVFS Designs”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2012.

I would like to thank my coauthors Professor Andrew B. Kahng, Professor Rakesh Kumar and Professor John Sartori.

Chapter 7

Approximate Arithmetic Designs

Guardbands for dynamic variations severely limit performance and energy efficiency of conventional IC designs. To overcome consequences of overdesign, several recent mechanisms for variation-resilient design [92] allow timing errors and manage design reliability dynamically. Relaxing the requirement of correctness for designs may dramatically reduce costs of manufacturing, verification and test [15]. In resilient designs, errors can be corrected with redundancy techniques (*error-tolerance*), or accepted in some applications relating to human senses such as hearing and sight (*error-acceptance*). In the error-acceptance regime, approximation via a simplified or inaccurate circuit can increase performance and/or reduce power consumption.

In the first part of this chapter, we propose an *accuracy-configurable approximate* (ACA) adder, which can configure the accuracy of results during runtime. Because of its configurability, the ACA adder can adaptively operate in both approximate (inaccurate) mode and accurate mode. The proposed adder can achieve significant throughput improvement and total power reduction relative to conventional adder designs. It can be used in accuracy-configurable applications, and improves the achievable tradeoff between performance/power and quality.

In the second part of this chapter, we propose an improved approach to estimate the output quality of approximate designs, based on lookup tables that characterize the statistical properties of approximate hardware modules and a regression-based technique for composing statistics to obtain expressions for output quality. Lookup tables that characterize approximate hardware modules improve the speed of our approach, while the regression-based composition technique improves accuracy for several error metrics by accounting for hardware configurations and data distributions.

The main contributions of our work are the following.

- The proposed ACA adder has runtime-configurable accuracy to enable better tradeoff of accuracy in computation versus performance and power.
- We provide quantitative metrics for an approximate arithmetic design. We compare the ACA adder to previous approximate adders based on these metrics.
- We demonstrate the power benefits of the ACA adder over previous approximate and conventional adder designs for accuracy-configurable applications.
- We propose composition rules for estimating the error metric (EM) observed at any net within an approximate circuit.
- We develop an approach to build pre-characterized libraries for individual approximate hardware modules and demonstrate how to accelerate the computation of composed EMs using the libraries. Our approach reduces runtime for characterization and results in improved accuracy compared to previous works [116][117].

7.1 Accuracy-Configurable Adder

In this section, we describe the new accuracy-configurable approximate (ACA) adder. We first present the proposed ACA adder design, along with its support of error detection and correction within a pipelined architecture. We then describe our setup for empirical validation, relevant metrics for approximate computation, and comparison of ACA versus previous approximate adders with respect to accuracy and power efficiency.

7.1.1 Approximate Adder Implementation

Previous approximate adders [166] [211] [247] have difficulty detecting and correcting errors since they are designed for error-acceptable applications with a target accuracy. However, accurate computations are still required at certain times, according to the application. The VLSA adder [233] can provide accurate results, but has large delay and area overhead for the error detection and correction. The central contribution of our present work is to propose an approximate adder which supports *both* accurate and inaccurate computation with error-correction and accuracy-configuration capability. Figure 7.1 shows our proposed approximate circuit for the case of a 16-bit adder. In the adder, the carry chain is cut to reduce critical-path delay, and

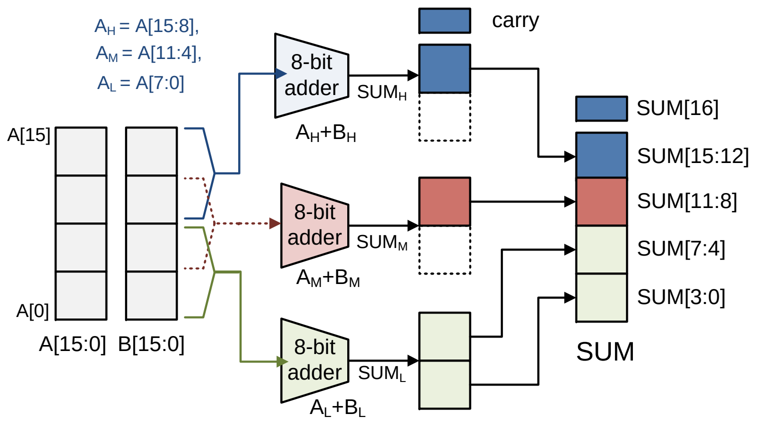


Figure 7.1: Proposed approximate adder – 16-bit adder case.

three sub-adders generate results of partial summations. With the reduced critical-path delay, high performance (by increasing the clock frequency) or low power consumption (by decreasing the operating voltage) is obtained. A middle sub-adder ($A_M + B_M$) is introduced to increase accuracy. Without the middle sub-adder (as in the ETAII approximate adder [246]), error occurs when the eighth carry bit is high, and for random input patterns the error rate is 50.1%. On the other hand, with the introduction of the middle sub-adder, error rate for random input patterns is reduced to 5.5%. (In an actual implementation, all redundant parts (four-LSB output of $A_H + B_H$ and $A_M + B_M$ sub-adders) would be optimized only for carry-generation.)

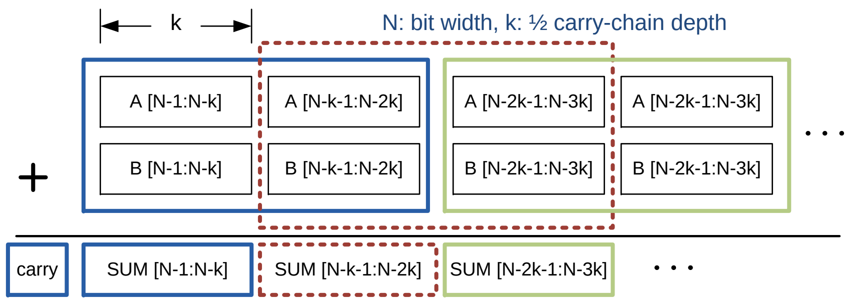


Figure 7.2: General implementation for the proposed adder.

We can generalize the implementation of the proposed approximate adder. Figure 7.2 shows the general implementation of an N-bit adder with a parameter k, which is the bit-width of the sub-adder result. In the adder, each divided submodule produces a k-bit result except for the last submodule, which produces a $2k$-bit result. The approximate adder thus consists of the $(N/k - 1)$ submodules as described in Equation (7.1).

$$\begin{aligned} &SUM[N-ik-1:N-(i+1)k] \\ &= A[N-ik-1:N-(i+2)k] \\ &+B[N-ik-1:N-(i+2)k], \\ &where\ i=0,...,N/k-2 \end{aligned} \tag{7.1}$$

In modern adder designs, such as carry-lookahead (CLA), carry-select and Kogge-Stone adders, the path depth and area are asymptotically proportional to log_2N and $Nlog_2N$ respectively, where N is the bit-width of the adder [248]. Based on this, we can express delay, area and power consumption of the proposed adder in terms of the parameters N and k. The proposed ACA adder has $(N/k-1)$ sub-adders, each of which is a $2k$-bit adder. Therefore, delay of the critical path can be expressed with Equation (7.2) and area can be estimated with Equation (7.3), where C_{delay} and C_{area} are constants for delay and area, respectively.

$$delay = C_{delay}(log_2k+1) \tag{7.2}$$

$$area = C_{area}(N-2k)(log_2k+1) \tag{7.3}$$

$$power_{dyn} = C_{power}(N-2k)(log_2k+1)^2 \tag{7.4}$$

Power consumption of the ACA adder can be roughly estimated as follows. Dynamic power consumption with voltage scaling at a fixed frequency is proportional to $capacitance \times {V_{dd}}^2$, where the $capacitance$ is proportional to the area. Cell delay is proportional to $1/(V_{dd}-V_t)^{\beta}$, and V_{dd}^2 is roughly proportional to $1/cell\ delay$ if we assume that β is 2. Since $cell\ delay \times path\ depth$ is constant at a fixed frequency, V_{dd}^2 is proportional to the path depth, which is log_2k+1. Consequently, dynamic power with voltage scaling can be expressed using Equation (7.4), where C_{power} is a constant fixed for given V_{dd} for dynamic power consumption. Static power consumption of the adder can be roughly estimated as proportional to the area in Equation (7.3).

In our proposed adder design, the output of each sub-adder (except the last sub-adder) is incorrect when a carry input should be propagated to the results. In Figure 7.1, when the $carry[4]$ (carry bit from A_L+B_L) is '1' and $SUM_M[3:0]$ is $1111_{(2)}$, the output result has an error in $SUM[11:8]$. In the general implementation, the output result will be correct when there are no errors in all $(N/k-1)$ sub-adders. In the i^{th} sub-adder, errors occur when (i) the LSB part of the result $(SUM_i[k-1:0])$ has all '1' values (probability $P = \frac{1}{2^k}$) and (ii) the LSB part ($[k-1:0]$) of the $(i+1)^{st}$ sub-adder produces a carry bit (probability $P = \frac{1}{4} + \frac{1}{2} \times \frac{1}{4} + \frac{1}{2} \times \frac{1}{2} \times \frac{1}{4} + ...$). Therefore, with a random input vector, the probability of having a correct result in the proposed adder is

$$P(N,k) = (1 - \frac{1}{2^k} \times \frac{2^k - 1}{2^{k+1}})^{\frac{N}{k}-2} \tag{7.5}$$

Table 7.1 shows the estimated results of 16-bit ACA adders with different parameter values k. With smaller k value, the minimum clock period and dynamic power can be reduced, but the pass rate (probability of having a correct result) will decrease. The estimations come from Equations (7.2), (7.3), (7.4) and (7.5). In Section 7.1.6 below, we validate the above estimation with real implementations.

Table 7.1: Estimated minimum clock cycle, area, dynamic power and pass rate for each k value when N = 16 (normalized to the conventional CLA 16-bit adder).

	k = 2	k = 3	k = 4	k = 5	k = 6
mininum clock period	0.5	0.65	0.75	0.83	0.89
area	0.87	1.05	1.12	1.15	1.12
dynamic power	0.44	0.68	0.84	0.95	1.00
pass rate	0.554	0.829	0.942	0.982	0.995

7.1.2 Error Detection and Correction for Accurate Computation

As described in Section 7.1.1, our proposed adder is incorrect when a carry bit is propagated between sub-adders. However, this error can be detected and corrected with a small overhead. We detect error for each sub-adder by checking the output of the sub-adder and the carry-in signal that comes from the previous sub-adder. Error detection can be implemented with several *AND* gates. To correct the error, '1' should be added to the approximate (inaccurate) output, and the error correction can be implemented with an incrementor circuit.

With these simple error detection and correction circuits, our proposed adder can be implemented to have variable latency as in the previous VLSA adder [233], but with small overhead for its *error detection and correction* (EDC) capability. Figure 7.3 shows an EDC system with our proposed adder. The error detection circuit (*AND* gates) checks the carry propagation and generates an error signal. The error correction (incrementor) circuit produces an error-free output

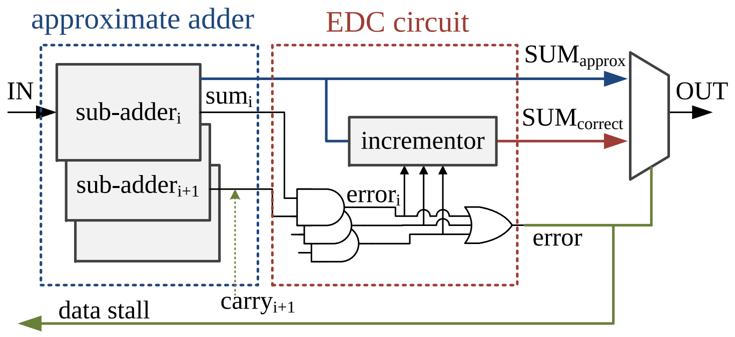


Figure 7.3: Error detection and correction with the approximate adder.

by adding compensation data, and requires an additional clock cycle. When errors are detected from input patterns, the *error* signal is activated. The *error* signal holds the input pattern during the error correction and chooses the error-corrected value ($SUM_{correct}$) as an output. With this approach, our approximate adder can provide accurate results at a higher clock frequency than that of conventional adders (e.g., CLA). According to the estimated results in Table 7.1, clock period can be reduced by 25% with 6% (= error rate) recovery-cycle overhead (16-bit ACA, $k = 4$).

7.1.3 Accuracy Configuration with Pipelined Architecture

When our proposed adder is combined with a pipelined architecture, we can obtain accurate results with the same throughput as a conventional adder. In the pipelined architecture, approximate additions are computed at the first pipeline stage, and error correction can be completed at the second stage. Figure 7.4 shows the conventional pipelined adder (above) and the approximate adder (below). The pipelined implementation of our approximate adder has a structural analogy with the pipelined adder of the 2006 U.S. patent of Mohammed and Hemmert [176] in which partial summations are performed at the first stage and carry bits are added at the later stages. However, the patent is clearly directed to accurate operations, not approximate computations. In addition, we use our approximate adder (Figure 7.2) in the first stage. In the pipelined approach, there is no improvement of the clock frequency since the achievable clock period is the same as that of the conventional adder. However, power benefits are obtained through configuration of accuracy: in the approximate mode, the error correction stage is power gated with footer (or, header) switches in Figure 7.4, and power reduction versus the conventional adder design can be achieved. We compare the conventional and approximate pipelined adders in Section 7.1.8.

In the proposed adder implementation, to achieve higher performance or lower power consumption, we can reduce the carry chain depth (k) of sub-adders (see Table 7.1). However, when k is less than $N/4$, it is impossible to correct all errors and achieve 100% correct results within one clock cycle, since the error-correction paths become critical. To achieve correct results in the pipelined implementation, the error-correction stage should be extended to multiple stages. Figure 7.5 shows the pipelined adder implementation with $k = N/8$, case, in which four pipeline stages are required to achieve a 100% accurate result. In the pipelined adder, each stage generates a result with different accuracy; the output accuracy increases as the number of pipeline stages increases. According to the given accuracy requirement, we can turn off the later stages with power gating, and we can further reduce power consumption by exploiting the accuracy tradeoff.

Since the proposed adder supports both approximate and accurate results, it can be used in applications that require accurate results only under certain conditions. Conventional accurate designs are energy-inefficient in the error-acceptable application context because they always compute the exact function. Previous approximate designs cannot handle a varying accuracy requirement, which limits the benefit of the accuracy tradeoff since, as noted above, the approximate function must meet the maximum accuracy threshold across all applications. Moreover, if the application requests an exact computation, previous approximate designs require augmentation with additional accurate circuits. By contrast, the ACA design efficiently exploits a tradeoff between accuracy and power/performance with its runtime accuracy configurability.

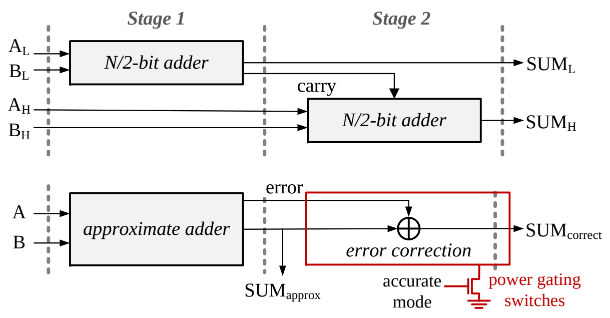


Figure 7.4: Pipelined adder implementation – conventional adder (above) and approximate adder (below). During approximate-mode operation, the error correction stage is power gated.

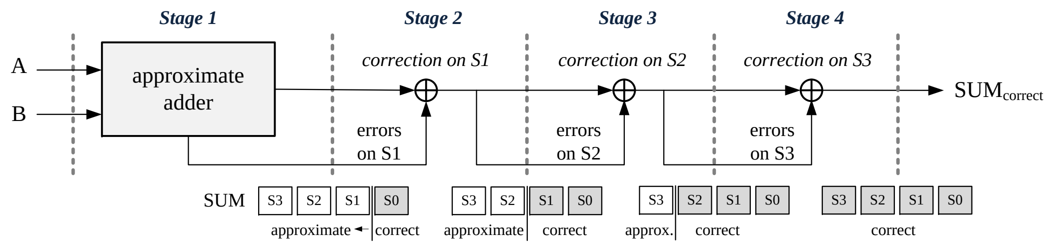


Figure 7.5: Accuracy-configurable implementation for pipelined adder.

7.1.4 Experimental Setup

To test the impact of ACA in approximate designs, we have written each design (ACA, CLA, Lu's adder, ETAI and ETAIIM) in Verilog and synthesized it to a TSMC 65GP cell library with *Synopsys DesignCompiler* [26]. We then perform gate-level simulations using *Cadence NC-Sim* [4]. In the simulation, gate delay is taken from an SDF (standard delay format) file [13]. For voltage scaling experiments, we prepare *Synopsys Liberty* (.lib) files for each voltage from 1.00V to 0.60V in 0.01V increments, using *Cadence Library Characterizer v9.1* [3]. The prepared libraries are used for SDF file generation and power estimation at each voltage. Each simulation is performed with input patterns for one million cycles. During the simulation, each output value is compared with a reference (correct) value to enable evaluation of various accuracy metrics. For the input patterns, we use random data as well as actual data from *SPEC 2006* [23] benchmarks; in the latter case we extract operand data from ADD instructions in the SPEC benchmarks.

7.1.5 Metrics for Approximate Design

To quantify errors in approximate designs, two metrics have been previously proposed [51]. *Error rate* (ER) is the percentage of cycles in which output value is different from the correct value. *Error significance* (ES) is the numerical difference between correct and output results; this quantifies the amount of error. In image/video applications, [66] uses the product of ES and ER as a metric of error tolerance. [211] introduces a criterion for acceptability: ES $\times$ ER $\leq$ *acceptance threshold*, where the acceptance threshold is specified according to the application. For the error significance (ES) metric, [247] considers only error amplitude. This is useful for many digital signal processing (DSP) systems that process, e.g., sound and image data. However, in communication systems that mainly handle information data, the number of incorrect bits (Hamming distance) is a more meaningful metric for accuracy – e.g., a *(32,28)*

Reed-Solomon code can correct up to 2-byte errors. This consideration for the ES metric is required when approximate arithmetic is applied to error-tolerant systems with a redundancy technique.

Table 7.2 defines two accuracy metrics for amplitude and information data. ACC_{amp} is used in [247] and quantifies the amplitude of errors, where R_c and R_e are the correct and obtained results, respectively. We propose another accuracy metric, ACC_{inf}, which measures error significance as Hamming distance. In the definition of ACC_{inf}, B_e is the number of error bits and B_w is the bit-width of the data. For example, when the correct (reference) data is $1000\_0000_{(2)}$ and the result data is $1100\_0000_{(2)}$, accuracy according to ACC_{amp} and ACC_{inf} will be $\frac{1}{2}$ and $\frac{7}{8}$, respectively. To assess both ER and ES in approximate circuits, we obtain average values of accuracy metrics ACC_{amp} and ACC_{inf} over entire simulation traces.

Table 7.2: Accuracy metrics for error significance (ES).

metric	definition	data type
ACC_{amp}	$1 - \lvert R_c - R_e \rvert / R_c$	amplitude data
ACC_{inf}	$1 - B_e/B_w$	information data

Table 7.3: ACA adder results with different k values.

k	2	3	4	5
min. clock period (ps)	180	190	220	230
area (um^2)	550	990	920	840
pass rate (%)	55.3	82.8	94.0	98.1
throughput improvement (%)	11.3	24.6	22.3	21.4

Table 7.4: Design comparison for each adder design.

	CLA	LU	ACA	ETAI	ETAIIM
area (um^2)	910	1356	923	576	678
min. clock period (ps)	280	210	200	200	260
pass rate (%)	100	99.2	94.1	10.0	97.0
ACC_{amp} (maximum)	1.000	0.998	0.997	0.999	0.999
ACC_{inf} (maximum)	1.000	0.999	0.993	0.694	0.996
area overhead for EDC	N/A	75%	28%	N/A	15%

7.1.6 Approximate Adder with Different Parameters

We explore the proposed ACA adder with different parameters (k: half of carry-chain depth). Table 7.3 summarizes results – minimum clock period, area, error rate and throughput improvements – for each implementation of the 16-bit adder with different k values. According to the results, with smaller k, the maximum operating frequency increases, but the error rate increases as well. With higher k, the error rate is reduced significantly, but the benefit of the approximate circuit, i.e., clock period reduction, is small. In the table, throughput improvement over conventional design is calculated including error recovery overhead. From the implementations, a maximum throughput improvement is achieved when k = 3. If we correct erroneous results with EDC as in Figure 7.3, then 17.2% additional clock cycles are required for error correction. With this overhead, the ACA adder can improve data throughput by 24.6% over the conventional CLA adder.

7.1.7 Approximate Adder Comparison

We now evaluate ACA and previous approximate as well as exact adders with respect to the pass rate and the accuracy metrics which we have proposed. We use gate-level simulation at each possible clock period to compare five adders: CLA, Lu's adder [166], ETAI, ETAIIM [247] and the proposed ACA adder (without error correction). In the experiment, the same carry-chain width (8-bit) is selected for the four approximate adders. In the implementation, a register (flip-flop) is inserted at each output port to detect timing errors.

Table 7.4 shows area, pass rate, accuracy, minimum clock period and EDC overhead for each adder design. According to the results, the ETAI adder has the smallest design area, but has a low pass rate and limited accuracy with respect to the ACC_{inf} metric. Therefore, the ETAI adder is preferred for applications which allow low accuracy in results. The ETAIIM adder shows fairly high accuracy, but does not have any speed (clock period) benefit. Lu's adder shows a smaller error rate and high accuracy with respect to both ACC_{amp} and ACC_{inf} metrics. However, it requires larger area than the other designs. The proposed adder shows similar results for both metrics as Lu's adder. However, the area of the ACA adder is smaller than that of Lu's adder, and EDC is possible with small area overhead (28%). With the ACA adder, the minimum clock period can be reduced by 26% compared to the accurate CLA.

Figure 7.6 shows the tradeoff of power versus accuracy in a voltage scaling scenario: the x-axis shows total power consumption, and the y-axis shows accuracy (ACC_{amp}, ACC_{inf}). The power consumption and accuracy are measured with different voltage libraries character-

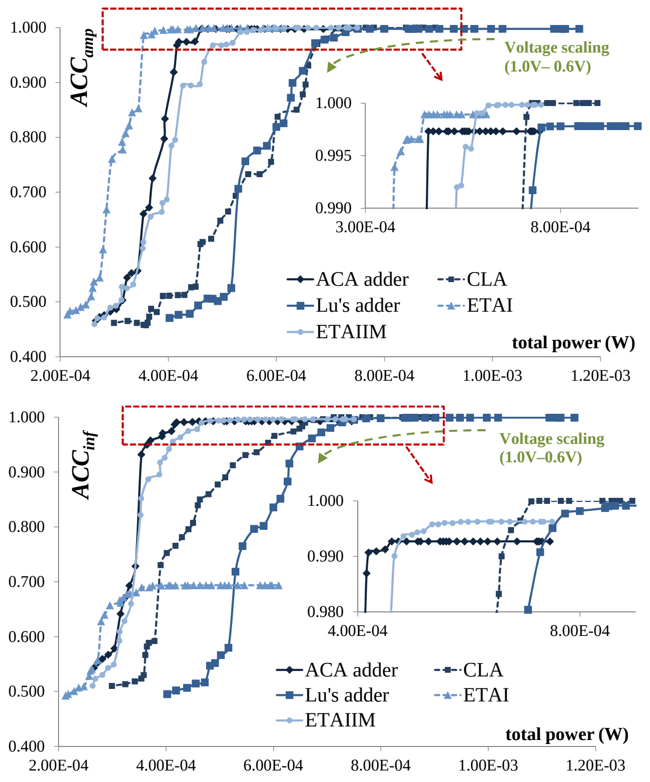


Figure 7.6: Accuracy (y-axis) versus power consumption (x-axis) under fixed clock period ($0.25ns$) and scaled voltage (from $1.0V$ to $0.6V$).

ized using *Cadence Library Characterizer* [3]. The clock period is fixed at $0.30ns$ during the simulations. In the results, Lu's adder does not show power benefits due to its design size. ETAI shows low power consumption and high ACC_{amp} accuracy, but has low ACC_{inf} accuracy and cannot detect and correct errors. ETAIIM shows similar characteristics to ACA in the voltage scaling case, but the adder cannot be used for a high-performance (high-frequency) design, as shown in Table 7.4. The results in Figure 7.6 imply that our proposed adder can provide a significant power reduction over existing adders, if a small accuracy penalty is accepted. When the required accuracy is 0.970 (ACC_{amp}), the ACA adder shows 37.0%, 36.4% and 15.9% total power reduction versus CLA, Lu's adder and ETAIIM, respectively.

We have tested our approximate adder on a real application, namely, the Gaussian smoothing filter used in [154]. Gaussian smoothing is performed on the input image by convolving with a matrix in the spatial domain. In the convolution, the addition operation is done with approximate 16-bit adders. Other operations, such as multiplication and division, are accurate computations. Figure 7.7 shows results for various approximate adders when they consume

50% of the power of accurate CLA. From the results, the ACA adder has PSNR of $24.5dB$, and this suggests that image processing/filtering applications could employ our proposed adder with significant power savings and only small loss in image quality.

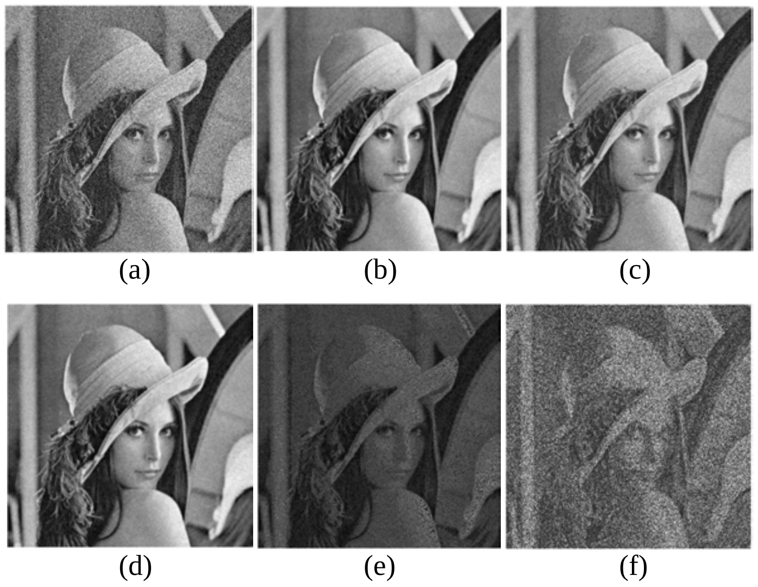


Figure 7.7: Image smoothing: (a) original image with noise; (b) accurate adder; (c) ACA (PSNR: $24.5dB$); (d) ETAI (PSNR: $25.3dB$); (e) ETAIIM (PSNR: $16.2dB$); (f) Lu's adder (PSNR: $11.1dB$).

Table 7.5: Comparison between conventional and approximate pipelined adders in accurate mode.

	conventional pipelined			approximate pipelined			
adder width (N)	area (um^2)	clock period (ns)	total power (mW)	k	area (um^2)	clock period (ns)	total power (mW)
8	459	0.313	0.557	2	576	0.312	0.564
16	1082	0.357	1.558	4	1171	0.358	1.669
32	2252	0.404	2.860	8	2420	0.414	2.914

7.1.8 Accuracy Configuration and Power Savings

When the architecture allows pipelining for addition, our proposed adder can be implemented as shown in Figure 7.4. We implement both the conventional pipelined adder and the approximate pipelined adder in order to compare the designs with respect to area, timing and

Table 7.6: Implementation results of 32-bit ACA adder with 4-stage pipeline (power consumption of each mode and power reduction over conventional pipelined adder).

configuration	power-gating	ACC_{amp} (max.)	ACC_{inf} (max.)	total power (*mW*)	reduction (%)
mode 1	none	1.000	1.000	5.962	-11.5%
mode 2	Stage 4	0.998	0.960	4.683	12.4%
mode 3	Stage 3, 4	0.991	0.925	3.691	31.0%
mode 4	Stage 2, 3, 4	0.983	0.900	2.588	51.6%

power. In the implementation, registers (flip-flops) are included at each pipeline stage (before Stage 1, between Stage 1 and Stage 2, and after Stage 2).

Table 7.5 shows the implementation results for the conventional and approximate pipelined adders. The parameter k has been selected as $N/4$ for a two-stage pipelined implementation. In the table, minimum clock period is measured at a fixed voltage ($1.0V$), and total power is measured at a fixed frequency ($2.5GHz$) with voltage scaling. In the ACA adder case, timing and power overheads from power gating cells, output MUXes, and IR drop are included. We can see that area, timing and power of both designs are similar when the ACA adder operates in the accurate mode. Total power of the approximate adder is comparable to that of the conventional adder, even though ACA has additional EDC circuits. This is because ACA has fewer registers between Stage 1 and Stage 2 than the conventional pipelined adder. (In Figure 7.4, the conventional adder requires registers for A_H, B_H, SUM_L and carry at the first stage. For a 16-bit adder, 25 registers (8 + 8 + 8 + 1) are required. On the other hand, ACA requires 18 registers (16 for SUM_{approx} and 2 for error indication).)

In the pipelined architecture, the ACA adder can provide various configurable modes according to the pipeline depth. To improve the design performance, we increase the pipeline depth; the deeper pipeline reduces the path depth of the design. In the conventional pipelined adder, bit-width of the adder in each stage can be reduced to $N/\#stage$, where N is the entire bit-width and $\#stage$ is the depth (number) of the pipeline stages. In the ACA adder, we can reduce the value of parameter k with deeper pipeline depth as shown in Figure 7.5. To show the benefit of accuracy configuration, we have implemented a 32-bit ACA adder ($N = 32$, $k = 4$) with 4-stage pipeline, and compared it with a conventional pipelined adder with an 8-bit CLA in each stage. Table 7.6 shows the implemented results for the 32-bit ACA adder. For the accuracy estimation, one million cycles of random patterns are used. The ACA adder can operate in four different modes, based on the power gating of each stage. We can see that the modes show

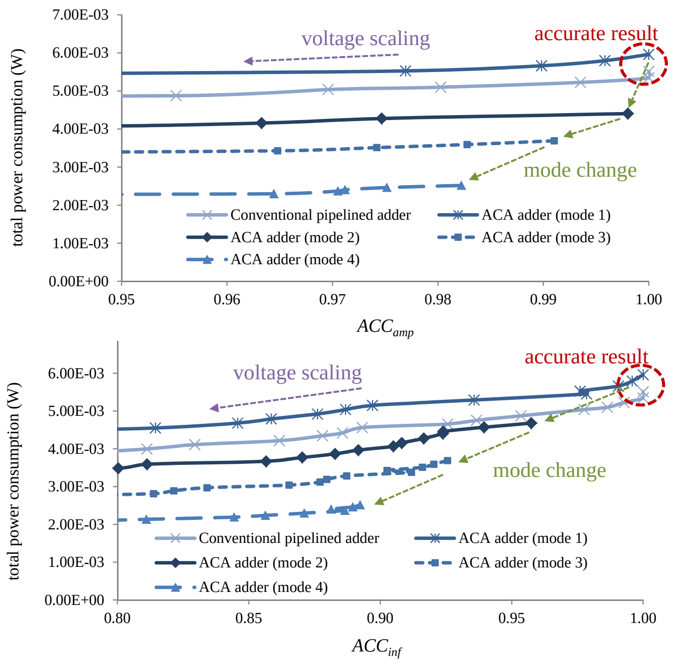


Figure 7.8: Accuracy metric ACC_{amp} (above) and ACC_{inf} (below) versus power consumption for conventional pipelined adder, ACA adder in accurate mode, and ACA adder in approximate mode (4-stage, 32-bit adder).

different power consumptions and different achievable accuracies. The ACA adder consumes 11.5% more power than the conventional adder in accurate mode (mode 1) due to the presence of recovery circuits. At the same time, it shows significant power reductions in the approximate modes: 12.4%, 31.0% and 51.6% in mode 2, mode 3 and mode 4, respectively. Figure 7.8 shows detailed results for power consumption versus accuracy metrics in each configuration. From the results, we can see that accuracy configuration with the mode change is much more effective than with voltage scaling, in terms of the tradeoff between accuracy and power.

Last, we also obtain accuracy results in each accuracy mode with real input patterns extracted from SPEC 2006 benchmarks. Table 7.7 shows accuracy results of a 32-bit ACA adder with such real input patterns. The accuracy results are different for each benchmark, e.g., the measured accuracy for *bzip2* is higher than for *gcc*. Furthermore, the accuracy with real patterns is greater than with random input patterns (Table 7.6), most likely because addition inputs for general-purpose processor traces have infrequently and/or systematically changing patterns in the applications. Figure 7.9 shows power reduction achieved by the ACA adder versus the conventional pipelined adder under the accuracy requirements. We assume that required accuracy is between 0.99 (0.95) and 1.0 for ACC_{amp} (ACC_{inf}), and that it varies uniformly

Table 7.7: Accuracy (ACC_{amp}, ACC_{inf}) results of 32-bit ACA adder for real benchmarks (SPEC 2006).

accuracy metric	benchmark	*astar*	*bzip2*	*calculix*	*gcc*	*h264ref*	*mcf*	*sjeng*	*soplex*
ACC_{amp}	mode 1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	mode 2	0.9999	1.0000	0.9999	0.9992	0.9999	0.9997	0.9998	0.9999
	mode 3	0.9993	0.9998	0.9972	0.9990	0.9990	0.9997	0.9995	0.9998
	mode 4	0.9979	0.9970	0.9958	0.9951	0.9978	0.9991	0.9981	0.9953
ACC_{inf}	mode 1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	mode 2	0.9979	1.0000	0.9978	0.9881	0.9953	0.9819	0.9897	0.9985
	mode 3	0.9949	0.9984	0.9967	0.9849	0.9897	0.9809	0.9876	0.9965
	mode 4	0.9940	0.9931	0.9910	0.9617	0.9851	0.9596	0.9787	0.9925

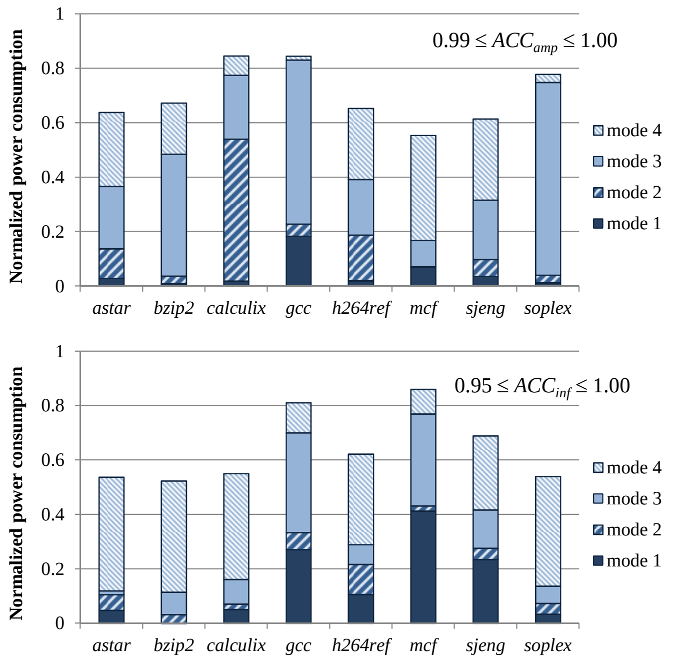


Figure 7.9: Normalized power consumption, relative to that of conventional pipelined design, when the accuracy requirement is varied uniformly.

over this range during the entire runtime. From the results, dynamic accuracy configuration achieves up to 44.5% (30.0% on average) and 47.1% (35.8% on average) power reduction over the conventional pipelined design for ACC_{amp} and ACC_{inf} metrics, respectively.

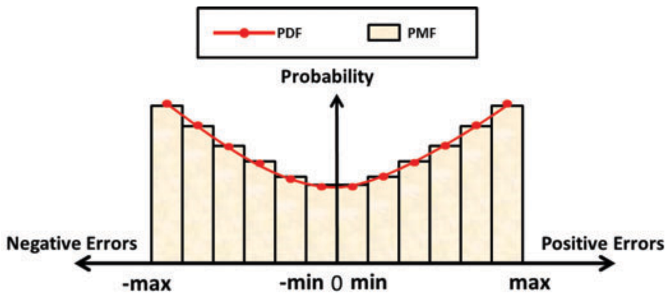


Figure 7.10: Probability mass function (PMF) used in the interval-based approach [116] [117].

7.2 Statistical Analysis and Modeling for Error Composition

In this section, we describe initial work toward a compositional methodology for error metric estimation in approximate arithmetic circuits. We first assess potential weaknesses of a previous interval-based approach to error metric composition. We then motivate and describe our statistical characterization of the error properties of approximate hardware modules, along with the use of regression and lookup table-based techniques. Last, we demonstrate the application of composition rules and pre-characterized error libraries to analyze arbitrary circuit topologies.

7.2.1 An Interval-based Approach to Error Rate Estimation

Huang et al. address the issue of error rate estimation for approximate circuits in [116] [117]. Their flow first characterizes approximate hardware modules by simulating the error probabilities for different input value intervals. They then use *interval arithmetic* to estimate the *probability mass function* (PMF) of errors produced and propagated in an approximate arithmetic circuit. After propagating and composing errors with interval arithmetic, the error metrics are obtained from PMFs. The interval-based approach samples the *probability distribution functions* (PDFs) or PMFs of errors to generate sampled PMFs. The height of each interval in the sampled PMF represents the probability of error. Figure 7.10 shows an example PMF which is used in the interval-based approach.

We observe two drawbacks in the interval-based approach. First, there is a quantization error, since the approach represents multiple error values with a single interval. If the actual error distribution varies greatly within one interval, the estimation will be inaccurate. Second, the interval-based approach requires consecutive intervals to cover the range from maximum to minimum error magnitude (max and min in Figure 7.10). If the errors exceed max or min, the interval-based approach will clamp the estimated errors to the max or min values, and the estimation error will be saturated. If high portion of errors or data experience *saturation issue*, the estimation inaccuracy will be high. To address these drawbacks, the interval-based approach requires re-characterization of the libraries to increase the number of intervals, requiring significant runtime overhead. For better understanding of the strengths and weaknesses of the interval-based error metric (EM) composition, we evaluate the EM estimation with a test-case shown in Figure 7.11(a). We vary the input distribution to evaluate accuracy for different input distributions and hardware configurations. We collect results from 100 combinations (10 Gaussian distributions with different standard deviations and 10 sets of ETAIIM configurations). Figure 7.11(b) shows the runtime of library characterization performed by the interval-based approach for different numbers of samples per interval. The accuracy results of the interval-based approach compared to Monte Carlo simulation are shown in the form of a correlation plot in Figure 7.11(c). From Figure 7.11(b) we notice that increasing the sample size to 18.5M requires 1.7 hours for library characterization, but estimation errors (offsets) are still observed in Figure 7.11(c). Possible reasons for the inaccuracy are (i) the use of discrete PMF and (ii) inaccurate propagation of EMs from the pre-characterized library.

7.2.2 Analysis for Computation of Error Metrics

We analyze an ETAIIM adder to understand the error generation of approximate modules. ER of ETAIIM adder is given in Equation (7.6). N is the total bit width of the adder. Bits-per-block (BPB) of ETAIIM is the size of carry-look-ahead (CLA) blocks and k is the number of connected CLA blocks, an architectural parameter used to control error magnitudes. From Figure (7.12), we observe that the errors are related to the input values of CLA blocks because errors occur when all input bits of the CLA block are in carry-propagate state. For example, if most of the input values are small, then the probability of generating larger errors will be small. This observation regarding ETAIIM motivates us to study the sensitivity of EMs to input distributions.

$$\begin{aligned} ER_{ETAIIM} &= 1 - (1 - \frac{1}{2^{BPB}} \frac{2^{BPB} - 1}{2^{BPB+1}})^{\frac{N}{BPB} - 2 - k} \\ &\times (1 - \frac{1}{2^{(BPB \times k)}} \frac{2^{(BPB \times k)} - 1}{2^{(BPB \times k) + 1}}) \end{aligned} \tag{7.6}$$

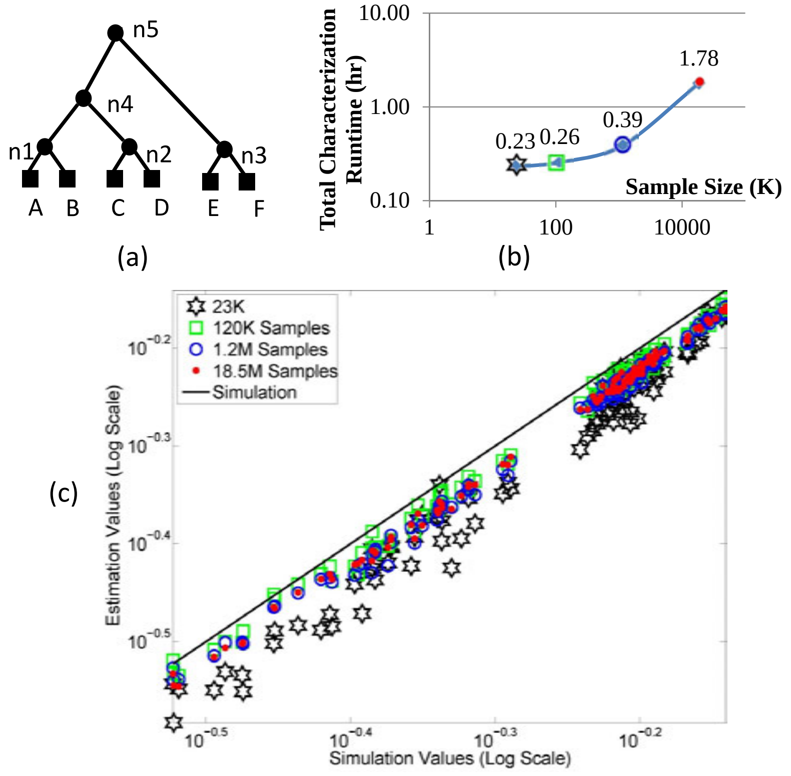


Figure 7.11: (a) Five-node testcase. (b) Runtime from interval-based approach for each sample size. (c) ER estimation results from interval-based approach. The results are generated from 100 testcases (10 hardware configurations and 10 combinations of input distributions).

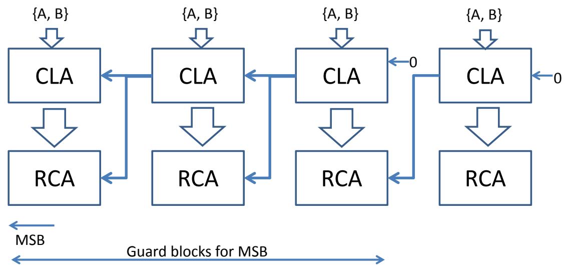


Figure 7.12: The structure of an ETAIIM approximate adder. CLAs are carry-lookahead sub-adders. RCAs are ripple-carry sub-adders.

7.2.3 Proposed Approach to Estimate EMs

The analytical expression in Equation (7.6) is based on the assumption that distributions of the input values are uniform and the ranges cover from the MSB to LSB. However, this is not always the case, and we need to consider input distributions for the accurately-estimated EMs.

To analyze the relationship between input distributions and EMs, we use 24-bit ETAIIM adders and simulate the EMs for different BPB and k. Figure 7.13 shows each simulated EM value (y-axis) with respect to the standard deviation of input data (x-axis). In the ETAIIM adder, 20 bits are used for the fractional part, and the MSB guard block size k takes on values from one to four.

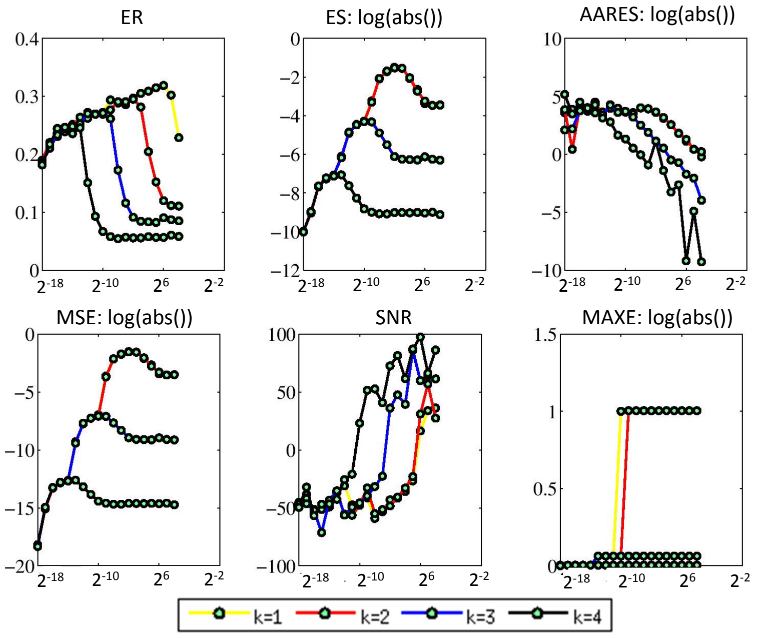


Figure 7.13: The simulated EM results for input distributions.

Figure 7.13 shows that EM values change with respect to both the standard deviations of input values and the hardware configuration (k). Based on the results, we construct lookup tables to model the error metric of approximate modules instead of using analytical expressions. Modeling with lookup tables is preferred since it is difficult to derive an analytical expression if input values are not uniformly distributed.

Figure 7.15 illustrates our EM formulation. To estimate the output EM (EM_Z), we consider *intrinsic* EM values (EM_{in}) which are generated by the approximate module itself, and *propagated* EM values (EM_A, EM_B) which come from the previous stages.

In our EM estimation framework, we propose a lookup table (LUT)-based approach to consider different input distributions. The lookup tables for different hardware configurations are merged to become the pre-characterized library. We construct two types of lookup tables, as illustrated in Figure 7.14(b). The tables, EM_Z and STD_Z, contain (intrinsic) EM values and output standard deviations, respectively, with respect to the input standard deviations.

Our LUT-based approach can be divided into three steps as described in Figure 7.14(a).

Step 1: Value distribution propagation in the circuit topology. We generate statistical properties with pre-characterized libraries. To obtain the statistical property of each node in the circuit, we traverse all the nodes in the circuit in a topological order from primary inputs to a primary output. During the traversal, we look up the statistical property (standard deviation) from a pre-characterized table (STD_Z), and annotate the standard deviation values at all nodes.

Step 2: EM estimation for approximate modules. With standard deviations of the internal nodes, we estimate EM values using a pre-characterized table (EM_Z) for each internal node. The lookup table, EM_Z, is characterized by simulating EM values as shown in Figure 7.13. We generate the LUTs for different approximate modules to estimate intrinsic error metric (EM_{in}), which is generated by modules themselves without input errors. By combining Steps 1 and 2, we can estimate the EM_{in} of each node in any circuit topology.

Step 3: Error composition with EMs of each approximate module. With the generated EMs (EM_{in}) of each approximate module, we apply a regression approach to find the composed EM values in the primary output. The error rate (ER) can be computed by multiplying pass rate (1-ER), and the composed ER is generated with Equation (7.7), where ER_Z is the composed ER, ER_A and ER_B are the propagated ERs to the inputs in Figure 7.15, ER_{in} is an intrinsic ER, and $\alpha_{\{in,P\}}$ are regression coefficients. Other EMs (ES, ARES, MSE, SNR and MAXE) are amplitude-based error metrics, and we generate the composed EM from Equation (7.8), where $\alpha_{\{in,P,C\}}$ are regression coefficients.

$$ER_Z = 1 - 10^{\alpha_C} \times (1 - ER_{in})^{\alpha_{in}} \times ((1 - ER_A) \times (1 - ER_B))^{\alpha_P} \tag{7.7}$$

$$EM_Z = \alpha_{in} EM_{in} + \alpha_P (EM_A + EM_B) + \alpha_C. \tag{7.8}$$

To verify the correctness of our table lookup method in Step 2, we estimate the standard deviation (STD) and EM_{in} as shown in Figure 7.16. We test with 10 combinations of hardware configurations and 10 combinations of different input distributions (Gaussian distribution with different standard deviations). Figure 7.16 shows the correlations between the estimated and simulated STD/EM values from all internal nodes. The results show that we can obtain correct STD values from the lookup table with the topology traversal. With the estimated STD, we observe correct estimation for EM_{in} (ER and ES). We find that ARES results are less accurate compared to the results of ER and ES. This is because ARES measures error relative to input data. If the magnitude of input data is small (near zero), the range of the ARES value will be

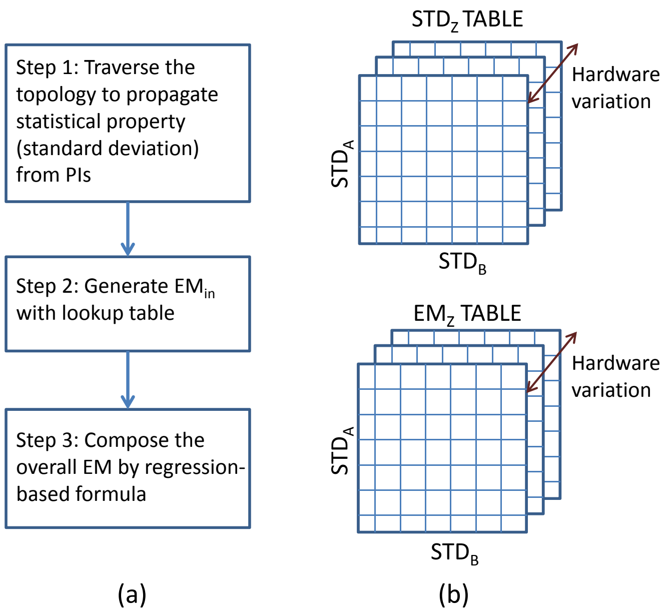

Step 1: Traverse the topology to propagate statistical property (standard deviation) from PIs
Step 2: Generate EM$_{in}$ with lookup table
Step 3: Compose the overall EM by regression-based formula
STD$_Z$ TABLE
Hardware variation
STD$_A$
STD$_B$
EM$_Z$ TABLE
Hardware variation
STD$_A$
STD$_B$
(a)
(b)

Figure 7.14: (a) Our proposed approach for error estimation, and (b) the lookup tables in the pre-characterized library for EM_{in} and STD_{out}.

large. In such a context, accurate estimations are difficult, given the limited number of grids in the lookup table.

Table 7.8 shows our regression results for improved EM estimations. The upper part (a) of the table shows regression parameters derived with different hardware configurations. The lower part (b) of the table shows the estimated inaccuracy, as defined in Equation (7.9), where R_c and R_e are the correct and obtained results, respectively. The inaccuracy is shown for both "without regression" and "with regression" cases.

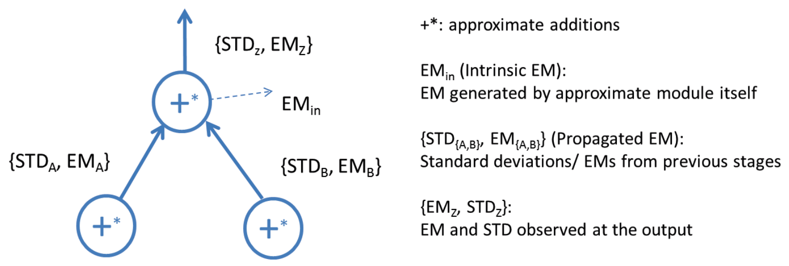

{STD$_Z$, EM$_Z$}
+*
EM$_{in}$
{STD$_A$, EM$_A$}
{STD$_B$, EM$_B$}
+*
+*
+*: approximate additions
EM$_{in}$ (Intrinsic EM):
EM generated by approximate module itself
{STD$_{\{A,B\}}$, EM$_{\{A,B\}}$} (Propagated EM):
Standard deviations/ EMs from previous stages
{EM$_Z$, STD$_Z$}:
EM and STD observed at the output

Figure 7.15: EM estimation at a given node (approximate module) considering intrinsic and propagated EMs.

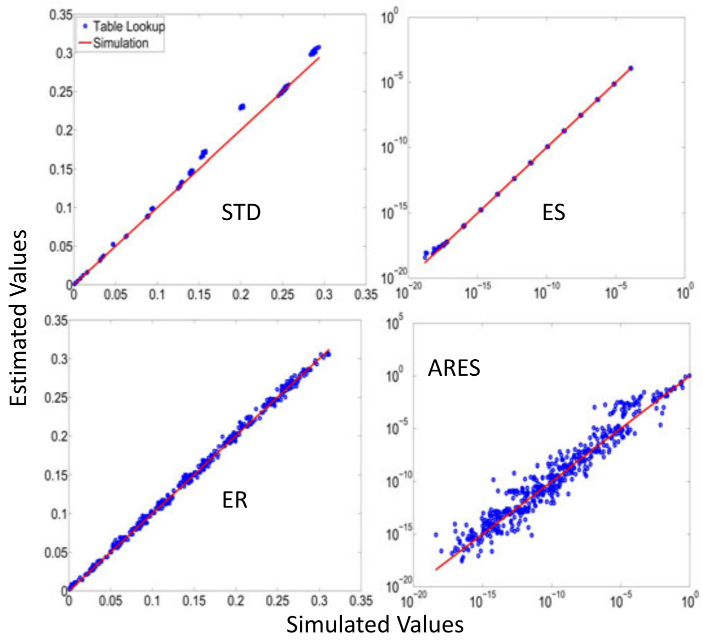


Figure 7.16: Estimated STD_A or STD_B and EM_{in} values obtained from lookup tables. The x-axes are simulated values and y-axes are estimated values. The red lines show the ideal estimations and the blue dots show the estimated results from our proposed method.

$$Inaccuracy = |R_c - R_e|/|R_c| \tag{7.9}$$

Without regression, we report inaccuracy results with α_{IN} = α_P = 1 and α_C = 0 for the coefficients in Equations (7.7) and (7.8); this is a pessimistic assumption (i.e., that there are no overlap effects from the composition). To explore the coefficients of our propagation model, we simulate a single approximate adder with different operating conditions, which we model by changing the input distribution (Gaussian distribution with different standard deviations), and applying artificial errors. The artificial errors are also assumed to have Gaussian distribution with different standard deviations. We obtain the regression coefficients from the simulation, then apply the coefficients in EM estimations, and report the inaccuracy of EM results. With the regression coefficients, the accuracy of estimation is significantly improved for ER, ARES and SNR. However, ES, MSE and MAXE have degraded accuracy results due to our simple regression models, and improving the models is one of our ongoing works.

Table 7.8: (a) Regression coefficients derived with different hardware configurations, and (b) estimation inaccuracy with or without the regression.

(a) regression parameters						
	ER	ES	ARES	MSE	SNR	MAXE
α_{IN}	1.03E+00	1.00E+00	2.42E-02	1.00E+00	3.46E-01	9.40E-01
α_P	1.26E+00	9.98E-01	9.76E-01	1.00E+00	7.15E-02	7.98E-01
α_C	-5.85E-03	5.74E-08	-5.92E-03	-5.55E-09	-1.27E+00	8.65E-05
(b) inaccuracy with regression parameters						
without regression	4.18e-02	8.3e-02	1.28+03	1.22e-01	1.35e+02	1.29e-01
with regression	7.47e-03	5.41e-01	2.65e+01	4.12+04	4.01-01	1.88+01

7.2.4 Experimental Results

To evaluate the accuracy and performance of our EM estimation approach, we perform several experiments. First, we demonstrate that our approach can be applied to a four-tap finite impulse response (FIR) filter. In the FIR experiment, the accuracies of six error metrics are evaluated. Second, we use *multiply-accumulator* (MAC) circuits with different sizes to compare the accuracy and runtime between our approach and the interval-based approach. Finally, we evaluate the accuracy of estimated results for randomly generated topologies. In the experiments, we use 64-bit ETAIIMs with different k parameters. The adders are assumed to have 60 fractional bits.

FIR filter. To demonstrate that our approach is applicable to realistic computation circuits, we estimate EMs for the FIR filter design illustrated in Figure 7.17(a). Lookup table characterization for each error metric and standard deviation is performed for 12×12 different combinations of standard deviations (2^0, 2^{-2}, ..., 2^{-22}). For each entry in the tables, we use 90K samples to obtain standard deviations and EMs. The runtime for building this set of lookup tables with ETAIIM adders is 1.37 hours on a 2.8*GHz* Intel Xeon E5-2640 Linux workstation with 128*GB* of memory. With our lookup tables, we implement the flow in Figure 7.14 with Matlab [21].

Table 7.9 shows inaccuracy results of the estimations for each EM. We assume that the constant multipliers are accurate, and the adders in the FIR filter are approximate modules. In the second column (error type), "IN" means an intrinsic EM value generated by the approximate modules themselves, and "P" means a propagated EM value composed from the EMs in previous stages. Based on the results in Table 7.9, our approach provides accurate EM estimations for ER, ES, MSE and MAXE metrics. For the same testcase, the inaccuracy of the interval-based approach is 17.6% and 60.2% for ER and ES, respectively.

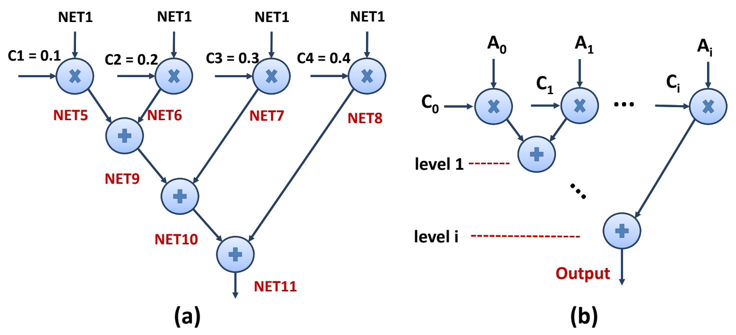


Figure 7.17: Configuration of (a) FIR filter and (b) multiply-accumulator (MAC) circuits used in the experiments.

Table 7.9: Estimation inaccuracy of a four-tap FIR filter shown in Figure 7.17(a).

		estimation inaccuracy					
net	type	ER	ES	ARES	MSE	SNR	MAXE
NET9	IN	0.3%	6.4%	17.0%	6.4%	19.1%	0.0%
NET10	IN	1.3%	2.6%	61.9%	3.3%	10.7%	0.0%
NET11	IN	1.0%	6.3%	419.6%	6.2%	6.1%	0.0%
NET11	P	13.4%	5.8%	692.3%	5.8%	436.4%	0.7%

MAC circuits. We test the accuracy and runtime of our approach against the interval-based approach for the MAC circuits shown in Figure 7.17(b), which are the general case of the FIR filter. We use 280 MAC circuits, having 14 different levels and 20 different configurations (parameters of each adder, constant values C_i, and input distributions). We estimate EMs for the MAC circuits using our approach and the interval-based approach. Figures 7.18 and 7.19 show correlation plots for ER and ES, respectively. For ER, we observe that our approach achieves 1.28× better accuracy than the interval-based approach with 8.4× faster runtime. For ES, we observe that the estimated results from the interval-based approach are clamped to -2^{-20} on the right end. This is due to the saturation issue mentioned in Section 7.2.1. Max inaccuracy is defined as $max(|R_c - R_e|)$, where R_c and R_e are the simulated and estimated results, respectively. Figure 7.19 shows that for the same testcases our approach is not affected by the saturation problem; this is because the estimates of ES are interpolated or extrapolated from the lookup tables.

We evaluate runtime and accuracy for increasing circuit complexity by increasing the number of circuit levels in Figure 7.17(b). Figure 7.20(a) shows how runtime scales with circuit complexity. We observe that the runtime of error composition increases linearly for both our approach and the interval-based approach. Our approach is 8.4× faster than the interval-based

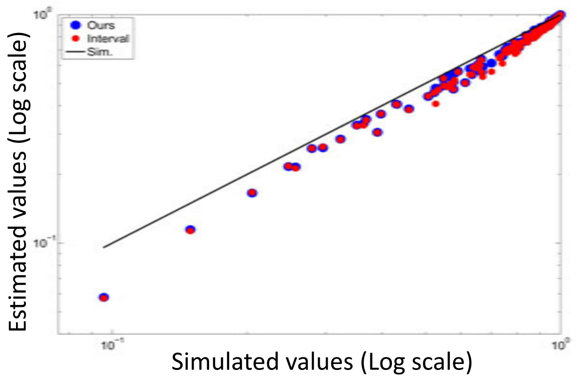


Figure 7.18: Comparison of ER metrics between our approach and the interval-based approach.

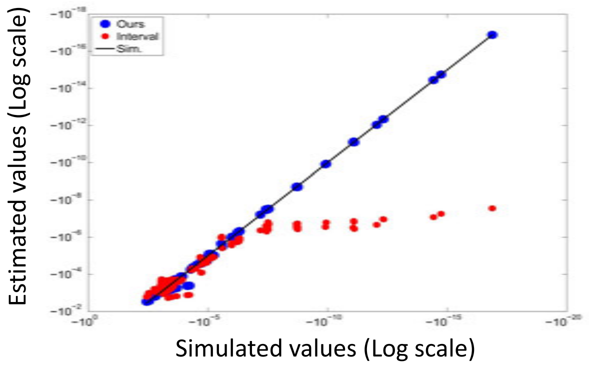


Figure 7.19: Comparison of ES metrics between our approach and the interval-based approach.

approach. Figure 7.20(b) shows inaccuracy results. Our approach demonstrates improved accuracy compared to the interval-based approach, especially for the ES metric. Note that when the number of nodes is small (the left side of the figure), the magnitude of estimation errors tends to be large relative to the magnitude of data, and the inaccuracy of the interval-based approach is very high due to the saturation issue. Our approach reduces inaccuracy by 3.75$\times$ compared to the interval-based approach.

Randomly generated topologies. To study the accuracy of EM estimation with respect to the size and topology of testcases, we use randomly generated testcases as in [126]. We use the following three components to generate the random testcases; (i) primary inputs (PI) with different standard deviations, (ii) adders with different hardware configurations, and (iii) arbitrary connections among adders and constant multipliers. We generate 50 artificial testcases with different numbers of nodes (adders or constant multipliers). The number of nodes ranges from 10 to 30

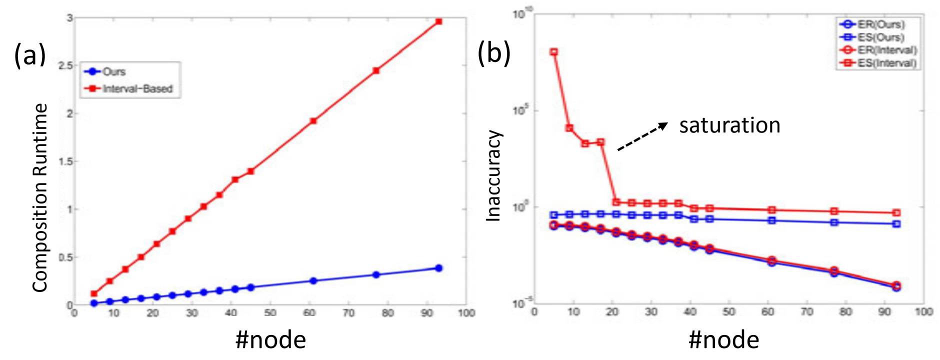
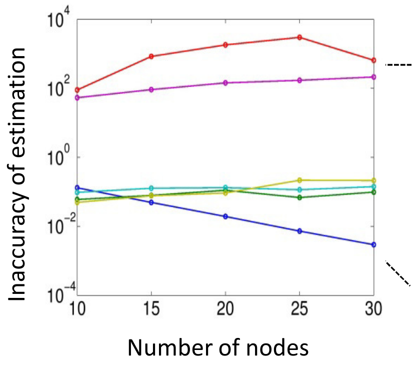


Figure 7.20: Comparison of (a) runtime for error composition and (b) inaccuracy of EM estimation for MAC circuits with different testcase sizes. Our average inaccuracy improvement against the interval-based approach is 3.75× excluding saturation.

with a step size of five, The accuracy results for each EM are plotted in Figure 7.21. We evaluate the estimated results from our approach with the regression coefficients generated from the model in Section 7.2.2. In the plot, inaccuracy results from 10 different topologies are averaged for each circuit size. For randomly generated circuits, we observe that ER, ES, MSE and MAXE show relatively accurate results with 4.18%, 8.30%, 12.2% and 12.9% inaccuracy, respectively. Moreover, the accuracy does not degrade as circuit complexity (number of nodes) increases. The estimates of ARES and SNR are inaccurate (1.28×10^3 and 1.35×10^2). Inaccuracy in these metrics arises because they measure error relative to input data, and accurate estimation is difficult, as we have discussed in Section 7.2.3. Methods that would accurately handle their composition are obvious directions for our future work.



EMs	min.	avg.	max.
ARES	2.00E+00	1.28E+03	8.97E+01
SNR	2.00E+00	1.35E+02	5.36E+01
MAXE	4.92E-02	1.29E-01	2.00E+00
MSE	9.61E-02	1.22E-01	2.00E+00
ES	5.97E-02	8.30E-02	2.00E+00
ER	1.31E-01	4.18E-02	2.00E+00

Minimun/average/maximum inaccuracy for #nodes = 30

Figure 7.21: Comparison of inaccuracy with respect to the number of nodes in randomly generated circuits.

7.3 Conclusions and Future Directions

In the first part of this chapter, we propose an accuracy-configurable approximate (ACA) adder for which the accuracy of results is configurable during runtime. Due to its configurability, the ACA adder can operate adaptively in both approximate (inaccurate) mode and accurate mode. To quantify the accuracy in approximate computation, we provide two metrics for amplitude data and information data. We compare the ACA adder against previous approximate adders based on the proposed metrics. The ACA adder shows high accuracy with respect to the metrics, and can provide up to 24.6% throughput improvement and 37.0% power reduction over the conventional CLA adder. The ACA adder can also be used in accuracy-configurable applications with pipelining. We demonstrate that the ACA adder can provide approximately 30% power reduction under a relaxed accuracy requirement versus the conventional pipelined adder. Finally, we show that our ACA adder can improve the achievable tradeoff between performance, power and quality for given accuracy requirements.

In the second part of this chapter, we propose an improved approach to estimating the output quality of approximate designs. Our LUT-based approach characterizes the statistical properties of approximate hardware modules and a regression-based technique improves the accuracy of EM estimation. With our composition approach, we achieve 1.36× and 8.4× runtime improvements for library characterization and error composition, respectively. We also achieve 3.75× accuracy improvement for ES compared to previous works [116] [117] on a set of MAC circuits. We also demonstrate that our approach is applicable to general designs using the randomly generated testcases with up to 30 nodes in the configuration.

For the accuracy-configurable design, our ongoing work seeks to implement other arithmetic components such as multipliers, multi-input adders, etc. More broadly, our research addresses additional aspects of (runtime) accuracy-configurable systems and applications. For the EM estimation, we will improve the accuracy of estimation for relative error metrics (e.g., ARES and SNR). To improve regression accuracy, we plan to include topological information of circuits in the model. We will also extend our approach to other approximate modules, including multipliers. In addition, we are working to develop a synthesis flow for approximate circuits using our EM estimation approach. We further anticipate broadening our current studies to include more approximate arithmetic units and different input distributions. Currently, assume that the input distributions are given; however, the distributions of inputs change in different applications. Our follow-on work will seek approaches that track the change of input distributions and adaptively reconfigure the hardware in order to maintain the error metric requirements.

7.4 Acknowledgments

Chapter 7 is in part a reprint of “Accuracy-Configurable Adder for Approximate Arithmetic Designs”, *Proc. ACM/IEEE Design Automation Conference*, 2012 and “Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits”, *Proc. IEEE International Conference on Computer Design*, 2013, to appear.

I would like to thank my coauthors Wei-Ting Chan, Professor Andrew B. Kahng, Professor Rakesh Kumar and Professor John Sartori.

Chapter 8

Memory Access Power Gating in Modern Systems

During every cycle that a core is on, even when stalled, leakage power is consumed via gate leakage, gate-induced drain leakage, junction leakage, and subthreshold leakage. A core may stall quite often if it is intensively accessing the memory subsystem, because every time a thread makes a memory request that misses the L1 cache, the core is subjected to a variable access latency. This variable access latency often translates into a core stall during which no forward thread progress occurs, and energy is wasted. For a $32nm$ out-of-order EV6 core, stall energy can be up to 39.1% of total energy consumption for the SPEC 2006 benchmarks [162].

Previous works have reduced core energy waste by lowering core frequency and voltage (i.e., DVFS) for memory-intensive threads when directed by L2-cache misses [79] [80] [120]. Some schemes can even direct core DVFS behavior based on signals from the L2-cache and estimates of instruction-level parallelism [159]. A slower core frequency results in fewer cycles waiting for the memory subsystem. Scaling down both frequency and voltage results in an estimated cubic dynamic power and quadratic leakage power savings [119]. However, the inability to scale device threshold voltages, coupled with aggressive scaling of supply voltages (subject to overdrive and performance requirements), means that cores have little room to reduce voltage during DVFS [206]. The net outcome is decreased energy savings from DVFS, which motivates the development of new techniques to reduce core energy consumption while waiting for the memory subsystem.

Power gating is a technique that drastically reduces leakage power by cutting off the current path from supply to ground through introduction of a transistor switch between them.

At one end of the spectrum, functional unit power gating reduces power consumption of unused core functional units [238] with wake-up latencies of several nanoseconds. At the other end, entire cores may be power gated and woken up, with latencies of several tens of microseconds to account for saving and restoring all core state from memory [200]. An intermediate mechanism, Memory Access Power Gating [122], provides the ability to power gate an entire core, wake up a power-gated core in about $10ns$, and maintain the core's architectural and cache state. This mechanism uses a combination of a *programmable power gating switch* (PPGS), state retention cells, and source biasing to enable the core to efficiently enter and exit a power-gated state.

In this chapter, we extend the analysis of the technique, *Token-Based Adaptive Power Gating* (TAP) [131]. TAP deterministically applies power gating during core stalls which are caused by the variable latency of requests to the memory subsystem. TAP achieves this by providing the capability to track every ongoing memory request and the expected response time for each memory access that misses in the L1 cache. An expected lower bound on latency is sent to each core's PPGS by modifying the cache controllers to send a token on any miss where the token includes an estimate of the access latency of a next-level memory hit. The result is that TAP can support power gating with no performance loss.

Our work makes the following contributions:

- We introduce a distributed Wake-up Controller to control core wake-up mode in many-core designs.
- We analyze TAP's energy savings for in-order cores to achieve predictions of energy savings for an arbitrary memory hierarchy and application, with 0.82% average error and 9.75% maximum errors.
- We compute break-even times of $8.53ns$ and $17.17ns$ for in-order and out-of-order cores, respectively.
- We decompose TAP behavior to determine the function of time spent power gating, waking up the core, restoring core state, and overhead to show that core wake-up and restore time averages 1.9% of execution time.
- We demonstrate that TAP can adapt to an increase in memory contention by increasing power-gated time by $3.69\times$ as the number of threads increases from 1 to 32.
- We design and implement a *staggered wake-up* scheme capable of reducing wake-up latency by up to 58.2%; this results in a 3.14% increase in energy savings for TAP.

The remainder of this chapter is organized as follows: Section 8.1 presents our power-gating and power distribution network analysis; Section 8.2 describes the distributed TAP power gating system and analyzes energy savings for an in-order core; Section 8.3 lays out our methodology for experiments; Section 8.4 presents experimental results; and Section 8.5 concludes with a summary of the results and possible future directions.

8.1 Power Gating and Power Distribution Network Analysis

This section provides a low-level analysis of our power gating methodology and its impact on the power distribution network. Section 8.1.1 gives the details of the programmable power gating switch. Section 8.1.2 describes our models for capacitance of a core and voltage noise in the power distribution network (PDN). Section 8.1.3 explains how we model core wake-up mode constraints and the benefit of a *staggered* wake-up.

8.1.1 Programmable Power Gating Switch (PPGS) Design

As noted above, power gating cuts off leakage current paths between supply (Vdd_core) and ground (Vss) by using switch transistors (often, high-V_t or long-channel devices). A typical power gating methodology with *header* switches is illustrated in Figure 8.1. When the pg_enable signal goes low, the header switches turn off and leakage current is reduced. While in the power-gated state, all logic gates connected to the virtual supply (Vdd_int) lose their logical states. Setting the pg_enable signal to high resumes circuit operation after a delay that corresponds to charging circuit capacitive loads, resetting memory elements, and restoring state from retention flip-flops connected to Vdd_core.

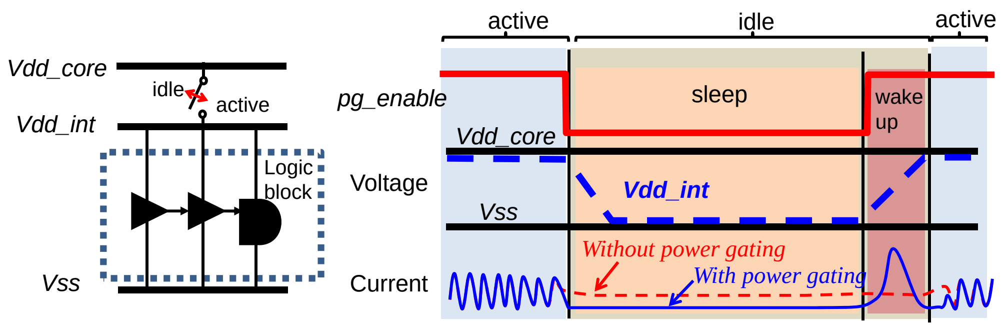


Figure 8.1: Operation of the power gating technique.

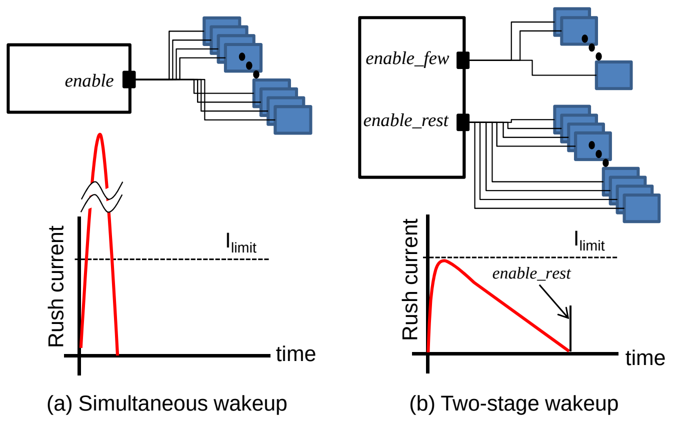


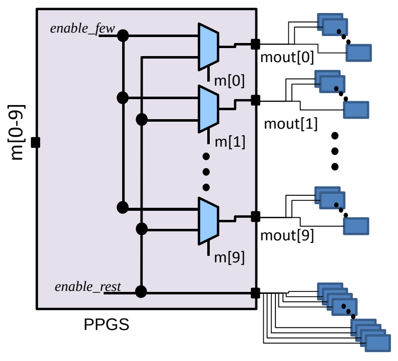
Figure 8.2: Wake-up current profiles with different wake-up controls.

The delay to charge circuit capacitive elements is a function of total design charge (Q) and peak charging current (I_{limit}). If all header switches turn on simultaneously, a large "inrush" current charges internal nodes in minimal time. To satisfy inrush current upper limits (too-large IR drop can affect functionality of neighboring active blocks), header switches are partially turned on in sequence, which increases charging time to at least $T_{charge} = Q/I_{limit}$. Minimal charging time is achieved with a rectangular current profile, but such a profile requires very fine-grained control of header switches. To avoid this design complexity, we use a two-stage wake-up control [107] where the first stage (*enable_few* signal) turns on header switches to allow I_{limit} charge current. The remaining header switches are turned on in the second stage (*enable_rest* signal) once the circuit nodes are nearly charged, resulting in a triangular charging current profile (see Figure 8.2(b)). This increases the wake-up latency to at least twice the minimum square wake-up profile, but simplifies signal connections.

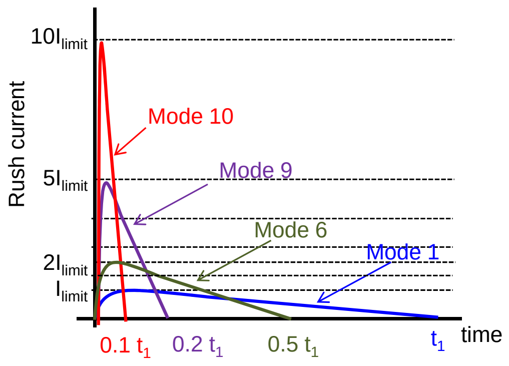
To maximize opportunities for power gating subject to wake-up inrush current and supply noise constraints, we seek to enable multiple *wake-up modes*, with a range of wake-up latencies, per core. Figure 8.3 shows our *programmable power gating switch* (PPGS) for a core, along with the wake-up current profile for different wake-up modes. We configure the number of first-stage wake-up switches to control the inrush current as shown in Figure 8.3(b). With the dynamic configuration of the PPGS, we can minimize the wake-up time according to the core configurations — e.g., the number or location of active cores relative to the waking-up cores. To power gate a core, all mode selection signals $m[0-9]$ are set to one, which turns off all switches at the same time.[23]

[23]Due to the large resistance of off-state switches, inrush current from simultaneous turn off is negligibly small compared to wake-up inrush current.

Core wake-up time and inrush current are determined by the mode selection. For example, Mode 1, which has the longest wake-up time and smallest inrush current, is set by $m[0]$ = 0 and $m[1-9]$ = 1. Thus, $m[0]$ is enabled by signal *enable_few* and $m[1-9]$ is enabled by signal *enable_rest*. Mode 2 is set by $m[0-1]$ = 0 and $m[2-9]$ = 1; inrush current increases with the number of first-stage switches, while wake-up time decreases, as shown in Figure 8.3(b). The other modes can be set similarly.



(a)



(b)

Figure 8.3: (a) PPGS design and (b) inrush current profiles for each wake-up mode.

8.1.2 PDN Model for Circuit Analysis

Table 8.1 shows estimated design parameters, power-gating results and PDN-model parameters for $32nm$ and $22nm$ cores with high performance (HP) and low-operating power (LOP) devices. To study wake-up latency and inrush current, we estimate the total charge for core logic and interconnect capacitance as $Q_{core} = (C_{logic} + C_{int})\,Vdd\_core$, where Q_{core}, C_{logic}, and C_{int} represent total charge, device capacitance, and interconnect capacitance for a single core without caches. We estimate a core's total transistor count using McPAT [162] to determine the core's area and average transistor density. Based on this transistor count and parameters from the 2009-2010 *International Technology Roadmap for Semiconductors* (ITRS) [16], we estimate C_{logic} and C_{int}. The inrush current limit (I_{limit}) and on-current (I_{active}) are estimated from McPAT data for peak power and average power, respectively.

From the calculated charge (Q_{core}), the minimum wake-up latency with a rectangular-form current profile is $T_{min-charge} = Q_{core}/I_{limit}$ and the minimum two-stage wake-up latency (Figure 8.2(b)) is $2 \times T_{min-charge}$.

We estimate leakage power consumption during power gating of the core logic and SRAM, as follows. For the core logic, leakage from retention registers and header switches must be taken into consideration. We assume that (live-slave type) retention flip-flops have 20% more leakage power than normal flip-flops during power gating [136]. For SRAM, we assume that the (separate) SRAM supply voltage is scaled using source biasing, and we estimate leakage based on [195].

Table 8.1: Estimated data of $32nm$ HP, LOP and $22nm$ HP, LOP cores.

estimated data	$32nm$ HP	$32nm$ LOP	$22nm$ HP	$22nm$ LOP
design data				
Vdd_core (V)	1.00	0.77	1.00	0.77
core area (mm^2)	4.593	4.608	2.701	3.657
logic area (mm^2)	2.891	2.863	1.635	1.636
C_{core} (F)	7.53E-9	7.48E-9	4.58E-9	4.58E-9
total charge (C)	7.53E-9	5.76E-9	4.26E-9	3.30E-9
core leakage (W)	0.355	0.042	0.147	0.019
I_{active} (A)	0.725	0.374	0.371	0.233
I_{limit} (A)	1.298	0.674	0.701	0.632
power gating and wake-up				
$T_{min-charge}$ (ns)	5.08	7.36	6.40	6.55
wake-up energy (pJ)	3.30E+3	1.91E+3	2.24E+3	1.60E+3
# of header switches	9,664	6,222	5,516	5,127
leakage in PG state (W)	8.03E-3	7.14E-4	3.37E-3	3.59E-4
leakage reduction in PG	97.74%	98.29%	97.71%	98.12%
PDN model				
# of bumps	45	45	95	95
R_{shared} (Ω)	0.01	0.01	0.01	0.01
$L_{pkg-core}$ (nH)	7.69E-4	7.76E-4	6.44E-4	6.44E-4
$R_{pkg-core}$ (Ω)	1.54E-5	1.55E-5	1.29E-5	1.29E-5
C_{decap} (F)	1.51E-9	1.50E-9	9.16E-10	9.16E-10
R_{PDN} (Ω)	0.07	0.10	0.12	0.15

Following the methodology of previous works [107] [114] [141], we construct a detailed PDN model that includes package parasitics to enable realistic noise analysis under various wake-up scenarios. Power is delivered from an external voltage regulator module (VRM) through a printed circuit board (PCB), a package ball, package interconnect, microbumps, on-die redistribution layers, the on-chip PDN, and power-gating switches. We model the entire power delivery network including power-gating switches as a simplified RLC circuit as shown in Figure 8.4. Package inductance and series resistance from VRM to bumps for a core are lumped as in-series inductance and resistance.[24] The PDN in package shared by multiple cores is represented as a resistance mesh with a branch resistance of R_{shared}. There are three variant models depending on the state of the core — core in active mode, core being woken up, and core in sleep mode (see Figure 8.4). On-chip decoupling capacitance C_{decap} is assumed to be 20% of C_{core} as in Huang et al. [114].

PDN parameter values in Table 8.1 are from personal communication with industry experts [81] and reflect production designs at the 28nm foundry half-node. Bump density is assumed to be 45 bumps per mm^2, and the number of bumps is then calculated from logic area (I/O signals are peripherally located in the SoC die plan). The package inductance and resistance to a bump are respectively assumed to be 0.05nH and 1$m\Omega$ based on empirical data. The lumped package inductance $L_{pkg-core}$ and resistance $R_{pkg-core}$ for a single core are respectively calculated as L_{pkg}/N_{bump} and R_{pkg}/N_{bump}, where N_{bump} is the number of bumps.

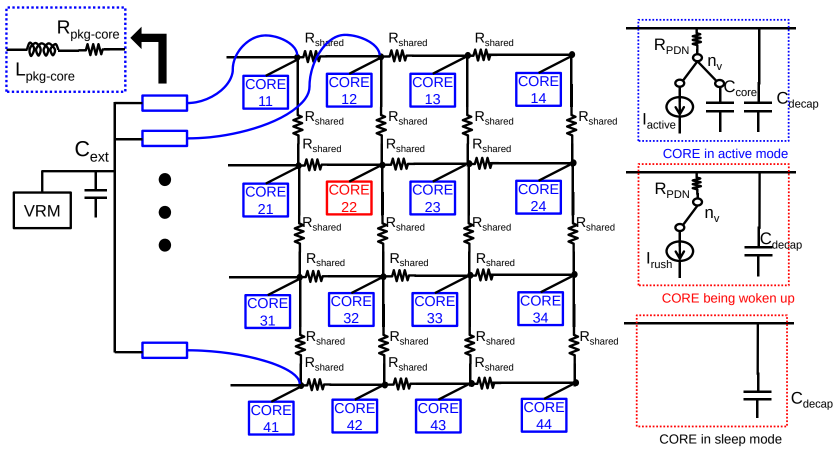


Figure 8.4: 16-core system power delivery network with power gating.

We measure the Vdd_core and Vdd_int voltages of all cores using HSPICE [27]. We vary the number of cores being woken up, and search over all configurations of woken-up and active cores. For each configuration, we find the minimum wake-up latency that satisfies two IR drop constraints: (a) Vdd_int of active cores should drop by no more than 5% and (b) Vdd_core of standby cores should drop by no more than 40% so as to retain data in retention circuits [81].

[24]Note that we do not model inductance of the on-chip power mesh. High-frequency effects are not relevant to the wake-up current analysis, and wire dimensions are such that resistive impedance dominates. To our knowledge, our approach matches that used in advanced SoC signoff methodologies today.

8.1.3 Safe Wake-up Mode Analysis and Equation

Previous PPGS work [122] selected a wake-up mode based on the worst-case wake-up time for each *number* of idle cores. The worst-case wake-up time assumption limits the benefit of power gating. A core's minimum wake-up time is constrained by the voltage noise seen by neighboring active cores – in particular, some *critical* active neighbor core where the voltage noise constraint is first violated. The voltage noise of an active core is mainly affected by adjacent woken-up cores and the latencies (i.e., associated inrush currents) with which they wake up. In other words, we may exploit knowledge of cores' locations to reduce pessimism. We have developed a model that determines the minimum wake-up time based on the number and location of active and woken-up cores. To simplify the model, we assume that all woken-up cores have the same (uniform) wake-up latency, but in principle our methodology easily extends to non-uniform cores and wake-up latencies.

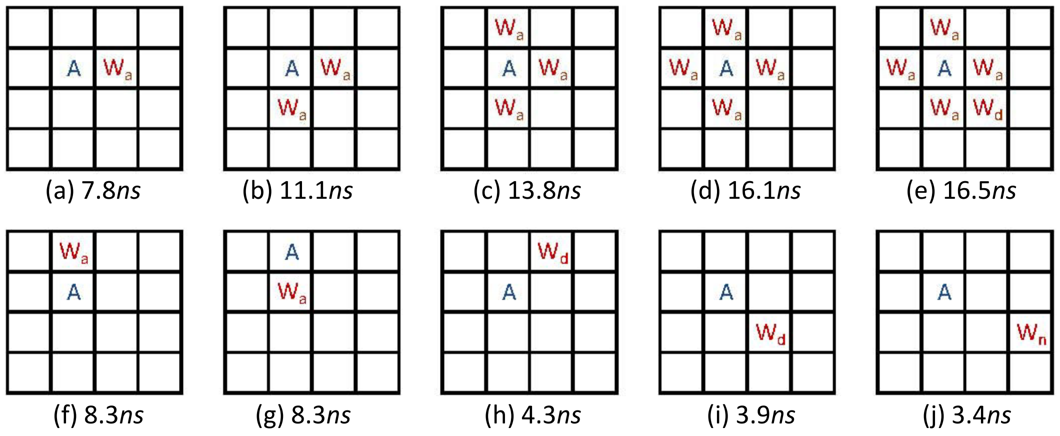


Figure 8.5: SPICE-calculated minimum wake-up latency for an EV6 16-core CMP with various wake-up scenarios.

Figure 8.5 shows the minimum wake-up time according to the location and status of cores for an example case of an EV6 16-core CMP. In the figures, A denotes the critical active core, W_a are adjacent woken-up cores, W_d are diagonally adjacent woken-up cores, W_n are non-adjacent woken-up cores, and blank squares are idle or non-critical active cores. The wake-up latency increases approximately as the square root of the number of adjacent woken-up cores (Figure 8.5 (a) – (e)). Woken-up cores in the diagonal adjacent (W_d) or non-adjacent positions impact wake-up latency less than adjacent woken-up cores (Figure 8.5 (f) and (g)). Cores located at an edge position (Figure 8.5 (h)) experience increased minimum wake-up latency.

From such observations, we have modeled the minimum wake-up latency based on the core status at each location as:

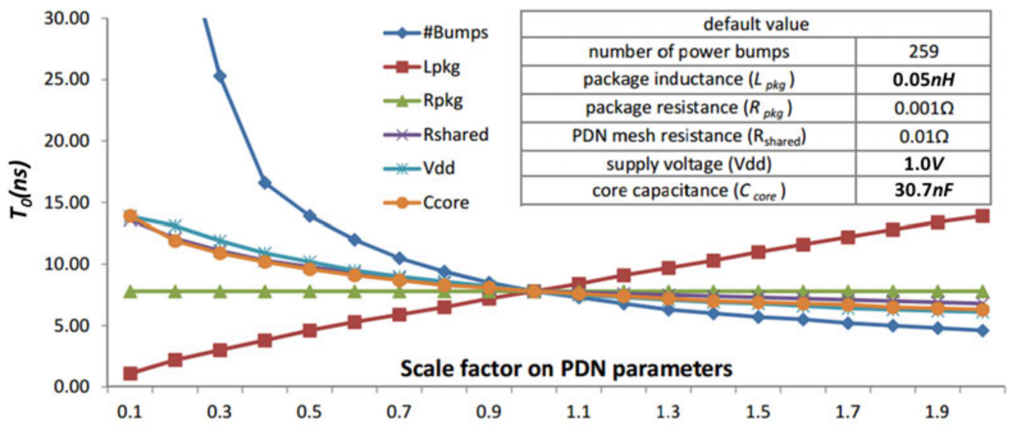
$$T = T_0(w + \beta \times x + \gamma \times y + \delta \times z)^{\alpha} \tag{8.1}$$

where T_0, α, β, γ and δ are fitting coefficients, w is the number of adjacent woken-up cores, x is the number of diagonally adjacent woken-up cores, y is the number of other (non-adjacent) woken-up cores, and z is the number of active or adjacent woken-up cores located at the edge.

We have verified our model with SPICE and modeled the wake-up times for 4-, 6-, 8-, and 16-core CMPs for all location permutations. Table 8.2 shows the results. Our model has an average error of 2.64%, 1.93%, 2.31% and 1.57% for 4-, 6-, 8-, and 16-core CMP cases, respectively.

Table 8.2: Average and maximum error of the modeled wake-up time for 4-, 6-, 8-, and 16-core cases (EV6, 32nm HP).

core	coefficient					error	
	T_0	α	β	γ	δ	average (%)	maximum (ns)
4-core	7.9	0.50	0.35	0.15	0.15	2.64	0.37
6-core	7.9	0.50	0.35	0.15	0.13	1.93	1.10
8-core	7.9	0.50	0.30	0.15	0.13	2.31	1.65
16-core	7.9	0.50	0.20	0.10	0.10	1.57	1.40



default value	
number of power bumps	259
package inductance (L_{pkg})	**0.05*nH***
package resistance (R_{pkg})	0.001Ω
PDN mesh resistance (R_{shared})	0.01Ω
supply voltage (Vdd)	**1.0*V***
core capacitance (C_{core})	**30.7*nF***

Figure 8.6: T_0 as a function of PDN parameters.

The results of the SPICE simulation have varying degrees of sensitivity to power distribution network (PDN) parameters - the number of bumps, package inductance (L_{pkg}), package resistance (R_{pkg}), PDN mesh resistance (R_{shared}), supply voltage and core capacitance. We have assessed the minimum wake-up time sensitivity to variations in the PDN model. Figure 8.6 shows the change in the T_0 coefficient when each PDN parameter is scaled by factors from 0.1$\times$ to 2$\times$ (a 20$\times$ range!) with respect to our default values, which are obtained from personal communication with industry experts. Since the actual wake-up latency depends on PDN variations, the T_0 coefficient will be determined by testing the actual packaged chip. It is important to note that our conclusions regarding energy savings and overheads remain qualitatively the same across the range of PDN parameter values – i.e., our conclusions are quite robust to the PDN design choices.

8.1.4 Core Wake-up Stagger

In the above wake-up analysis, we assume that all cores wake up simultaneously which is the worst case. However, wake-up latency is significantly reduced when we *stagger* the wake-up sequence so that two cores wake up at slightly different times (e.g., offset by $1ns$). We design the wake-up controller to insert *stagger* between waking cores to reduce wake-up latency. Figure 8.7 shows minimum wake-up latency for an EV6 16-core CMP when we add stagger between woken-up cores. The minimum wake-up time (y-axis) is reported for the worst case for each number of woken-up cores (x-axis). When stagger is zero, wake-up time increases according to the number of woken-up cores. However, if we avoid simultaneous wake-up, minimum wake-up time reduces greatly. When two, three and four cores are waking up within an interval of three cycles ($0.9ns$), we obtain 18.8%, 31.9% and 40.3% wake-up latency reductions, respectively, over simultaneous wake-up. From SPICE results in Figure 8.7, we can see that the minimum wake-up time does not increase with staggered wake-up when the number of woken-up cores is larger than four. We have modeled the minimum wake-up time with Equation (8.1) for up to three woken-up cores by changing the parameter α from Table 8.2. The dotted lines in Figure 8.7 show the modeled wake-up latency from Equation (8.1) and its error with respect to SPICE simulation. Our measurements of model accuracy show an average (maximum) error of 2.66% (7.62%), 1.89% (6.61%), 0.93% (3.59%) and 2.51% (3.08%) for the 4-, 6-, 8-, and 16-core CMP cases, respectively.

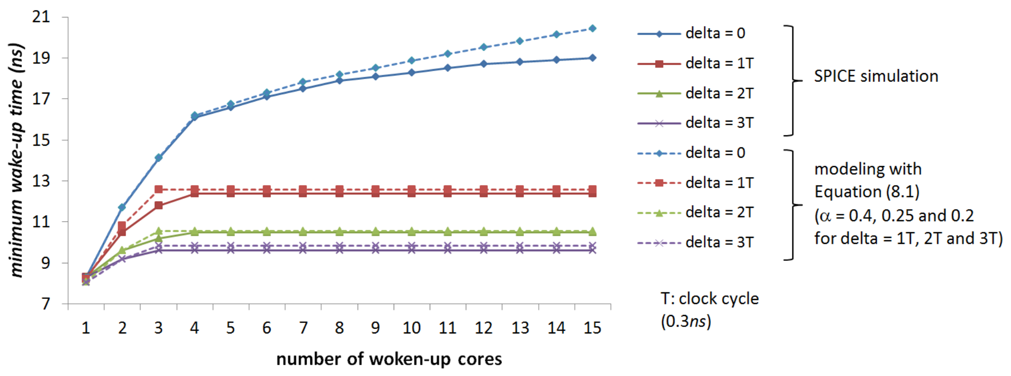


Figure 8.7: Minimum wake-up latency versus wake-up stagger.

8.2 System Design

We now present our architectural modifications used to control power gating for each core. A memory access power-gating controller must provide three functions. First, the controller ensures that each core's PPGS uses a wake-up mode that does not violate supply voltage noise constraints of the system when waking up a core. Second, the controller should be able to predict the expected duration of core stalls. Last, the controller must retain essential core architectural and performance related state. Together, these functions allow for energy savings and minimal performance hit without violating voltage noise constraints. The rest of this section describes how TAP provides these three functions.

8.2.1 Wake-up Controller (WUC)

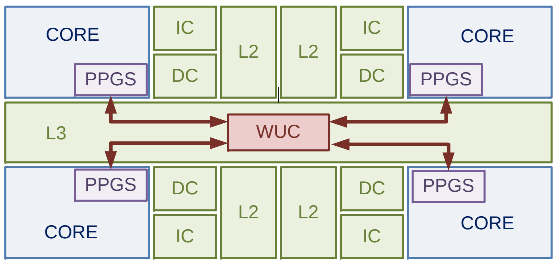


Figure 8.8: WUC and PPGS integration into a 4-core CMP.

The WUC is a centrally located wake-up controller (see Figure 8.8) that listens in on the cache interconnect and orchestrates the assignment of wake-up modes to core PPGSs. The WUC maintains a lookup table that maps the possible values of variables w, x, y and z from

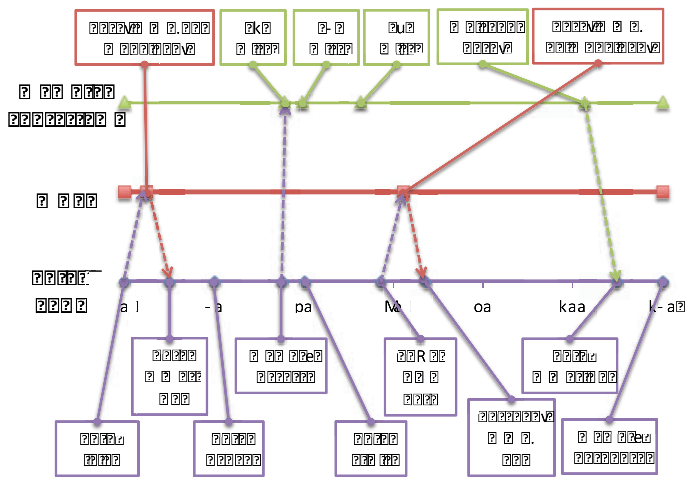


Figure 8.9: WUC, Core$_i$ PPGS, and memory subsystem timing diagram.

Equation (8.1) to safe wake-up modes. For the 16-core case, the WUC requires 12 $\times$ 1000 bits of (SRAM) registers to hold all entries. In addition, the WUC maintains the status of each core (idle, active, power gated, or waking-up) to determine the entry to lookup for a new request.

Figure 8.9 shows how a core wakes from an idle state, power gates during a stall, and then wakes up again via communication with the WUC. At time $0ns$, the core is idle and power-gated off. The core wakes up by its PPGS requesting a worst-case wake-up mode that it may assume is always safe to use (provided it notifies the WUC). At $5ns$, the WUC receives a request, looks up a safe wake-up mode in its table based on the system state, and returns that mode to the core PPGS. The PPGS wakes up the core and the core executes code. At time $35.5ns$, the core attempts to access the memory subsystem which causes a stall at $40.5ns$. At $57.5ns$, the core PPGS detects a core stall dependent on a memory miss and then power gates the core. At the same time, the PPGS requests a lower-latency wake-up mode from the WUC in hopes of power gating for longer. The WUC receives this wake-up mode request at $62.5ns$ causing the WUC to update the state of the core. Should there be no conflicting wake-ups, the WUC may issue a one-use lower-latency wake-up mode to the requesting core. The core PPGS receives this response at $67.5ns$ and may reschedule the wake-up time of the core. The PPGS wakes up its core at $110ns$ for the memory response at $120.5ns$.

For a large multi-core system (e.g., 64 cores), a given core's PPGS may not tolerate the latency to communicate with a centralized WUC due to propagation and queuing delay across the chip. However, we observe that non-adjacent cores do not significantly affect core wake-up latency, and with proper guardband, only adjacent cores need be considered.

This observation motivates a distributed design which assigns each core to a recurring wake-up slot. In this scheme, each core is given a recurring slot at which it can start waking up. To avoid any performance hit, a core should select a slot before the deadline to start waking up. The average wake-up delay for a core is defined by two degrees of freedom: the number of unique wake-up slots, η, and the stagger between two adjacent wake-up slots, ψ. The worst-case reduction in power-gated time occurs when a core predicts that it would need to wake up ϵ seconds before its assigned slot such that $\epsilon < \psi$, causing the core to wake up $\eta \times \psi - \epsilon$ seconds earlier. Given a core that wakes up at any time, uniformly at random, the average expected reduction in power-gated time is $\frac{\eta \times \psi}{2}$.

η and ψ should be chosen to maximize a core's power-gated time and the minimal safe wake-up latency of the core. In particular, given η, wake-up slots should be assigned to cores to minimize the number of adjacent woken-up cores. Figure 8.10 shows three ways of assigning wake-up modes to cores such that the number of adjacent woken-up cores is minimized with preference given to cores waking up simultaneously in the diagonal position. An increase in η and ψ acts to reduce the maximum number of simultaneous core wake-ups and reduce minimal safe wake-up latency. At the same time, increasing those two parameters increases average expected reduction in power-gated time. According to our simulations, system energy savings are maximized when η and ψ equal 5 (no simultaneous wake-ups) and 0.9ns, respectively. This setting results in a 10.3ns minimal wake-up latency and 2.25$ns \pm$ 1.30ns average reduction in power-gated time per core power-gating opportunity, compared to 9.6ns with 0.9ns stagger for an ideal centralized WUC. Further, our simulations show that the distributed WUC has a maximum decrease in power-gated time of 1.61% (0.40% on average) for the benchmark *mcf*.

0	1	2	3	4	…
3	4	5	6	7	…
6	7	8	0	1	…
0	1	2	3	4	…
…	…	…	…	…	…

(a) η = 9

0	1	2	3	4	…
2	3	4	0	1	…
4	0	1	2	3	…
1	2	3	4	0	…
…	…	…	…	…	…

(b) η = 5

0	1	2	0	1	…
2	1	0	2	1	…
1	2	0	1	2	…
0	2	1	0	2	…
…	…	…	…	…	…

(c) η = 3

Figure 8.10: Wake-up slot assignments with different number of slots (η).

8.2.2 TAP: Token-Based Adaptive Power Gating

TAP informs each PPGS about expected memory latency by modifying the cache controllers to send tokens on cache misses that include an estimate of the lower-bound access latency of a next-level memory hit derived from Table 8.3 and a time stamp of creation.[25] The controllers send the tokens to the PPGS of the core that requested the memory access. Once the PPGS receives the token, it looks at the lower-bound latency to satisfy the request and power gates the core if the core is both stalled and idle long enough to save energy. Should the core receive more than one token for simultaneous memory requests, it will track each expected response separately and schedule the resumption of core execution to satisfy the earliest response. If a token is delayed in the memory subsystem by a controller or queue, the PPGS can compare its arrival time with its generation time stamp and previous tokens to determine whether the token should be ignored.

Whenever a memory request misses all the way to the memory controller, the response latency experiences a significant amount of variability. This variability is caused by the complexity of DRAM memory [245], which includes bank queues, availability of the data in the row-buffer, writing wrong address row-buffers, accessing the column in the row-buffer, and channel contention between banks. TAP adapts to memory variability by adding a special token. As soon as the last-level cache experiences a miss, a token is sent to the requesting core's PPGS with an estimated completion time of *UNKNOWN*. This is a directive to the PPGS to start power gating its core immediately and to expect one additional token with the estimated time of arrival (ETA) of the memory response. Once the memory controller submits the memory access to one of the banks and determines whether the access is a row-buffer hit or miss, it sends the second ETA token to the core's PPGS with the ETA of the response assuming that there is no memory channel contention. The PPGS receives the second token before the response and schedules core wake-up for the appropriate time.

Figure 8.11 shows a timing-accurate diagram of a PPGS power gating the core in response to messages from the TAP technique. At time $0ns$, a memory request occurs that will miss in the cache hierarchy and cause a memory access. The PPGS then receives tokens for the L1, L2 and L3 misses. Just after receiving the L2 token, the core stalls due to a dependency. After the L3 token is received, the PPGS decides to power gate the core and saves all core state. The core is then power gated and the memory controller (MC) sends a updated ETA for the

[25] A cache controller sitting on the core side of a shared NUCA cache would require a per-bank lower-bound access latency.

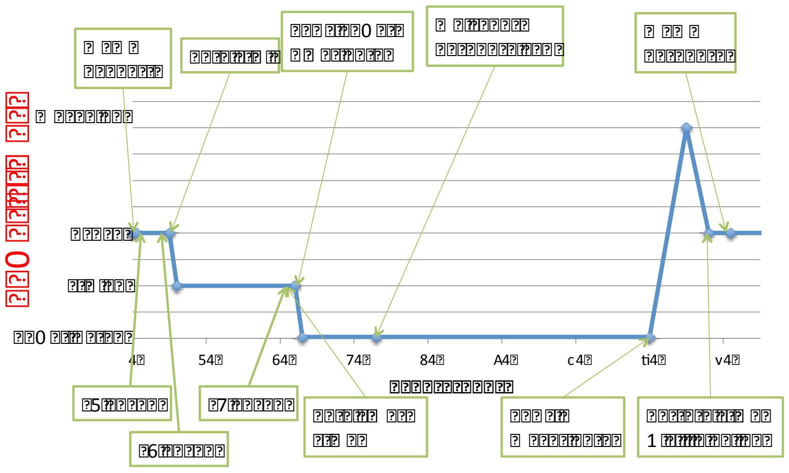


Figure 8.11: Power states (power gated, stalled, active, woken up) as the PPGS power gates the core on a memory access.

memory response. At 70ns, the PPGS begins waking up the core. At 78ns, the core state is restored and the pipeline is restarted. The memory response comes back at 81ns and the core resumes execution as if nothing happened.

The benefit of TAP is that core-level power gating can be directed by system-level information about the memory subsystem. This information represents lower-bound estimates of when a memory response can arrive. Because TAP operates on lower-bound estimates, it avoids over-prediction of core idle latency and achieves *zero* performance impact. The disadvantage of lower-bound estimates is that TAP misses out on potential power gating time and additional energy savings.

To implement TAP in hardware, additional structures are added to both the memory controller and the core. For each bank of each rank of memory, we add a 15-bit delay counter, which indicates the soonest time at which the bank would go idle, and is capable of tracking up to 3.28μsec at a granularity of 100sec of picoseconds. With each cycle, every memory controller decrements its counter by the number of picoseconds in a memory clock cycle until the counter reaches a minimum of zero. When the memory controller schedules a memory operation to a particular bank, it increments the bank's counter with the lower-bound estimate for the completion of the memory operation. If the memory operation is a row-buffer hit, the bank counter is incremented by the time to issue the command, performs the column address select, and transfers the data across the memory bus. If the memory operation is a row-buffer miss, then the counter is incremented by the time to issue the command, pre-charges the row-buffer, issues the row lookup, performs the column address select on that row, and finally transfers the bytes across the memory bus. The value of the counter is the ETA returned to the core's PPGS by the token sent from the memory controller. To quickly determine a row-buffer hit, a register at each bank maintains the row-buffer address of the last memory access.

Each core also requires additional state to track currently valid tokens. For each unique address memory request that causes a token-generation event, we require 80 bits of storage. The first bit indicates validity of the entry. The next 64 bits contain the physical address of the request. The last 15 bits track the ETA for the given request. In the worst case, we require sufficient entries to track the maximum number of parallel memory requests with unique physical cache line addresses that can issue from a core. For our technique, this number is limited by the number of MSHR (Miss Status Handler Registers) queue entries in the instruction and data caches, which is 20 for our EV6 architecture and 4 for our in-order architecture. However, our simulations show that fewer entries are actually required because long core stalls do not usually occur with a large number of parallel memory requests. For example, the benchmark *astar*, which has little benefit from our techniques, does experience 20 parallel requests while the benchmarks *mcf* and *gobmk*, which benefit the most from our techniques, experience at most 13 parallel memory requests. In any case, we estimate that the support to track tokens at the core adds 1456μm of area overhead per core (0.05% of EV6 area), while the modifications to estimate memory latencies add 677$\mu^2 m$ area overhead per memory rank (0.02% of EV6 area).

8.2.3 Formal Analysis of In-order Core Energy Savings

We now derive the expected energy savings from TAP for an in-order core, to gain intuition regarding how energy savings change with system conditions, and to independently verify our reported energy savings. In the following terms and equations, latencies are in units of seconds, power is in units of watts, and energy is in units of joules. When a core experiences a $level_i$ cache miss, it receives either a token or response from the $level_{i+1}$ cache. The idle period between when the core receives the $level_i$ token and the $level_{i+1}$ token or response can be estimated by Equation (8.2), where $txlat_{L_i,L_{i+1}}$ is a bus (transmit) latency from $level_i$ to $level_{i+1}$ cache miss, and $hitlat_{L_i}$ is a memory hit latency at $level_i$. Packets from both the $level_i$ and $level_{i+1}$ caches need to travel through the same memory hierarchy from $level_i$ upwards to the core. The only difference is that the packet from the $level_{i+1}$ cache travels twice across the interconnect between $level_i$ and $level_{i+1}$, waits for the $level_{i+1}$ cache controller, and waits twice for the $level_i$ cache controller.

$$lat_{\Delta Li\_miss} = txlat_{L_i,L_{i+1}} + hitlat_{L_{i+1}} + txlat_{L_{i+1},L_i} + hitlat_{L_i} \tag{8.2}$$

Because of core wake-up latency, waking the core when it receives the token or response from the $level_i$ cache would incur a significant performance penalty, as this effectively increases the $level_i$ cache miss latency by the core wake-up latency. We avoid this performance overhead by preemptively waking up the core even if there is a miss in the $level_{i+1}$ cache. This reduces the period of energy savings in some cases, but avoids the performance overhead. Further, core wake-up costs energy, which places a constraint on how long the idle period must last to amortize the energy loss from core wake-up.

The core's wake-up event is not the only overhead. When a cache sends a token, it must contend for the CPU-side ports of each cache on the way to the core, traverse the shared interconnect, and wait in any queues to shared resources. These delays can reduce the period over which our technique power gates the core. In summary, we can estimate the energy savings of power gating a $level_i$ cache miss using Equation (8.3). E_{Li_miss} is the energy savings on a $level_i$ cache miss; lat_{Li_miss} is the latency to propagate a request from the core to the $level_{i+1}$ cache and back; lat_{token_i} is the latency to propagate a request from the core to the $level_i$ cache and send a token back to the core; lat_{core_wakeup} is the core wake-up latency; P_{Li_miss} is the core idle power during a $level_i$ cache miss; L_R is the factor of reduction of leakage from power gating; and E_{core_wakeup} is the energy to wake up a core from the power-gated state.

$$\begin{aligned} E_{Li\_miss} &= (lat_{Li\_miss} - lat_{token\_i} - lat_{core\_wakeup}) \\ &\quad \times P_{Li\_miss} \times L_R + E_{core\_wakeup} \\ &\quad + lat_{core\_wakeup} \times P_{Li\_miss} \end{aligned} \tag{8.3}$$

We extend this analysis to estimate the energy savings from token-based power gating for all levels in the cache hierarchy in Equation (8.5). $E_{save\_L_N}$ denotes the energy savings for an N-level cache hierarchy; $\%T_{idle\_Li\_miss}$ is the percent of time the core spent idle waiting for a $level_i$ cache miss (estimated as $MPS_{Li} \times lat_{\Delta Li\_miss}$); E_{Li_miss} is defined in Equation (8.3); MPS_{Li} is the number of $level_i$ cache misses per second; and $\%L$ is the percentage of total active power accounted for by leakage. The denominator is simply the sum of idle and active energy for when no power gating is being used. The numerator is equal to core energy during a $level_i$ cache miss without power gating, minus the energy with power gating summed across all cache levels: i.e., the sum of energy savings for each cache level. The total system time is factored from the numerator and denominator, leaving the percentage of time spent in active execution and waiting for $level_i$ cache misses. We compare energy savings measured from McPAT [162] and M5 [47] with the analytical model shown in Equation (8.5) and see a good match. The average error between M5 and the equation is 0.82% with a maximum error of 9.75% for *lbm*. This model demonstrates that TAP's energy savings is a strong function of core wake-up latency and memory behavior. Further, TAP achieves a maximum energy savings of 25.5% (4.00% on average) for an in-order core.

$$\kappa = \%T_{idle\_Li\_miss} \times P_{Li\_miss} \tag{8.4}$$

$$E_{save\_L_N} = \frac{\sum_{i=1}^{N}(\kappa - E_{Li\_miss} \times MPS_{Li})}{\sum_{i=1}^{N}(\kappa) + \frac{(1 - \sum_{i=1}^{N}(\%T_{idle\_Li\_miss})) \times P_{Li\_miss}}{\%L}} \tag{8.5}$$

8.2.4 Core State Retention and Restoration

To avoid losing core state that is required for correct and efficient execution, essential sequential and SRAM cells must be retained. We use the technique from [122] which replaces a subset of sequential cells with live-slave retention flip-flops [136] which can be triggered to retain their logical values before a power gating action at a cost of 20% increase in area and power versus a normal flip-flop. Only those sequential cells comprising the architectural registers necessary to refill the pipeline are selected, which results in 3.4% area overhead for the processor. SRAM cells are retained through source biasing [195] in which the supply voltage is reduced to 50% of nominal supply voltage so that SRAM leakage is reduced, but logical state is maintained. This technique allows for saving the contents of L1 caches, TLBs, branch predictor state, physical registers, etc. To provide supply power during power gating, a separate non-collapsible voltage domain provides power to the retention flip-flops and SRAM cells. Thus, as the power is gated from combinational logic and non-essential sequential cells, the separate voltage rail provides power to maintain core state. The overhead from multi-power domains and separate voltage rails already exist for power-gating cores today. Figure 8.12 shows an in-order core implementation for the power gating and restoration with retention flip-flops.

Additional cycles are required for the power gating and wake-up sequence, and to account for the time to disable/enable the clock, trigger data retention, refill the pipeline, and de-assert/assert the clamps. We model the entire power down and wake-up sequence as in [122]. For example, to wake-up an EV6 core after signals *enable_few* and *enable_rest* have charged core logic, it takes one cycle to enable the clock signal, one cycle to asynchronously reset logic, one cycle to restore registers from retention flip-flops, and seven cycles to restore the pipeline which takes 3.03ns at 3.3GHz.

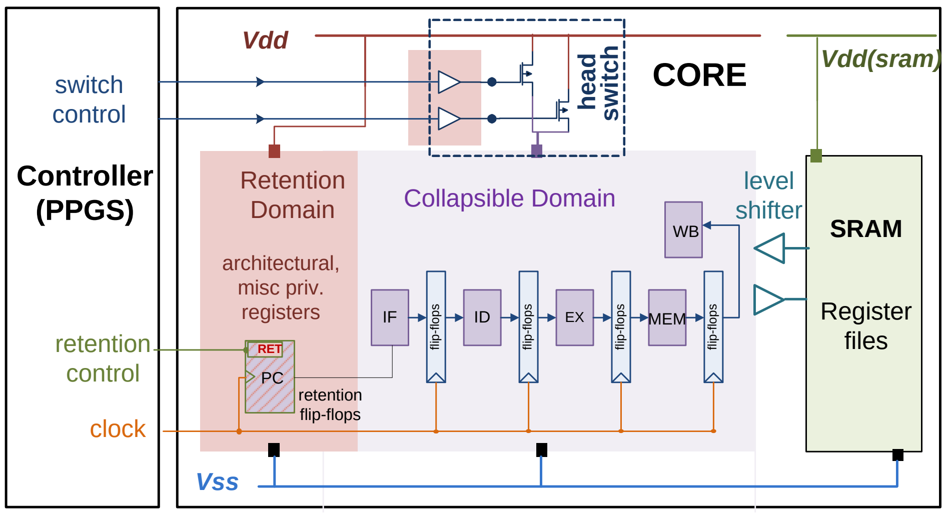


Figure 8.12: Interface for power gating and data retention.

8.3 Simulation Methodology

Table 8.3 summarizes all system parameters in our experiments. The system has 4 cores, each with its own private L1 and L2 caches, and a large shared L3 cache. The L3 cache forwards requests to the memory controller through a shared memory bus. The L1 and L2 cache configurations are 32KB 8-way and 256KB 8-way. The L3 cache is a relatively large 8MB 16-way, which we expect to minimize pressure on the memory subsystem and hence minimize gains we see from our power gating technique. We model an out-of-order core, the DEC Alpha EV6, clocked at 3.3GHz and able to issue six instructions on a cycle.

We simulate the system with the GEM5 simulator [47]. GEM5 is a full-system simulator that can boot an unmodified OS. It features cycle-level models of an out-of-order core, the cache hierarchy, and the interconnect. We integrate GEM5 with DRAMSim2 [8] to provide cycle-level modeling of the memory subsystem including the memory controller, DRAM modules, and shared channels used for communication. We modify GEM5 to support our power gating methodology described in Section 8.2. We simulate our system with 21 of the SPEC 2006 benchmarks using the Simpoint methodology [191] in which 100M-instruction representative regions of execution are determined for each benchmark. To simulate each region, we fast-forward to 100M instructions before the region, warm-up the memory and caches, and perform detailed simulation.

Once simulation is complete, we feed the system configuration and performance counters to McPAT [162] to model power consumption. McPAT is comprised of a power, area, and timing framework that provides off-line power and area estimates for full systems designed in technology nodes between $90nm$ and $16nm$. McPAT generates values for dynamic power, leakage power, peak power, thermal design power, and area. We update McPAT's *technology.cc* file to accurately reflect the ITRS 2010 update report [16].

We compare both techniques to dynamic voltage and frequency scaling (DVFS) via simulation. We calibrate our DVFS settings to match those of [65] for the $32nm$ technology node, in which a 7.5% reduction in voltage follows each 20% reduction in frequency. To direct the DVFS policy, we apply the technique from [80], which uses a cycle-per-instruction based metric, μmean, to detect memory bounded phases of execution. During execution, we sample the application's μmean to determine the most aggressive DVFS setting that may be used to save energy while sustaining at most a 5% performance hit. In addition, we also consider an *oracle* DVFS technique that chooses the DVFS point that results in the lowest energy-delay product (EDP). This technique takes an arbitrary performance hit as long as more energy is saved. For both policies, we model the availability of five DVFS modes which include 100%, 95%, 90%, 80%, and 60% frequency.

Unless otherwise stated, our results assume a $32nm$ HP circuit technology, a $10.2ns$ wake-up mode, and a four-core system that is 50% utilized (two cores idle) which results in conservative energy savings during little memory contention, shorter core-stall durations, and utilization of a slower wake-up mode than if only one core was utilized. The following overheads were considered when calculating the reported results (including the Oracle policy): core wake-up energy, core wake-up delay, core pipeline-refill latency, retention overhead of live-slave retention cells, SRAM leakage during source biasing mode of operation, and voltage noise safety.

Table 8.3: System configuration values.

parameter	value	notes
IO core model	DEC-Alpha EV4	
IO core clock	2.0*GHz*–1.2*GHz*	
IO execution	2-way in-order	
EV6 core clock	3.3*GHz*–1.9*GHz*	
EV6 execution	6-way out-of-order	
EV6 functional units	6ALU, 2IMULT, 2FPALU	
Icache/Dcache	32*KB* 8-way one cycle	
L2 cache	256*KB* 8-way 4*ns*	Private per core
L3 cache	8*MB* 16-way 13ns	Shared
Core-to-L1 token latency	0.5ns	controller delays
Core-to-L2 token latency	4.5ns	controller delays
Core-to-L3 token latency	17.5ns	controller delays
Core-to-WUC latency	5ns	controller delays
PPGS wake-up modes	4.5ns–16.9ns	SPICE
IO pipeline refill latency	2ns	4-pipeline stages
IO core wake-up energy (IWE)	4,020pJ	Charge cells
IO leakage power (ILP)	0.486W	McPAT [162]
IO PG leakage reduction (ILPR)	97.74%	[136]
IO PG break-even point	8.53ns	$IEW/(ILPR \times ILP)$
IO DFLT core wake-up latency	8.06ns	SPICE
IO FUPG wake-up energy	1780pJ	McPAT, ITRS [16]
IO FUPG wake-up latency	6.0ns	SPICE
EV6 pipeline refill latency	2.12ns	7 pipeline stages
EV6 core wake-up energy (EWE)	15,358pJ	Charge cells
EV6 leakage power (ELP)	0.916W	McPAT [162]
EV6 PG leakage reduction (ELPR)	97.65%	[136]
EV6 PG break-even point	17.17ns	$EWE/(ELPR \times ELP)$
EV6 DFLT core wake-up latency	10.2ns	SPICE
EV6 FUPG wake-up energy	9641pJ	McPAT, ITRS [16]
EV6 FUPG wake-up latency	6.4ns	SPICE

8.4 Experimental Results

We now analyze TAP to understand its energy savings and how those energy savings depend on system configuration and system memory utilization. We also reveal that staggered wake-ups are an easy way of reducing core wake-up latencies in CMPs. In the following, Section 8.4.1 examines the energy savings of TAP. Section 8.4.2 dissects simulation time of TAP into time spent power gating, waking up, restoring state, executing code, and adding execution overhead. Sections 8.4.3 and 8.4.4 examine energy savings as a function of wake-up latency and memory congestion. Last, Section 8.4.5 examines the energy savings due to staggered wake-up in a CMP with up to 16 cores.

8.4.1 EV6 Power Gating Energy Savings

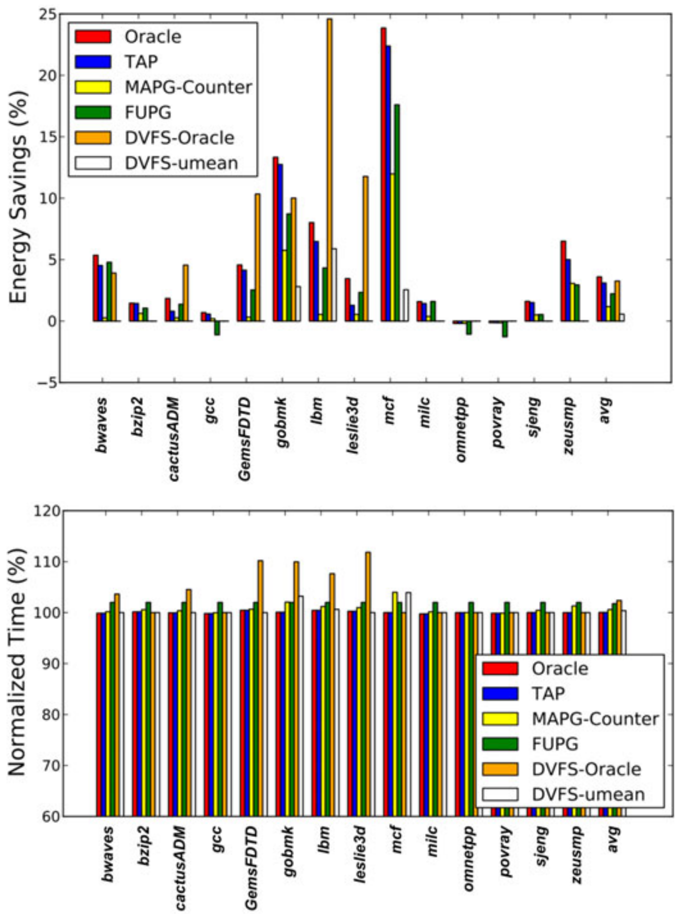


Figure 8.13: Energy savings and performance overhead of power gating Oracle, TAP, FUPG, DVFS-Oracle and DVFS-μmean.

Figure 8.13 compares the energy savings of TAP with the energy savings of an oracle memory predictor (Oracle), Functional Unit Power Gating (FUPG) [113] [167] [170], DVFS-Oracle, and DVFS-μmean.

Oracle-Based Power Gating To understand the limit of energy savings from power gating cores during memory stalls, the oracle memory predictor assumes *a priori* knowledge of all memory accesses and determines the optimal power gating behavior. The EV6 oracle policy achieves a maximum of 23.9% energy savings, and 3.6% energy savings on average. A few benchmarks show negative energy savings as high as -0.2%. These negative energy savings are caused by the lack of power gating opportunities and the retention cells' power overhead on CPU-bound benchmarks.

TAP In comparison with the Oracle, TAP must determine memory latencies in a running system to ensure that sufficient time is available to power gate a core. TAP EV6 is able to achieve 22.4% (23.9% is Oracle) maximum energy savings, and 3.10% on average. TAP does not achieve the same energy savings as the Oracle because TAP is not able to power gate memory accesses until they miss in the L3 cache, and because lower-bound latencies are used. The result is that TAP avoids any performance hit but misses out on power gating at the beginning of the core stall. TAP also sees a few benchmarks with -0.2% energy savings due to CPU-bound behavior.

FUPG FUPG EV6 has a maximum and average energy savings of 17.6% and 2.2% with a maximum performance hit of 2% (1.5% average), at which point control logic prevents future power gating actions. FUPG does achieve more energy savings than TAP on a few CPU-bound integer codes in which not all the functional units are being used, but the core does not go idle. However, TAP achieves 1.4$\times$ the average energy savings of the FUPG mechanism. Greater energy savings could result from the cooperation of FUPG and TAP.

DVFS-Oracle We also examine DVFS-Oracle using the scaling properties described in Section 8.4. The maximum and average EV6 core energy savings are 24.6% (*lbm*) and 3.3%, respectively. DVFS-Oracle on an EV6 core sees less maximum and average energy savings compared to the power gating oracle, but greater savings when compared to TAP. However, these energy savings suffer from two shortcomings. First, DVFS-Oracle has a maximum and average performance hit of 11.8% (*GemsFDTD*) and 2.4%. Second, these energy savings rely on oracle knowledge.

DVFS-μmean To understand DVFS under a realistic state-of-the-art policy, we consider DVFS-μmean [80], which predicts the performance hit of DVFS at each interval based on performance counters. DVFS-μmean achieves a maximum and average energy savings of 5.9% and 0.6% respectively. Thus, DVFS-μmean achieves less energy savings than TAP while experiencing a 1.0% performance hit on average (3.9% maximum). This result highlights the challenge of applying DVFS at 32nm to match different application behaviors, and why TAP's determinism can result in higher energy savings.

8.4.2 Breakdown of Execution Time and Overheads

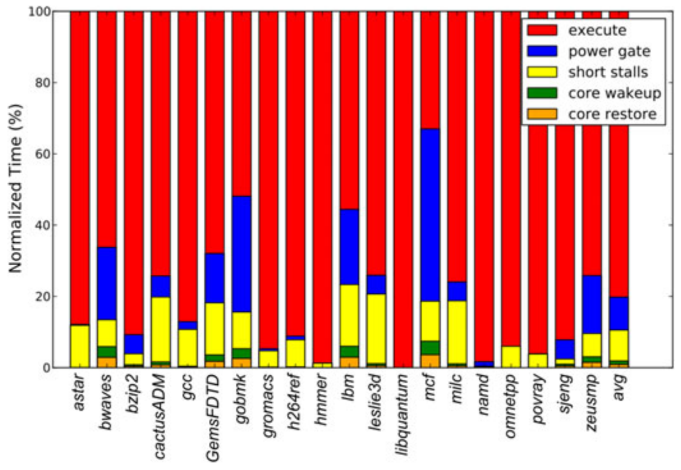


Figure 8.14: Breakdown of simulation time for each benchmark and core combination utilizing TAP to save energy.

Figure 8.14 examines simulation time of each benchmark on an EV6 core and separates it into time spent executing (execute), time spent power gating the core (power gate), short stalls that could not be power gated without energy loss (short stalls), core wake-up time to charge core logic (core wake-up), core restore time to restore data from retention flip-flops and fill the pipeline (core restore), and execution overhead from waking up the core too late (overhead).

We observe that TAP has no measurable performance overhead for EV6 cores. The reason for this is that TAP wakes up the power-gated core for the lower-bound access latency of a next-level hit in the memory hierarchy. The result is that the power-gated core is resumed in advance and always ready for the memory response. TAP power gate cores up to 48.34% of the time for the benchmark *mcf*. Averaged across all benchmarks, TAP power gate cores for 9.24% of time, respectively. In order to power gate, TAP spends an average of 0.97% and 0.93% waking up and restoring its cores from a power gated state.

8.4.3 Energy Savings as a Function of Wake-up Latency

TAP should save more energy as wake-up latency decreases. In this subsection, we examine TAP's sensitivity to wake-up latency, and further consider outcomes if wake-up latencies are reduced below our calculated limits. Figure 8.15 shows that TAP's energy savings increase linearly with reduced wake-up delay. Across the 12 benchmarks, TAP's average energy savings increase from 4.23% to 6.18% as wake-up latency ranges from of 16ns and 2ns. TAP's peak en-

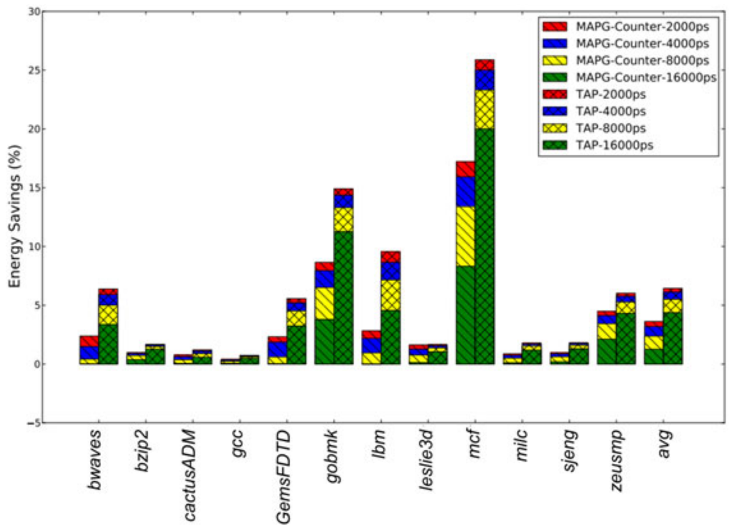


Figure 8.15: Energy savings for TAP as wake-up mode changes from $2ns$ to $16ns$ wake-up latency for an EV6 core. Benchmarks *astar*, *gromacs*, *h264ref*, *hmmer*, *libquantum*, *povray*, *namd*, and *omnetpp* are filtered out due to small changes (less than 0.2%) and space limits.

ergy savings for *mcf* increase from 20.08% to 25.91% as wake-up latency decreases from $16ns$ to $2ns$. In general, improved energy savings results from less wake-up time overhead, and the ability to power gate the core for longer periods of time. Also, we note that changes in wake-up latency have a subdued effect on energy savings. Indeed, as wake-up latency decreases from $16ns$ to $2ns$, TAP's average energy savings increases by 1.46$\times$ on average, indicating that a majority of energy savings already occur at a wake-up latency of $16ns$.

8.4.4 Adapting to Memory Contention

TAP can adapt to varying levels of memory contention and can power gate cores for longer as memory subsystems become oversubscribed. We show how TAP adapts to a system facing increased memory contention for the multi-threaded memory benchmark *stream* running on a CMP with up to 32 cores. *stream* is a memory-intense benchmark used to measure sustained memory bandwidth and computation rates for simple vector kernels [24]. We modify *stream* to act as an embarrassingly parallel memory benchmark such that more threads cause more simultaneous requests to the memory subsystem. Increasing *stream's* thread count causes more queuing of memory requests, longer delay per request and more frequent stalling of each core. A good power gating technique should be able to power gate the core more often to reduce power consumption.

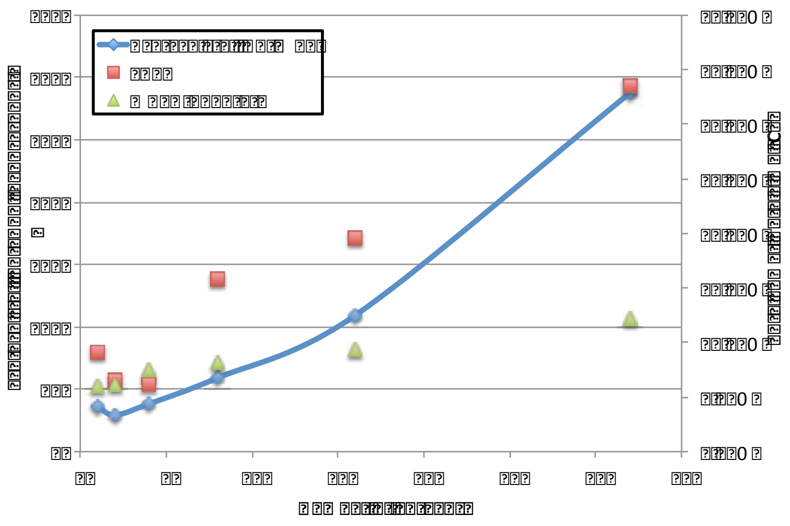


Figure 8.16: TAP adaptation to increasing memory contention. The left y-axis shows the duration of the stall in nanoseconds, while the right y-axis shows the percentage of time that TAP can power gate as a function of the number of *stream* threads.

Figure 8.16 depicts both average duration of core stalls and the percentage of total simulation time TAP power gates the core. The x-axis tracks the number of threads that are run simultaneously. The left y-axis indicates the average stall duration for a core in nanoseconds as the number of threads increases from 1 to 32. The right y-axis shows the percentage of time that TAP can power gate the core as the memory subsystem experiences more contention.

First, we note that as the number of threads increases, the average duration of a core stall increases. For 1, 2, 4, 8, 16, and 32 threads, the average stall durations are 36.77ns, 29.31ns, 38.29ns, 59.191ns, 109.22ns, and 287.63ns, respectively. From 1 to 32 threads, average core-stall duration increases by 7.82$\times$.[26]. The increase in the average core-stall duration is caused by more threads making parallel requests to the memory subsystem at once. This increase causes longer queues in the memory controller and contention to use the limited number of memory channels to transfer the requested cache line. Further, an increase in memory demand decreases the probability of a row-buffer hit and yields longer access latencies.

In addition, Figure 8.16 shows that as cores experience increased memory latency, TAP power gates the core longer. TAP power gates the core for 9.08%, 6.61%, 6.21%, 15.81%, 19.55%, and 33.50% of the time for 1, 2, 3, 4, 8, 16, and 32 threads, respectively. From 1 to 32 threads, TAP power gates cores 3.69$\times$ longer. However, TAP power gates its core less

[26]From 1 to 2 threads, core-stall duration decreases. This happens because more threads increase the amount of available cache (more cores) while increasing the number of simultaneous memory requests.

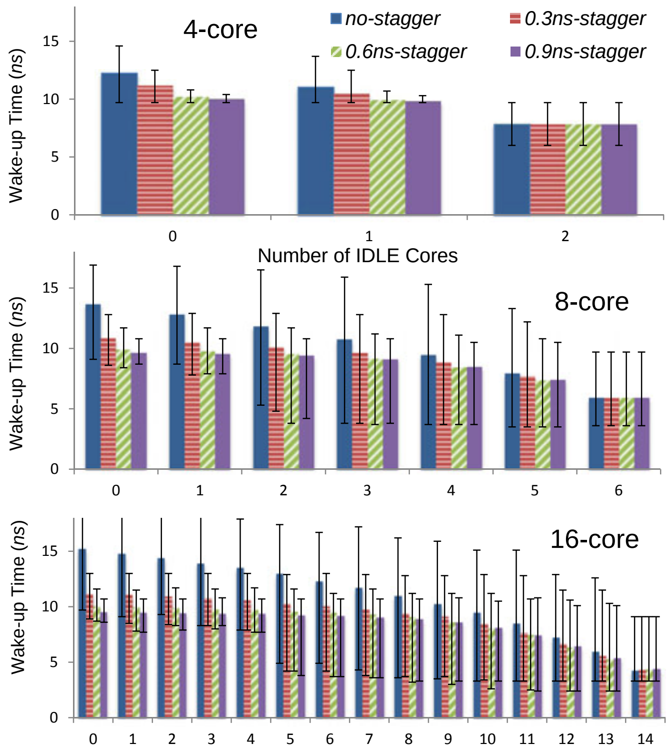


Figure 8.17: Improvement in core wake-up latency with increased stagger for a CMP. The average latency is shown with bars denoting the minimum and maximum safe wake-up modes.

for 4 threads than for 2 even though average core-stall time increases. This is because TAP uses a conservative lower-bound estimate of memory-response time and does not account for all memory scheduling possibilities.

8.4.5 The Staggered Wake-up Effect

For a 16-core system with multiple cores waking up simultaneously, voltage noise on the power distribution network can cause unsafe voltage drop on neighboring active cores. By introducing a sub-nanosecond stagger between two cores waking up with a shared adjacent core, worst-case inrush current and resulting voltage noise are reduced. The result is a faster wake-up mode and increased energy savings on the chip. Hence, cores should wake up when no other core will be waking up for at least a fraction of a nanosecond.

Figure 8.17 shows SPICE simulation of the effect of stagger on core wake-up latency for no-stagger, 0.3ns-stagger, 0.6ns-stagger, and 0.9ns-stagger for CMPs composed of 4, 8, and 16 EV6 cores as 0 to N-1 cores are idle. The first observation is that staggered wake-up can reduce the variance between the minimum and maximum wake-up latency when most cores are actively executing or waking-up. For the 16-core CMP with no cores idle and no stagger, the max and min wake-up latency is 18.4ns and 9.7ns, whereas a staggered wake-up of 0.9ns decreases the max and min values to 10.7ns and 8.6ns respectively. The second observation is that as more cores go idle, staggered wake-up has less impact on reducing the variance of wake-up latency because cores are less likely to interfere with each other, and the core's location becomes the dominant factor in wake-up latency. For example, when 14 cores are idle in a 16-core CMP, the no-stagger and 0.9ns-stagger cases max and min wake-up latencies are both 3.3ns and 9.1ns, respectively.

The third observation is that stagger reduces the maximum wake-up latency as more cores are active. An example of this is for the 16-core case when no core is idle; maximum wake-up latency is 18.4ns without stagger and 10.7ns with 0.9ns stagger, a reduction of 58.2%. Thus, stagger can relax the guardband on maximum wake-up latency.

Finally, we consider the energy impact of staggered wake-up in Figure 8.18 for a CMP made of 16 EV6 cores. The largest improvement in energy savings is 3.14% for *mcf* as energy savings increase from 18.92% to 22.06% for staggered wake-ups of 0.0ns and 0.9ns respectively. On average, energy savings go from 2.42% to 3.00% as stagger increases from 0ns to 0.9ns. Staggered wake-up does not cause any decrease in energy savings. Although cores may have to wake up at slightly different times, the savings in latency with which they wake up is much greater than the stagger offset of 0.9ns.

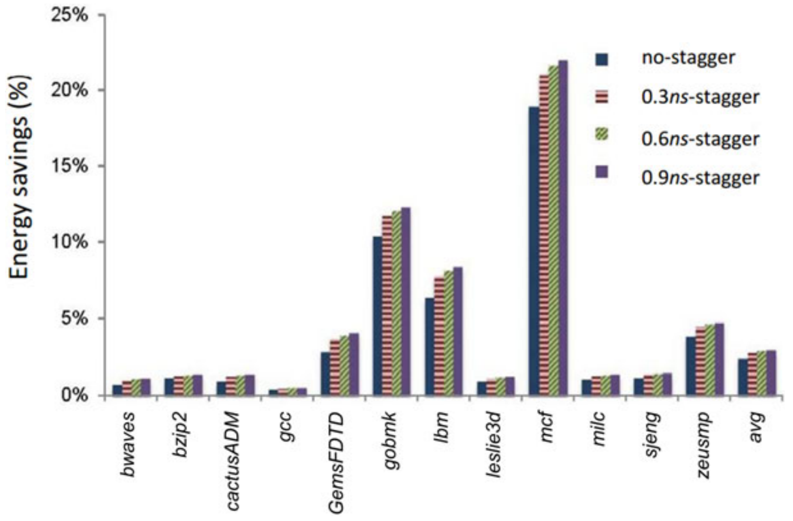


Figure 8.18: Improvement in energy savings with staggered wake-ups in a 16 EV6 core CMP. Benchmarks with less than 0.2% energy savings are omitted.

8.5 Conclusions and Future Directions

With each generation of microprocessors produced, leakage power will become an increasingly dominant issue. In this chapter, we have described TAP, which effectively reduces wasted leakage power for cores waiting on the memory subsystem. TAP achieves 22.4% maximum energy savings for an out-of-order core. The energy savings for the out-of-order core are noteworthy for being within 13.9% of an Oracle scheme. TAP is also shown to be adaptive to different levels of memory contention. Lastly, we demonstrate that a wake-up stagger of $0.9ns$ reduces core wake-up latency by up to 58.2% ($7.7ns$), and increases TAP's energy savings by an additional 3.14%.

Looking toward the future, Figure 8.17 indicates the importance of power-gating short stalls. If core wakeup delay can be significantly reduced and delay periods predicted sooner, the amount of time the core spends power gating can be greatly increased. However, TAP is still conservative about its power-gating intervals, and better prediction on the safe wake-up time is required. To reduce the pessimistic prediction on the wake-up time, we are working on more accurate PDN models.

8.6 Acknowledgments

Chapter 8 is in part a reprint of "MAPG: Memory Access Power Gating", *Proc. Design, Automation and Test in Europe*, 2012; "TAP - Token-Based Adaptive Power Gating", *Proc. International Symposium on Low Power Electronics and Design*, 2012; and "Many-Core Token-Based Adaptive Power Gating", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2013.

I would like to thank my coauthors Dr. Kwangok Jeong, Professor Andrew B. Kahng, Professor Tajana S. Rosing and Dr. Richard Strong. In particular, Dr. Strong made enabling contributions to Section 8.2.

Bibliography

[1] *Blaze MO User's Manual*, http://www.tela-inc.com .

[2] *Cadence Encounter Timing System User's Manual*, http://www.cadence.com .

[3] *Cadence LC User's Manual*, http://www.cadence.com .

[4] *Cadence NC-Sim User's Manual*, http://www.cadence.com .

[5] *Cadence NC-Verilog User's Manual*, http://www.cadence.com .

[6] *Cadence SOCEncounter User's Manual*, http://www.cadence.com .

[7] *Calypto PowerPro CG User's Manual*, http://www.calypto.com .

[8] *DRAMSim2 User's Manual*, http://www.ece.umd.edu/dramsim .

[9] *Enhanced OAGear-Static-Timer with Heuristics for Gate Sizing*, http://nanocad.ee.ucla.edu/Main/DownloadForm .

[10] *ExtremeDA GoldTime User's Manual*, Extreme DA Corp., 2010.

[11] *ILOG CPLEX User's Manual*, http://www.ilog.com/products/cplex .

[12] Intel Corporation, "Intel Atom Processor Z5xx Series", 2008, http://versalogic.com/support/Downloads/PDF/Intel_Atom_Datasheet.pdf .

[13] IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process, *IEEE Std 1497-2001.*

[14] IEEE Standard for Integrated Circuit (IC) Delay and Power Calculation System, *IEEE Std 1481-1999.*

[15] *International Technology Roadmap for Semiconductors 2009 Edition*, http://www.itrs.net/Links/2009ITRS/Home2009.htm .

[16] *International Technology Roadmap for Semiconductors 2010 Edition*, http://www.itrs.net/Links/2010ITRS/Home2010.htm .

[17] *International Technology Roadmap for Semiconductors 2011 Edition (Design Chapter)*, http://www.itrs.net/Links/2011ITRS/Home2011.htm .

[18] *International Technology Roadmap for Semiconductors 2012 Edition*, http://www.itrs.net/Links/2012ITRS/Home2012.htm .

[19] ISPD-2013 Discrete Gate Sizing Contest and Benchmark Suite, http://ispd.cc/contests/13/ispd2013_contest.html .

[20] *IWLS 2005 Benchmarks*, http://iwls.org/iwls2005/benchmarks.html .

[21] *Mathworks MATLAB*, http://www.mathworks.com/products/matlab .

[22] *OpenCores: Open Source IP-Cores*, http://www.opencores.org .

[23] *Standard Performance Evaluation Corporation (SPEC) CPU2006*, http://www.spec.org/cpu2006 .

[24] *STREAM: Sustainable Memory Bandwidth in High Performance Computers*, 2011, http://www.cs.virginia.edu/stream .

[25] *Sun OpenSPARC Project*, http://www.sun.com/processors/opensparc .

[26] *Synopsys Design Compiler User's Manual*, http://www.synopsys.com .

[27] *Synopsys HSPICE User's Manual*, http://www.synopsys.com .

[28] *Synopsys PowerCompiler User's Manual* http://www.synopsys.com .

[29] *Synopsys PrimeTime User's Manual*, http://www.synopsys.com .

[30] *Synopsys Timing Constraints and Optimizaiton User Guide*, http://www.synopsys.com .

[31] *Synopsys Value Change Dump (VCD) Plus Format*, http://www.synopsys.com .

[32] *Tcl/Tk Built-in Socket Commands Manual*, http://www.tcl.tk/man/tcl8.4/TclCmd .

[33] *UC Benchmark Suite for Gate Sizing*, http://vlsicad.ucsd.edu/SIZING .

[34] *UCLA Eyecharts*, http://nanocad.ee.ucla.edu/Main/DownloadForm .

[35] *UCSD Sensitivity-Based Leakage Optimizer*, http://vlsicad.ucsd.edu/SIZING/opimizer.html#SensOpt .

[36] K. Agarwal, H. Deogun, D. Sylvester and K. Nowka, "Power Gating with Multiple Sleep Modes", *Proc. International Symposium on Quality Electronic Design*, 2006, pp. 633–637.

[37] K. Agarwal, D. Sylvester and D. Blaauw, "A Simple Metric for Slew Rate of RC Circuits Based on Two Circuit Moments", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23(9) (2004), pp. 1346–1354.

[38] M. Agarwal, B. C. Paul, M. Zhang, S. Mitra, "Circuit Failure Prediction and Its Application to Transistor Aging", *Proc. VLSI Test Symposium*, 2007, pp. 277–286.

[39] D. Aldous and U. Vazirani, "'Go with the Winners' Algorithms", *Proc. IEEE Symposium on Foundations of Computer Science*, 1994, pp. 492–501.

[40] C. J. Alpert, A. Devgan and C. Kashyap, “A Two Moment RC Delay Metric for Performance Optimization”, *Proc. ACM International Symposium on Physical Design*, 2000, pp. 73–78.

[41] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, “Opportunities and Challenges for Better Than Worst-Case Design”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2005, pp. 2–7.

[42] N. D. P. Avirneni, V. Subramanian and A. K. Somani, “Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Systems”, *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks*, 2009, pp. 185–194.

[43] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel and M. Horowitz, “Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis”, *Proc. International Symposium on Computer Architecture*, 2010, pp. 26–36.

[44] L. Benini, A. Bogliolo and G. De Micheli, “A Survey of Design Techniques for System-Level Dynamic Power Management”, *IEEE Transactions on Very Large Scale Integration Systems* 8(3) (2000), pp. 299–316.

[45] L. Benini and G. De Micheli, “Automatic Synthesis of Low-Power Gated-Clock Finite-State Machines”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15(6) (1996), pp. 630–643.

[46] M. R. C. M. Berkelaar and J. A. G. Jess, “Gate Sizing in MOS Digital Circuits with Linear Programming”, *Proc. European Design Automation Conference*, 1990, pp. 217–221.

[47] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi and S. K. Reinhardt, “The M5 Simulator: Modeling Networked Systems”, *IEEE Micro* 26(4) (2006), pp. 52–60.

[48] K. D. Boese, A. B. Kahng and S. Muddu, “A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations”, *Operations Research Letters* 16(2) (1994), pp. 101–113.

[49] L. Bolzani, A. Calimera, A. Macii, E. Macii and M. Poncino, “Enabling Concurrent Clock and Power Gating in an Industrial Design Flow”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2009, pp. 334–339.

[50] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De and S. Borkar, “Circuit Techniques for Dynamic Variation Tolerance” *Proc. ACM/IEEE Design Automation Conference*, 2009, pp. 4–7.

[51] M. A. Breuer, “Intelligible Test Techniques to Support Error-Tolerance”, *Proc. Asian Test Symposium*, 2004, pp. 386–393.

[52] D. Brooks, V. Tiwari and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations”, *Proc. International Symposium on Computer Architecture*, 2000, pp. 83–94.

[53] B. H. Calhoun and A. P. Chandrakasan, “Standby Power Reduction Using Dynamic Voltage Scaling and Canary Flip-Flop Structures”, *IEEE Journal of Solid-State Circuits* 39(9) (2004), pp. 1504–1511.

[54] B. H. Calhoun and A. P. Chandrakasan, “A 256-kb $65nm$ Sub-threshold SRAM Design for Ultra-Low-Voltage Operation”, *IEEE Journal of Solid-State Circuits* 42(3) (2007), pp. 680–688.

[55] K. Cao, S. Dobre and J. Hu, “Standard Cell Characterization Considering Lithography Induced Variations”, *Proc. ACM/IEEE Design Automation Conference*, 2006, pp. 801–804.

[56] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem and B. Seshasayee, “Ultra-Efficient (Embedded) SOC Architectures Based on Probabilistic CMOS (PCMOS) Technology”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2006, pp. 1110–1115.

[57] C.-C. Chang, J. Cong and M. Xie, “Optimality and Scalability Study of Existing Placement Algorithms”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2003, pp. 621–627.

[58] M. L. Chang and S. Hauck, “Precis: A Design-Time Precision Analysis Tool”, *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 229–238.

[59] M. L. Chang and S. Hauck, “Variable Precision Analysis for FPGA Synthesis”, *Proc. NASA Earth Science Technology Conference*, 2003.

[60] C.-P. Chen, C. C. N. Chu and D. F. Wong, “Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18(7) (1999), pp. 1014–1025.

[61] L. Cheng, P. Gupta, C. Spanos, K. Qian and L. He, “Physically Justifiable Die-Level Modeling of Spatial Variation in View of Systematic Across Wafer Variability”, *Proc. ACM/IEEE Design Automation Conference*, 2009, pp. 104–109.

[62] D. G. Chinnery and K. Keutzer, “Linear Programming for Sizing, V_{th} and V_{dd} Assignment”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2005, pp. 149–154.

[63] V. Chippa, A. Raghunathan, K. Roy and S. Chakradhar, “Dynamic Effort Scaling: Managing the Quality-Efficiency Tradeoff”, *Proc. ACM/IEEE Design Automation Conference*, 2011, pp. 603–608.

[64] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy and S. T. Chakradhar, “Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency”, *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 555–560.

[65] M. Cho, N. Sathe, M. Gupta, S. Kumar, S. Yalamanchilli and S. Mukhopadhyay, “Proactive Power Migration to Reduce Maximum Value and Spatiotemporal Non-uniformity

of On-Chip Temperature Distribution in Homogeneous Many-Core Processors", *Proc. Semiconductor Thermal Measurement and Management Symposium*, 2010, pp. 180–186.

[66] I. S. Chong, H.-Y. Cheong and A. Ortega, "New Quality Metric for Multimedia Compression Using Faulty Hardware", *Proc. International Workshop on Video Processing and Quality Metrics for Consumer Electronics*, 2006, pp. 267–272.

[67] H. Chou, Y.-H. Wang and C. C.-P. Chen, "Fast and Effective Gate Sizing with Multiple-V_t Assignment Using Generalized Lagrangian Relaxation", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2005, pp. 381–386.

[68] N. Choudhary, S. Wadhavkar, T. Shah, S. Navada, H. Hashemi and E. Rotenberg, "FabScalar", *Proc. Workshop on Architectural Research Prototyping*, 2009.

[69] M. Choudhury, V. Chandra, K. Mohanram and R. Aitken, "TIMBER: Time Borrowing and Error Relaying for Online Timing Error Resilience", Proc. IEEE/ACM Design, Automation and Test in Europe, 2010, pp. 1554–1559.

[70] M. H. Chowdhury, J. Gjanci and P. Khaled, "Innovative Power Gating for Leakage Reduction", *Proc. IEEE International Symposium on Circuits and Systems*, 2008, pp. 1568–1571.

[71] W. Chuang, S. S. Sapatnekar and I. N. Hajj, "Timing and Area Optimization for Standard-Cell VLSI Circuit Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14(3) (1995), pp. 308–320.

[72] E. Chung and J. Smolens, "*OpenSPARC T1: Architectural Transplants*", http://transplant.sunsource.net .

[73] J. Cong and K. Minkovich, "Logic Synthesis for Better Than Worst-Case Designs", *Proc. International Symposium on VLSI Design, Automation and Test*, 2009, pp. 166–169.

[74] J. Cong, M. Romesis and M. Xie, "Optimality and Stability Study of Timing-Driven Placement Algorithms", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2003, pp. 472–478.

[75] O. Coudert, "Gate Sizing for Constrained Delay/Power/Area Optimization", *IEEE Transactions on Very Large Scale Integration Systems* 5(4) (1997), pp. 465–472.

[76] J. Darnauer and W. W.-M. Dai, "A Method for Generating Random Circuits and Its Application to Routability Measurement", *Proc. ACM International Symposium on Field Programmable Gate Arrays*, 1996, pp. 66–72.

[77] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull and D. T. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance", *IEEE Journal of Solid-State Circuits* 44(1) (2009), pp. 32–48.

[78] A. Dharchoudhury, D. Blaauw, J. Norton, S. Pullela and J. Dunning, "Transistor-Level Sizing and Timing Verification of Domino Circuits in the Power PC^{TM}Microprocessor", *Proc. IEEE International Conference on Computer Design*, 1997, pp. 143–148.

[79] G. Dhiman, K. K. Pusukuri and T. Rosing, “Analysis of Dynamic Voltage Scaling for System Level Energy Management”, *Proc. USENIX Workshop on Power Aware Computing and System*, 2008, pp. 9–14.

[80] G. Dhiman and T. S. Rosing, “Dynamic Voltage Frequency Scaling For Multi-Tasking Systems Using Online Learning”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2007, pp. 207–212.

[81] S. Dobre, Qualcomm CDMA Technologies, Inc., *personal communication*, March-Sept. 2011.

[82] W. C. Elmore, “The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers”, *Journal of Applied Physics* 19(1) (1948), pp. 55–63.

[83] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge, “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation”, *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 7–18.

[84] C. F. Fang, R. A. Rutenbar, M. Püchel and T. Chen, “Toward Efficient Static Analysis of Finite-Precision Effects in DSP Applications via Affine Arithmetic Modeling”, *Proc. ACM/IEEE Design Automation Conference*, 2003, pp. 496–501.

[85] J. P. Fishburn, “Clock Skew Optimization”, *IEEE Transactions on Computers* 39(7) (1990), pp. 945–951.

[86] J. P. Fishburn and A. E. Dunlop, “TILOS: A Posynomial Programming Approach to Transistor Sizing”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1985, pp. 326–328.

[87] G. Flach, T. Reimann, G. Posser, M. Johann and R. Reis, “Simultaneous Gate Sizing and V_{th} Assignment using Lagrangian Relaxation and Delay Sensitivities”, *IEEEComputer Society Annual Symposium on VLSI*, 2013, to apear.

[88] K. Fujita, Y. Torii, M. Hori, J. Oh, L. Shifren, P. Ranade, M. Nakagawa, K. Okabe, T. Miyake, K. Ohkoshi, M. Kuramae, T. Mori, T. Tsuruta, S. Thompson and T. Ema, “Advanced Channel Engineering Achieving Aggressive Reduction of V_T Variation for Ultra-Low-Power Applications”, *Proc. IEEE International Electron Devices Meeting*, 2011, pp. 32.3.1–32.3.4.

[89] K. Fukuoka, M. Iijima, K. Hamada, M. Numa and A. Tada, “Leakage Power Reduction for Clock Gating Scheme on PD-SOI”, *Proc. IEEE International Symposium on Circuits and Systems*, 2004, pp. 613–616.

[90] J. George, B. Marr, B. E. S. Akgul and K. V. Palem, “Probabilistic Arithmetic and Energy Efficient Embedded Signal Processing”, *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006, pp. 158–168.

[91] S. Ghosh, S. Bhunia and K. Roy, “CRISTA: A New Paradigm for Low-Power and Robust Circuit Synthesis Under Parameter Variations Using Critical Path Isolation”,

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(11) (2007), pp. 1947–1956.

[92] S. Ghosh and K. Roy, “Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era”, *Proceedings of the IEEE* 98(10) (2010), pp. 1718–1751.

[93] F. Glover, “Tabu Search – Part I”, *ORSA Journal on Computing* 1(3) (1989), pp. 190–206.

[94] T. F. Gonzalez (editor), *Handbook of Approximation Algorithms and Metaheuristics*, Chapman and Hall/CRC, 2007.

[95] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson and M. B. Taylor, “The GreenDroid Mobile Application Processor: An Architecture for Silicon’s Dark Future”, *IEEE Micro* 31(2) (2011), pp. 86–95.

[96] P. Grassberger, “Go with the Winners: A General Monte Carlo Strategy”, http://arxiv.org/pdf/cond-mat/0201313v1.pdf .

[97] B. Greskamp and J. Torrellas, “Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking”, *Proc. International Conference on Parallel Architectures and Compilation Techniques*, 2007, pp. 213–224.

[98] B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen and C. Zilles, “BlueShift: Designing Processors for Timing Speculation from the Ground Up”, *Proc. International Symposium on High-Performance Computer Architecture*, 2009, pp. 213–224.

[99] P. Gupta, A. B. Kahng, A. Kasibhatla and P. Sharma, “Eyecharts: Constructive Benchmarking of Gate Sizing Heuristics”, *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 597–602.

[100] P. Gupta, A. B. Kahng and P. Sharma, “A Practical Transistor-Level Dual Threshold Voltage Assignment Methodology”, *Proc. International Symposium on Quality Electronic Design*, 2005, pp. 421–426.

[101] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, “Selective Gate-Length Biasing for Cost-Effective Runtime Leakage Control”, *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 327–330.

[102] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, “Gate-Length Biasing for Runtime-Leakage Control”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25(8) (2006), pp. 1475–1485.

[103] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan and K. Roy, “IMPACT: Imprecise Adders for Low-Power Approximate Computing”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414.

[104] L. W. Hagen, D. J. -H. Huang and A. B. Kahng, “Quantified Suboptimality of VLSI Layout Heuristics”, *Proc. ACM/IEEE Design Automation Conference*, 1995, pp. 216–221.

[105] S. Hanson, B. Zhai, K. Bernstein, D. Blaauw, A. Bryant, L. Chang, K. Das, W. Haensch, E. Nowak and D. Sylvester, “Ultralow-Voltage Minimum-Energy CMOS”, *IBM Journal of Research and Development* 4.5(50) (2006), pp. 469–490.

[106] B. Hargreaves, H. Hult and S. Reda, “Within-die Process Variations: How Accurately Can They Be Statistically Modeled?”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2008, pp. 524–530.

[107] K. He, R. Luo and Y. Wang, “A Power Gating Scheme for Ground Bounce Reduction during Mode Transition”, *Proc. IEEE International Conference on Computer Design*, 2007, pp. 388–394.

[108] R. Hegde and N. R. Shanbhag, “Energy-Efficient Signal Processing via Algorithmic Noise-Tolerance”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 1999, pp. 30–35.

[109] S. Held, “Gate Sizing for Large Cell-Based Designs”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2009, pp. 827–832.

[110] M. Horiguchi, T. Sakata and K. Itoh, “Switched-Source-Impedance CMOS Circuit for Low Standby Subthreshold Current Giga-Scale LSI’s”, *IEEE Journal of Solid-State Circuits* 28(11) (1993), pp. 1131–1135.

[111] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim and I. L. Markov, “Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 233–239.

[112] S. Hu, M. Ketkar and J. Hu, “Gate Sizing for Cell-Library-Based Designs”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28(6) (2009), pp. 818–825.

[113] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson and P. Bose, “Microarchitectural Techniques for Power Gating of Execution Units”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2004, pp. 32–37.

[114] G. Huang, D. Sekar, A. Naeemi, K. Shakeri and J. D. Meindl, “Physical Model for Power Supply Noise and Chip/Package Co-Design in Gigascale Systems with the Consideration of Hot Spots”, *Proc. IEEE Custom Integrated Circuits Conference*, 2007, pp. 841–844.

[115] J. Huang and J. Lach, “Exploring the Fidelity-Efficiency Design Space Using Imprecise Arithmetic”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2011, pp. 579–584.

[116] J. Huang, J. Lach and G. Robins, “Analytic Error Modeling for Imprecise Arithmetic Circuits”, *Proc. Silicon Errors in Logic - System Effects*, 2011.

[117] J. Huang, J. Lach and G. Robins, "A Methodology for Energy-Quality Tradeoffs Using Imprecise Hardware", *Proc. ACM/IEEE Design Automation Conference*, 2012, pp. 504–509.

[118] M. D. Hutton, J. Rose, J. P. Grossman and D. G. Corneil, "Characterization and Parameterized Generation of Synthetic Combinational Benchmark Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17(10) (1998), pp. 985–996.

[119] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 347–358.

[120] C. Isci, A. Buyuktosunoglu and M. Martonosi, "Long-Term Workload Phases: Duration Predictions and Applications to DVFS", *IEEE Micro* 25(5) (2005), pp. 39–51.

[121] K. Jeong and A. B. Kahng, "Methodology From Chaos in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 885–892.

[122] K. Jeong, A. B. Kahng, S. Kang, T. S. Rosing and R. Strong, "MAPG: Memory Access Power Gating", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2012, pp. 1054–1059.

[123] K. Jeong, A. B. Kahng and H. Yao, "Rent Parameter Evaluation Using Different Methods", http://vlsicad.ucsd.edu/WLD/RentCon.pdf .

[124] K. Jeong, A. B. Kahng and H. Yao, "Revisiting the Linear Programming Framework for Leakage Power vs. Performance Optimization", *Proc. International Symposium on Quality Electronic Design*, 2009, pp. 127–134.

[125] A. B. Kahng and S. Kang, "Accuracy-Configurable Adder for Approximate Arithmetic Designs", *Proc. ACM/IEEE Design Automation Conference*, 2012, pp. 820–825.

[126] A. B. Kahng and S. Kang, "Construction of Realistic Gate Sizing Benchmarks with Known Optimal Solutions", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 153–160.

[127] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs", *Proc. International Symposium on High-Performance Computer Architecture*, 2010, pp. 119–129.

[128] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Recovery-Driven Design: A Methodology for Power Minimization for Error Tolerant Processor Modules", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 825–830.

[129] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 825–831.

[130] A. B. Kahng, S. Kang, H. Lee, S. Nath and J. Wadhwani, "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. IEEE System-Level Interconnect Prediction*, 2013.

[131] A. B. Kahng, S. Kang, T. S. Rosing and R. Strong, "TAP - Token-Based Adaptive Power Gating", *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2012, pp. 203–208.

[132] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. International Symposium on Quality Electronic Design*, 2002, pp. 206–211.

[133] A. B. Kahng and S. Muddu, "An Analytical Delay Model for RLC Interconnects", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(12) (1997), pp. 1507–1514.

[134] K. Kasamsetty, M. Ketkar and S. S. Sapatnekar, "A New Class of Convex Functions for Delay Modeling and Its Application to the Transistor Sizing Problem", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(7) (2000), pp. 779–788.

[135] C. V. Kashyap, C. J. Alpert, F. Liu and A. Devgan, "PERI: A Technique for Extending Delay and Slew Metrics to Ramp Inputs", *Proc. TAU*, 2002, pp. 57–62.

[136] M. Keating, D. Flynn, R. Aitken, A. Gibbons and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design*, Springer, 2007.

[137] Z. Kedem, V. J. Mooney, K. K. Muntimadugu, K. V. Palem, A. Devarasetty and P. D. Parasuramuni, "Optimizing Energy to Minimize Errors in Dataflow Graphs Using Approximate Adders", *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2010, pp. 177–186.

[138] D. R. Kelly, B. J. Phillips and S. Al-Sarawi, "Approximate Signed Binary Integer Multipliers for Arithmetic Data Value Speculation", *Proc. Conference on Design and Architectures for Signal and Image Processing*, 2009, pp. 97–104.

[139] V. Khandelwal and A. Srivastava, "Active Mode Leakage Reduction Using Fine-Grained Forward Body Biasing Strategy", *Integration, the VLSI Journal* 40(4) (2007), pp. 561–570.

[140] N. S. Kim, J. Seomun, A. Sinkar, J. Lee, T. H. Han, K. Choi and Y. Shin, "Frequency and Yield Optimization Using Power Gates in Power-Constrained Designs", *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2008, pp. 121–126.

[141] S. Kim, S. V. Kosonocky and D. R. Knebel, "Understanding and Minimizing Ground Bounce during Mode Transition of Power Gating Structures", *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2003, pp. 22–25.

[142] S. Kim, S. V. Kosonocky, D. R. Knebel, K. Stawiasz and M. C. Papaefthymiou, "A Multi-Mode Power Gating Structure for Low-Voltage Deep-Submicron CMOS ICs", *IEEE Transactions on Circuits and Systems II* 54(7) (2007), pp. 586–590.

[143] S. Kim, I. Kwon, D. Fick, M. Kim, Y.-P. Chen and D. Sylvester, “Razor-Lite: A Side-Channel Error-Detection Register for Timing-Margin Recovery in 45nm SOI CMOS”, *Proc. IEEE International Solid-State Circuits Conference*, 2013, pp. 264–265.

[144] A. B. Kinsman and N. Nicolici, “Finite Precision Bit-Width Allocation Using SAT-Modulo Theory”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2009, pp. 1106–1111.

[145] T. Kitahara, F. Minami, T. Ueda, K. Usami, S. Nishio, M. Murakata and T. Mitsuhashi, “A Clock-Gating Method for Low-Power LSI Design”, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 1998, pp. 307–312.

[146] D. Koes, T. Chelcea, C. Onyeama and S. C. Goldstein, “Adding Faster with Application Specific Early Termination”, *Proc. CMU Research Showcase*, 2005.

[147] P. Kulkarni, P. Gupta and M. Ercegovac, “Trading Accuracy for Power with an Underdesigned Multiplier Architecture”, *Proc. IEEE/ACM International Conference on VLSI Design*, 2011, pp. 346–351.

[148] K. Kum and W. Sung, “Combined Word-Length Optimization and High-Level Synthesis of Digital Signal Processing Systems”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20(8) (2001), pp. 921–930.

[149] R. Kumar, “Stochastic Processors”, *NSF Workshop on Science of Power Management*, March 2009.

[150] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan and D. M. Tullsen, “Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction”, *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 81–92.

[151] R. Kumar and G. Hinton, “A Family of 45nm IA Processors”, *Proc. IEEE International Solid-State Circuits Conference*, 2009, pp. 58–59.

[152] T. Kuroda, “CMOS Design Challenges to Power Wall”, *Proc. International Microprocesses and Nanotechnology Conference*, 2001, pp. 6–7.

[153] B. S. Landman and R. L. Russo, “On a Pin versus Block Relationship for Partitions of Logic Graphs”, *IEEE Transactions on Computers* C-20(12) (1971), pp. 1469–1479.

[154] M. S. K. Lau, K.-V. Ling and Y.-C. Chu, “Energy-Aware Probabilistic Multiplier: Design and Analysis”, *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2009, pp. 281–290.

[155] D.-U. Lee, A. A. Gaffar, O. Mencer and W. Luk, “MiniBit: Bit-Width Optimization via Affine Arithmetic”, *Proc. ACM/IEEE Design Automation Conference*, 2005, pp. 837–840.

[156] J. Lee and P. Gupta, “Incremental Gate Sizing for Late Process Changes”, *Proc. IEEE International Conference on Computer Design*, 2010, pp. 215–221.

[157] J. Lee and P. Gupta, "Discrete Circuit Optimization: Library Based Gate Sizing and Threshold Voltage Assignment", *Foundations and Trends in Electronic Design Automation* 6(1) (2012), pp. 1–120.

[158] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan and C. Kozyrakis, "Power Management of Datacenter Workloads Using Per-Core Power Gating", *IEEE Computer Architecture Letters* 8(2) (2009), pp. 48–51.

[159] H. Li, C.-Y. Cher, T. N. Vijaykumar and K. Roy, "VSV: L2-Miss-Driven Variable Supply-Voltage Scaling for Low Power", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 19–28.

[160] L. Li, K. Choi and H. Nan, "Effective Algorithm for Integrating Clock Gating and Power Gating to Reduce Dynamic and Active Leakage Power Simultaneously", *Proc. International Symposium on Quality Electronic Design*, 2011, pp.74–79.

[161] L. Li, P. Kang, Y. Lu and H. Zhou, "An Efficient Algorithm for Library-Based Cell-Type Selection in High-Performance Low-Power Designs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 226–232.

[162] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[163] Y.-J. Lin, C.-L. Yang, J.-W. Huang and N. Chang, "Memory Access Aware Power Gating for MPSoCs", *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2012, pp. 121–126.

[164] Y. Liu and J. Hu, "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29(2) (2010), pp. 223–234.

[165] V. S. Livramento, C. Guth, J. L. Güntzel and M. O. Johann, "Fast and Efficient Lagrangian Relaxation-Based Discrete Gate Sizing", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2013, pp. 1855–1860.

[166] S.-L. Lu, "Speeding Up Processing with Approximation Circuits", *IEEE Computer* 37(3) (2004), pp. 67–73.

[167] A. Lungu, P. Bose, A. Buyuktosunoglu and D. J. Sorin, "Dynamic Power Gating with Quality Guarantees", *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2009, pp. 377–382.

[168] N. D. MacDonald, "Timing Closure in Deep Submicron Designs", *DAC Knowledge Center Article*, 2010.

[169] E. Macii, L. Bolzani, A. Calimera, A. Macii and M. Poncino, "Integrating Clock Gating and Power Gating for Combined Dynamic and Leakage Power Optimization in Digital CMOS Circuits", *Proc. Euromicro Conference on Digital System Design Architecturs, Methods and Tools*, 2008, pp. 298–303.

[170] N. Madan, A. Buyuktosunoglu, P. Bose and M. Annavaram, “A Case for Guarded Power Gating for Multi-Core Processors”, *Proc. International Symposium on High-Performance Computer Architecture*, 2011, pp. 291–300.

[171] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt and B. Werner, “Simics: A Full System Simulation Platform”, *IEEE Computer* 35(2) (2002), pp. 50–58.

[172] O. Martin, S. W. Otto and E. W. Felten, “Large-Step Markov Chains for the Traveling Salesman Problem”, *Complex Systems* 5(3) (1991), pp. 299–326.

[173] S. P. McCormick, “Modeling and Simulation of VLSI Interconnections with Moments”, *PhD Thesis*, MIT, June 1989.

[174] J. Miao, K. He, A. Gerstlauer and M. Orshansky, “Modeling and Synthesis of Quality-Energy Optimal Approximate Adders”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 728–735.

[175] J. N. Mistry, B. M. Al-Hashimi, D. Flynn and S. Hill, “Sub-Clock Power-Gating Technique for Minimising Leakage Power During Active Mode”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2011, pp. 1–6.

[176] H. D. Mohammed and L. Hemmert, “Fast Pipelined Adder/Subtractor Using Increment/Decrement Function with Reduced Register Utilization”, *U.S. Patent* No. 7,007,059, 2006.

[177] C. W. Moon, P. Gupta, P. J. Donehue and A. B. Kahng, “Method of Designing a Digital Circuit by Correlating Different Static Timing Analyzers”, *U.S. Patent* No. 7,823,098, 2010.

[178] S. Narayanan, J. Sartori, R. Kumar and D. L. Jones, “Scalable Stochastic Processors”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2010, pp. 335–338.

[179] A. Nayak, M. Haldar, A. Choudhary and P. Banerjee, “Precision and Error Analysis of MATLAB Applications during Automated Hardware Synthesis for FPGAs”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2001, pp. 722–728.

[180] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson and K. Keutzer, “Minimization of Dynamic and Static Power through Joint Assignment of Threshold Voltages and Sizing Optimization”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2003, pp. 158–163.

[181] W. Ning, “Strongly NP-Hard Discrete Gate-Sizing Problems”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(8) (1994), pp. 1045–1051.

[182] A. Odabasioglu, M. Celik and L. T. Pileggi, “PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1997, pp. 58–65.

[183] T. Okuma, T. Ishihara and H. Yasuura, “Real-time Task Scheduling for a Variable Voltage Processor”, *Proc. International Symposium on System Synthesis*, 1999, pp. 24–29.

[184] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, “The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite”, *Proc. ACM International Symposium on Physical Design*, 2012, pp. 161–164. http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html .

[185] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, “An Improved Benchmark Suite for the ISPD-2012 Discrete Cell Sizing Contest”, *Proc. ACM International Symposium on Physical Design*, 2013, pp. 168–170, http://archive.sigda.org/ispd/contests/13/ispd2013_contest.html.

[186] M. M. Ozdal, S. Burns and J. Hu, “Gate Sizing and Device Technology Selection Algorithms for High-Performance Industrial Designs”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 724–731.

[187] S. Palacharla, N. P. Jouppi and J. E. Smith, “Complexity-Effective Superscalar Processors”, *Proc. International Symposium on Computer Architecture*, 1997, pp. 206–218.

[188] P. Pant, R. K. Roy and A. Chatterjee, “Dual-Threshold Voltage Assignment with Transistor Sizing for Low Power CMOS Circuits”, *IEEE Transactions on Very Large Scale Integration Systems* 9(2) (2001), pp. 390–394.

[189] J. Patel, “CMOS Process Variations: A Critical Operation Point Hypothesis”, *Online Presentation*, 2008.

[190] P. I. Pénzes and A. J. Martin, “Energy-Delay Efficiency of VLSI Computations”, *Proc. Great Lakes Symposium on VLSI*, 2002, pp. 104–111.

[191] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood and B. Calder, “Using SimPoint for Accurate and Efficient Simulation”, *Proc. International Conference on Measurement and Modeling of Computer Systems*, 2003, pp. 318–319.

[192] B. J. Phillips, D. R. Kelly and B. W. Ng, “Estimating Adders for a Low Density Parity Check Decoder”, *Proc. International Society for Optical Engineering*, 2006, pp. 1–9.

[193] D. Pisinger, “A Minimal Algorithm for the Multiple-Choice Knapsack Problem”, *European Journal of Operational Research* 83(2) (1995), pp. 394–410.

[194] M. Qazi, M. E. Sinangil and A. P. Chandrakasan, “Challenges and Directions for Low-Voltage SRAM”, *IEEE Transactions on Design and Test of Computers* 28(1) (2011), pp. 32–43.

[195] H. Qin, Y. Cao, D. Markovic, A. Vladimirescue and J. Rabaey, “SRAM Leakage Suppression by Minimizing Standby Supply Voltage”, *Proc. International Symposium on Quality Electronic Design*, 2004, pp. 55–60.

[196] M. Rahman, H. Tennakoon and C. Sechen, “Power Reduction via Near-Optimal Library-Based Cell-Size Selection”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2011, pp. 1–4.

[197] M. Rahman, H. Tennakoon and C. Sechen, "Library-Based Cell-Size Selection Using Extended Logical Effort", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32(7) (2013), pp. 1086–1099.

[198] C. L. Ratzlaff, N. Gopal and L. T. Pillage, "RICE: Rapid Interconnect Circuit Evaluator", *Proc. ACM/IEEE Design Automation Conference*, 1991, pp. 555–560.

[199] H. Ren and S. Dutt, "A Network-Flow Based Cell Sizing Algorithm", *Proc. International Workshop on Logic Synthesis*, 2008, pp. 7–14.

[200] A. Rogers, D. Kaplan, E. Quinnell and B. Kwan, "The Core-C6 (CC6) Sleep State of the AMD Bobcat x86 Microprocessor", *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2012, pp. 367–372.

[201] S. Roy, W. Chen, C. C.-P. Chen and Y. H. Hu, "Numerically Convex Forms and Their Application in Gate Sizing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26(9) (2007), pp. 1637–1647.

[202] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya and S.-M. Kang, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12(11) (1993), pp. 1621–1634.

[203] J. Sartori and R. Kumar, "Overscaling-Friendly Timing Speculation Architectures", *Proc. Symposium on Great Lakes Symposium on VLSI*, 2010, pp. 209–214.

[204] J. Sartori, J. Sloan and R. Kumar, "Fluid NMR - Performing Power/Reliability Tradeoffs for Applications with Error Tolerance", *Workshop on Power Aware Computing and Systems*, 2009.

[205] T. Sato and Y. Kunitake, "A Simple Flip-Flop Circuit for Typical-Case Designs for DFM", *Proc. International Symposium on Quality Electronic Design*, 2007, pp. 539–544.

[206] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas and M. L. Scott, "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling", *Proc. International Symposium on High-Performance Computer Architecture*, 2002, pp. 29–40.

[207] J. Seomun, I. Shin and Y. Shin, "Synthesis of Active-Mode Power-Gating Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(3) (2012), pp. 391–403.

[208] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw and V. Zolotov, "Discrete V_t Assignment and Gate Sizing Using a Self-Snapping Continuous Formulation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 704–711.

[209] N. R. Shanbhag, R. A. Abdallah, R. Kumar and D. L. Jones, "Stochastic Computation", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 859–864

[210] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically Characterizing Large Scale Program Behavior", *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002, pp. 45–57.

[211] D. Shin and S. K. Gupta, "A Re-Design Technique for Datapath Modules in Error Tolerant Applications", *Proc. IEEE Asian Test Symposium*, 2008, pp. 431–437.

[212] D. Shin and S. K. Gupta, "Approximate Logic Synthesis for Error Tolerant Applications", *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2010, pp. 957–960.

[213] Y. Shin, J. Seomun, K.-M. Choi and T. Sakurai, "Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-Cell VLSI Designs", *ACM Transactions on Design Automation of Electronic Systems* 15(4) (2010), pp. 1–37.

[214] A. Shye, B. Ozisikyilmaz, A. Mallik, G. Memik, P. A. Dinda, R. P. Dick and A. N. Choudhary, "Learning and Leveraging the Relationship between Architecture-Level Measurements and Individual User Satisfaction", *Proc. International Symposium on Computer Architecture*, 2008, pp. 427–438.

[215] H. Singh, K. Agarwal, D. Sylvester and K. J. Nowka, "Enhanced Leakage Reduction Techniques Using Intermediate Strength Power Gating", *IEEE Transactions on Very Large Scale Integration Systems* 15(11) (2007), pp.1215–1224.

[216] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda and D. Blaauw, "Duet: An Accurate Leakage Estimation and Optimization Tool for Dual-V_t Circuits", *IEEE Transactions on Very Large Scale Integration Systems* 10(2) (2002), pp. 79–90.

[217] S. Sirichotiyakul, T. Edwards, C. Oh, J. Zuo, A. Dharchoudhury, R. Panda and D. Blaauw, "Stand-By Power Minimization through Simultaneous Threshold Voltage Selection and Circuit Sizing", *Proc. ACM/IEEE Design Automation Conference*, 1999, pp. 436–441.

[218] A. Srivastava, D. Sylvester and D. Blaauw, "Power Minimization Using Simultaneous Gate Sizing, Dual-V_{dd} and Dual-V_{th} Assignment", *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 783–787.

[219] M. Stephenson, J. Babb and S. Amarasinghe, "Bitwidth Analysis with Application to Silicon Compilation", *Proc. ACM Conference on Programming Language Design and Implementation*, 2000, pp. 108–120.

[220] D. Stroobandt, P. Verplaetse and J. V. Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(9) (2000), pp. 1011–1022.

[221] V. Subramanian and A. K. Somani, "Conjoined Pipeline: Enhancing Hardware Reliability and Performance through Organized Pipeline Redundancy", *Proc. IEEE Pacific Rim International Symposium on Dependable Computing*, 2008, pp. 9–16.

[222] A. K. Sultania, D. Sylvester and S. S. Sapatnekar, "Tradeoffs between Gate Oxide Leakage and Delay for Dual T_{ox} Circuits", *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 761–766.

[223] V. Sundararajan, S. S. Sapatnekar and K. K. Parhi, “Fast and Exact Transistor Sizing Based on Iterative Relaxation”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21(5) (2002), pp. 568–581.

[224] H. Tennakoon and C. Sechen, “Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based Pre-Processing Step”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2002, pp. 395–402.

[225] H. Tennakoon and C. Sechen, “Efficient and Accurate Gate Sizing with Piecewise Convex Delay Models”, *Proc. ACM/IEEE Design Automation Conference*, 2005, pp. 807–812.

[226] J. Tschanz, K. Bowman, S.-L. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik and V. De, “A $45nm$ Resilient and Adaptive Microprocessor Core for Dynamic Variation Tolerance”, *Proc. IEEE International Solid-State Circuits Conference*, 2010, pp. 282–283.

[227] J. W. Tschanz, S. G. Narendra, Y. Ye, B. A. Bloechel, S. Borkar and V. De, “Dynamic Sleep Transistor and Body Bias for Active Leakage Power Control of Microprocessors”, *IEEE Journal of Solid-State Circuits* 38(11) (2003), pp. 1838–1845.

[228] D. M. Tullsen, “Simulation and Modeling of a Simultaneous Multithreading Processor”, *Proc. Annual Computer Measurement Group Conference*, 1996, pp. 819–828.

[229] K. Usami and N. Ohkubo, “A Design Approach for Fine-Grained Run-Time Power Gating Using Locally Extracted Sleep Signals”, *Proc. IEEE International Conference on Computer Design*, 2006, pp. 155–161.

[230] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy and A. Raghunathan, “SALSA: Systematic Logic Synthesis of Approximate Circuits”, *Proc. ACM/IEEE Design Automation Conference*, 2012, pp. 796–801.

[231] R. Venkatesan, A. Agarwal, K. Roy and A. Raghunathan, “MACACO: Modeling and Analysis of Circuits for Approximate Computing”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 667–673.

[232] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson and M. B. Taylor, “Conservation Cores: Reducing the Energy of Mature Computations”, *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010, pp. 205–218.

[233] A. K. Verma, P. Brisk and P. Ienne, “Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design”, *Proc. IEEE/ACM Design, Automation and Test in Europe*, 2008, pp. 1250–1255.

[234] P. Viola and M. J. Jones, “Robust Real-Time Face Detection”, *International Journal of Computer Vision* 52(2) (2004), pp. 137–154.

[235] L. Wan and D. Chen, “DynaTune: Circuit-Level Optimization for Timing Speculation Considering Dynamic Path Behavior”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 172–179.

[236] J. Wang, D. Das and H. Zhou, "Gate Sizing by Lagrangian Relaxation Revisited", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28(7) (2009), pp. 1071–1084.

[237] Z. Wang and A. C. Bovik, "Mean Squared Error: Love it or Leave it? A New Look at Signal Fidelity Measures", *IEEE Signal Processing Magazine* 26(1) (2009), pp. 98–117.

[238] O. Wechsler, "Insisde Intel®Core$^{\text{TM}}$Microarchitecture: Setting New Standards for Energy-Efficient Performance", *Technology@Intel Magazine*, 2006.

[239] L. Wei, Z. Chen, K. Roy, M. C. Johnson, Y. Ye and V. K. De, "Design and Optimization of Dual Threshold Circuits for Low-Voltage Low-Power Applications", *IEEE Transactions on Very Large Scale Integration Systems* 7(1) (1999), pp. 16–24.

[240] L. Wei, K. Roy and C.-K. Koh, "Power Minimization by Simultaneous Dual-V_{th} Assignment and Gate-Sizing", *Proc. IEEE Custom Integrated Circuits Conference*, 2000, pp. 413–416.

[241] T. Yamada, S. Kataoka and K. Watanabe, "Heuristic and Exact Algorithms for the Disjunctively Constrained Knapsack Problem", *Information Processing Society of Japan Journal* 43(9) (2002), pp. 2864–2870.

[242] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor", *IEEE Micro* 16(2) (1996), pp. 28–40.

[243] B. Zahn, "A Utility for Leakage Power Recovery within PrimeTime SI", *Synopsys User Group Conference*, 2008.

[244] Z. Zhang, X. Kavousianos, K. Chakrabarty and Y. Tsiatouhas, "A Robust and Reconfigurable Multi-mode Power Gating Architecture", *Proc. International Conference on VLSI Design*, 2011, pp. 280–285.

[245] H. Zheng and Z. Zhu, "Power and Performance Trade-Offs in Contemporary DRAM System Designs for Multicore Processors", *IEEE Transactions on Computers* 59(8) (2010), pp. 1033–1046.

[246] N. Zhu, W. L. Goh and K. S. Yeo, "An Enhanced Low-Power High-Speed Adder for Error-Tolerant Application", *Proc. International Symposium on Integrated Circuits*, 2009, pp. 69–72.

[247] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo and Z. H. Kong, "Design of Low-Power High-Speed Truncation-Error-Tolerant Adder and Its Application in Digital Signal Processing", *IEEE Transactions on Very Large Scale Integration Systems* 18(8) (2010), pp. 1225–1229.

[248] M. Ziegler and M. Stan, "Optimal Logarithmic Adder Structures with a Fanout of Two for Minimizing the Area-Delay Product", *Proc. IEEE International Symposium on Circuits and Systems*, 2001, pp. 657–660.

[249] V. Zyuban and P. Strenski, “Unified Methodology for Resolving Power-Performance Tradeoffs at the Microarchitectural and Circuit Levels”, *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, 2002, pp. 166–171.