University of California
Los Angeles

Energy Minimization under Uncertainty using Coordinated Multi-phase Synthesis Techniques

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

by

Nathaniel Alcala Conos

2014

Abstract of the Dissertation

Energy Minimization under Uncertainty using Coordinated Multi-phase Synthesis Techniques

by

Nathaniel Alcala Conos

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Miodrag Potkonjak, Chair

Energy minimization is one of the premiere design objectives in modern integrated circuits (ICs). Currently, there is a pressing need to reduce energy consumption in systems that span a wide array of form factors, ranging from small sensor networks, mobile phones, and tablets [30][31][182]-[189], where simultaneously maximizing battery lifetime and satisfying user experience is of paramount importance, to data centers/super-computers, where even a small reduction in energy can translate to billions of dollars saved in operating costs. However, as transistors continue to scale deeper into the sub-micron regime, producing energy efficient designs has become more challenging. Leakage power has increased significantly with respect to its total power contribution, which in turn is exponentially dependent on operating temperature; moreover, this is further exacerbated by increased device densities. Furthermore, the impact of process variation in the design flow under these scenarios requires increased attention, as small random alterations on a device (e.g., threshold-voltage variations) can greatly impact overall energy and delay yields. Thus, as we continue to delve into the billion transistor era and beyond, new techniques are needed to adapt to the continuously evolving physical landscape of IC technology.

This thesis presents several systematic and coordinated methods that simul-

taneously address energy and performance objectives for nano-scale technologies. We introduce a multi-phase IC synthesis framework with an emphasis on optimization parameters that, in recent years, have become more pronounced in near- and super-threshold technology regimes; these parameters include gate switching activity, input vector control, load capacitance, and operating temperature. We present new gate-level and structural transformation techniques that, when performed in a coordinated fashion, enable more energy efficient designs later in the design flow. These techniques include gate sizing and threshold-voltage selection, circuit unfolding, and retiming. Each technique accounts for the aforementioned parameters in generating ultra-low energy designs that satisfy the specified performance target. We also present a scenario-based approach for optimization under uncertainty in order to address the impact of process variation.

The dissertation of Nathaniel Alcala Conos is approved.

Glenn Reinman

Dejan Markovic

Milos Ercegovac

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2014

To my parents – for providing the environment
that enabled me to dream big.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my research advisor, Prof. Miodrag Potkonjak, for the guidance and the patience that he has provided to me during my Ph.D. studies at UCLA. I appreciate the life stories and endless advices that he has given.

I am grateful to my dissertation committee members Prof. Milos Ercegovac, Prof. Dejan Markovic, and Prof. Glenn Reinman, for their insightful comments and advices towards completion of my dissertation. I would also like to thank the computer science graduate staff, Steve Arbuckle and Craig Jessen, and the math department staff, Leticia Dominguez, for their administrative support. Additionally, I would like to acknowledge the Eugene V. Cota-Robles fellowship support in my first four years of graduate study at UCLA.

I would like to extend my deep gratitude to each of my supervisors, Manish Pagey (Space Micro Inc.), Boyd Carter (Aerospace Corp.), Robert Rush Jr. (SPAWAR SSC-PAC), Micah Barany and Sundaram Chinthamani (Intel Corp.) for allowing me to gain valuable industrial experience. I would also like to thank the UC LEADS program at UC Riverside and Prof. Walid Najjar for exposing me to research early in my undergraduate studies.

To my colleagues, Eun-Sook Sung, Vishwa Goudar, Jong Ahnn, Foad Dabiri and Sheng Wei, I would like thank them for their valuable technical advice. I would also like to acknowledge their respective contributions that helped make this thesis possible. Chapters 3, 5, 8, and 9 were done collaboration with Saro Meguerdichian and are based on publications P1, P2, P7 and P9. Chapter 5 includes contribution from Sheng Wei. Chapter 8 includes contribution from Foad Dabiri. All work presented in this thesis was done in close supervision by my research advisor, Prof. Miodrag Potkonjak.

Lastly, I would like to thank my close family and friends and for their limitless

love and support. To my best friends, Julian Cabais, Jaypee Punzal, Anthony Scarlett, Chad Catbagan, Brian Guzman, and Henry Flores – I would like to especially thank them for their moral support during my Ph.D. studies. To my parents, Resurreccion and Joseph Cregg, I would like to thank them for the many sacrifices they made in order to provide better opportunities for my brothers, Ryan and Rio Vir Conos, and me – this work would not have been possible without them.

Vita

2003–2007	B.S. (Computer Science), UCR.
2007–2010	M.S. (Computer Science), UCLA.
2012–2013	Teaching Assistant, Math Department, UCLA.
2010–2014	Teaching Assistant, Computer Science Department, UCLA.
2007–2014	Graduate Student, Computer Science Department, UCLA.

Publications

P1. N. A. Conos, S. Meguerdichian, and M. Potkonjak, "Coordinated and Adaptive Power Gating and Dynamic Voltage Scaling for Energy Minimization," accepted for publication *Application-specific Systems, Architectures and Processors (ASAP)*, 2014

P2. N. A. Conos, S. Meguerdichian, F. Dabiri, and M. Potkonjak, "Provably Minimal Energy using Coordinated Power Gating and DVS,," *Design Automation Test Europe (DATE)*, pp. 1–6 2014

P3. N. A. Conos, T. Xu and M. Potkonjak, "Retiming for Dual Supply Voltages," under review, 2014

P4. N. A. Conos and M. Potkonjak, "Sequential Circuit Unfolding for Simultaneous Energy and Delay Optimization in Near-Threshold Computing," under review

P5. J. B. Wendt, N. A. Conos and M. Potkonjak, "Multiple Constant Multiplication Implementations in Near-Threshold Computing Systems," under review, 2014

P6. N. A. Conos and M. Potkonkak, "Coordinated Synthesis Techniques for Power, Delay, and Yield Optimization in the Near-Threshold Regime," under review, 2014

P7. N. A. Conos, S. Meguerdichian, S. Wei, and M. Potkonjak, "Maximizing Yield in Near-Threshold Computing under the Presence of Process Variation," *Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp 1–8, 2013

P8. N. A. Conos and M. Potkonjak, "A Temperature-aware Synthesis Technique for Simultaneous Delay and Leakage Optimization," pp 361-321, *International Conference in Computer Design (ICCD)*, pp. 316–321, 2013

P9. N. A. Conos, S. Meguerdichian, and M. Potkonjak, "Gate Sizing in the Presence of Switching-activity and Input Vector Control," pp 138–143, *International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 138–143, 2013

P10. N. A. Conos, "An energy optimization methodology using a switching capacitance-based DVFS on bitwidth customized execution units," *Master's Thesis*, UCLA library, 2010

CHAPTER 1

Introduction

1.1 Motivation

Energy minimization is an important objective in the modern integrated circuit (IC) synthesis flow and can serve as a focal point for coordinating several optimization phases. First, the amount of *energy* that can be stored in a battery is limited. Therefore, minimizing energy consumption in both pre- and post-silicon phases is critical for extending the battery life for energy constrained systems. Starting with the pre-silicon phases, the most common objective is to identify an energy efficient design to meet a specified performance target. Once the system architecture has been signed off for tape-out, the post-silicon phases focus on power management strategies to minimize energy consumption and is achieved by coordinating and adapting resources through dynamic voltage scaling (DVS), and clock and/or power-gating. Synergistic approaches that coordinate the intermediate steps between these phases are becoming increasingly important in order to maximize energy efficiency, since battery technology advancements have historically lagged behind in IC improvements. Therefore, energy-centric methodologies are becoming the major focus as advancements in process technology continue to trend in accordance to Moore's Law.

Energy-centric approaches have also taken center stage for non-energy constrained systems, such as PCs and server-class systems, since they are plugged into a power source. However, these systems account for *power*, the instantaneous

representation of energy with respect to *time*. Several design challenges that arise in order manage power. For instance, identifying the optimal power configuration that achieve minimum energy for a given speed is non-trivial, due to the increased relative contribution of leakage power and its non-linear/non-convex relationship in deep submicron technology [18][181]. The problems in these domains are generally NP-Hard and require the deployment of complex and intricate heuristics that are technology-process dependent for obtaining solutions at the gate-level and behavioral-level [97][127]. At the gate or cell-level, standard cell libraries are the de facto standard for design for very-large-scale-integration (VLSI) systems and are combinatorial, since cells are available at discrete sizes and/or threshold-voltage (V_{DD}) configurations [58][59][60][115]. Additionally, another major challenge to address scalability, since the sheer size of modern circuits, now in the billions, is expected to grow. Advancements in transistor density have enabled the realization of complex system-on-chips (SoC). However, power budgets have remained relatively constant and have not kept in pace with circuit transistor density. Therefore, without proper consideration of power, achieved yields may worsen due to low fidelity present in the pre-silicon phases. Additionally, in the post-silicon phase, manufactured designs may have increased power delivery and circuit reliability issues, and are further impacted by environmental effects such as temperature.

The second term that impacts energy, *time*, can be tied to the clock rate or throughput (e.g., operations/cycle) to represent a performance metric. The performance target is driven by the market in order to meet consumer needs (e.g., high-performance computing, high-definition graphics, etc.). Thus, all phases of the synthesis flow are required to address energy minimization such that the performance or timing target is satisfied. In this thesis, we present novel pre-and -post-silicon energy minimization strategies, where energy is minimized to meet a specified performance (e.g., delay) constraint.

In recent years, optimization under uncertainty has become a major paradigm shift in IC design in the deep submicron regime. In this thesis, we classify uncertainty into three major groups:

- *Technological uncertainty* - dependent on the process technology generation and is directly impacted by process variation (PV) factors. The sensitivity of parameters that impact the power and delay of a device have varied across each technology generation. Therefore, academic and commercial tools are required to be continuously refreshed and new coordinated techniques are needed to adapt to fully leverage physical device-level properties.

- *Application uncertainty* - the behavior of an application on a given microarchitectural design. The behavior of an application, which is often categorized by its duty-cycle, active and idle periods, has a direct impact on how energy is consumed in the gate-level (e.g., switching, leakage vector state), which further impacts and is impacted by environmental factors, such as temperature hotspot generations on a chip [113].

- *Optimization uncertainty* - the numerous phases of optimization conducted in the overall chip design flow. The design flow is divided into several phases in order to address the design complexity of modern billion-transistor designs; thus, although one phase may yield positive benefits, it may negatively impact another.

Therefore, severe ramifications can result in generated solutions that impact power, energy, performance, and obtain yields of a design, if the aforementioned uncertainty factors in the design flow are not properly addressed, which in turn increase design costs.

In this thesis, we address energy minimization under uncertainty, where energy is minimized to satisfy a target performance constraint. Energy minimization is

conducted in such a way where specified uncertainty factors are also accounted for. The next subsection highlights our contributions and thesis organization.

1.2 Contributions and Organization

We develop a coordinated multi-phase integrated circuit synthesis for conducting energy minimization under uncertainty at both low (gate-level) and high-level (behavioral) synthesis domains. Chapters 1 to 7 present gate-level pre-silicon techniques for energy minimization. Chapters 8 and 9, present behavioral-level in both pre- and -post-silicon phases. The following detailed contributions listed below:

- A gate-sizing method is presented that utilizes accurate gate-switching, input vector control (IVC), and duty-cycle (active and idle) knowledge to enable more efficient power and speed trade-offs (Chapter 3).

- Standby Leakage energy and circuit delay are simultaneously optimized using a temperature-aware gate-sizing and threshold-voltage (V_{TH}) technique. The presented approach coordinates several leakage minimization techniques, such as IVC, gate-replacement, and pin-reordering in optimizing leakage and delay (Chapter 4). The incremental improvement of each applied step is presented and showing benefits when utilizing accurate temperature knowledge.

- Process variation is addressed using a gate-sizing and V_{TH} technique in (Chapter 5); the objective is to simultaneously maximize the circuit yield (e.g., maximize number of circuit instances that meet timing requirements) and minimize energy consumption. The technique is applied on near-threshold regime and improvements are shown when using our PV-aware approach over a state-of-the-art gate sizing method that does not.

- Chapter 6 presents a coordinated circuit unfolding and gate-sizing/V_{TH} selection technique to improve delay and energy yield targets. The yield target is improved by: 1) increasing the performance per iteration; 2) minimizing the impact of PV by increasing the logic depth; and 3) enabling additional optimization opportunity through gate-sizing/V_{TH} selection techniques in deep-logic design.

- Retiming is combined with dual supply-voltage (dual-V_{DD}) techniques for enabling additional energy reductions (Chapter 7). The circuit is transformed using retiming and min-cut techniques to enable additional improvements when applying dual-V_{DD} application. Additionally, an efficient heuristic is presented, which finds the best high and low V_{DD} pair that achieves minimal energy under a delay constraint.

- Hardware adaptation is investigated in (Chapter 8) and a convex programming formulation is presented that utilizes at most two hardware allocations to achieve minimum energy under a specified timing constraint. Hardware allocations are realized using a combination of power gating and dynamic voltage scaling (DVS) techniques and is proven to be minimal under specified coarse-grained hardware adaptation assumptions.

- System customization and two fine-grained hardware adaptation (post-silicon) techniques are presented in Chapter 9. The first step presents a dynamic programming solution which identifies the best set of hardware allocations (e.g., caches, ALUs, etc.) given a set of predicted tasks or applications to run. Then, given the actual set of tasks to run, the next step statically assigns static hardware allocation at fine task granularity, each tailored to solve optimization scenarios: 1) where it is assumed that hardware adaptation transition overheads negligible; and 2) non-negligible.

CHAPTER 2

Preliminaries

In this chapter, we introduce basic power and delay, and process variation models that we use throughout the body of work in this thesis. Additional preliminaries material may be presented in their respective chapters.

2.1 Power and Delay

We use the delay and power models proposed in [154] in our design process. These models connect gate-level delay and power properties with the physical level properties such as gate width (W), gate length (L), and threshold voltage (V_{TH}). Equation (2.1) shows the gate-level delay model, with supply voltage V_{DD}, sub-threshold slope n, mobility μ, oxide capacitance C_{ox}, gate width W, gate length L, thermal voltage $\phi_t = (kT/q)$, drain-induced barrier lowering (DIBL) factor σ, threshold voltage V_{TH}, and delay and model fitting parameters k_{tp} and k_{fit}. Load capacitance C_L is proportional to the sum of the interconnect and driving pin capacitance's of its fan-out gates. Thus, assuming the interconnect to be fixed, the gate propagation delay (t_p) is affected by the sizing of its immediate fan-out cell(s), requiring careful sizing options to be selected during the sizing procedure.

$$t_p = \frac{k_{tp} \cdot C_L \cdot V_{dd}}{2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (\frac{kT}{q})^2} \cdot \frac{k_{fit}}{(ln(e^{\frac{(1+\sigma)V_{dd}V_{th}}{2 \cdot n \cdot (kT/q)}} + 1))^2} \tag{2.1}$$

Note that the parameters other than W, L, and V_{TH} are transistor level properties that can be derived using transistor-level simulation. Since they are not affected by PV and therefore do not impact the design process, we assume they

are constant values in the model. Equation (2.2) describes the leakage power model that depends on the value of W/L.

$$P_{leakage} = 2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (\frac{kT}{q})^2 \cdot V_{dd} \cdot e^{\frac{\sigma \cdot V_{dd} V_{th}}{n \cdot (kT/q)}} \tag{2.2}$$

The gate-level switching power model [154] is described by Equation (2.3), where α is the activity factor and f is the frequency.

$$P_{switching} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \tag{2.3}$$

The delay and power models indicate the trade-off between delay and power in terms of gate sizing. For example, increasing W of a gate typically reduces its propagation delay but increases its leakage and switching powers. Simultaneously, the decision increases the delays and switching powers of the transitive-input gates of the directly affected gate due to the increase in their capacitive loads. In works that conduct gate sizing (chapters 3, 4, 5, and 6), the goal is to determine an optimized set of gate widths and V_{TH}'s to meet specified delay and power requirements.

2.2 Standard Cell Library Modeling

The previous section presented detailed formulations for modeling delay and power. In this sub-section, we present higher-level models and algorithms for adapting formulations into a standard-cell library.

The total energy of a CMOS integrated circuit can be characterized into two main components: 1) dynamic (switching) energy due to charging of input pin/output load capacitance's; and 2) static (leakage) energy, which we model from the dominant sub-threshold leakage and gate leakage currents. Thus, the

total energy consumed can be computed as:

$$E_{total} = E_{switch} + E_{leak} \tag{2.4}$$

$$E_{switch} = \sum_{i}^{N} es(g_i), \qquad E_{leak} = \sum_{i}^{N} el(g_i) \tag{2.5}$$

es and el represent the switching and leakage energies, respectively, for gate g_i. es is the product between probability that a gate's input pin j will switch, α, and the estimated full-cycle power consumed from propagating a signal from input pin j to output pin k. el is the sum of leakage energies consumed at each possible leakage state of a gate, which is also dependent the ratio of the total time spent at each leakage state for both *active* and *standby* (idle) periods. The total time (T) is directly proportional to product of the circuit delay (D) and total cycles, where D represents the critical output-pin arrival time ($rise$ or $fall$) of a primary output gate $ot^{r,f}(g_i)$:

$$D = max(ot^{r,f}(g_i)) \qquad s.t.\ g_i \in G_{out} \tag{2.6}$$

G_{out} represents a circuit's set of primary output gates. Therefore, the delay of a circuit can be determined by solving:

$$ot^{r,f}(g_i) = dl^{r,f}(g_i) + max(ot(fin_j^{f,r})) \qquad s.t.\ fin_i \in FI_i \tag{2.7}$$

$fin_j^{f,r}$ is the $fall$, $rise$ arrival time of a fan-in gate j in the set FI_i of gate g_i. Note that the propagation of delay depends on the unateness assumption. For simplicity, we assume all cells are negative unate, thus, rise (r) and fall (f) gate delays are propagated as assumed to the next stage.

We use a standard cell table library look-up, as presented in [96], to model gate $rise$ and $fall$ delay ($dl^{r,f}$) as a function of its input slew (transition time), and driving load. However, we use the relative weights from an alternate 45 nm cell library (Nandgate) [118] to account for switching and input vector dependent

leakage power (transistor stacking effect), which are obtained in a similar look-up table fashion, provided per-input pin accurate switching, and input vector state probabilities and can be obtained using gate-level simulation. The obtained weights were re-mapped to its technology generation in generating respective fitting parameters for consistency.

2.3 Process Variation

We capture the impact of PV using a quad-tree model proposed by Cline et al. [64], which considers the spatial correlations among gates. Effective channel length L is modeled after a random distribution or a combination of multiple distributions to reflect both spatial and inter-chip correlation and variations. In the quad-tree model, the effective channel length subject to PV is distributed into multiple levels, with a different number of grids allocated on each level. In our approach, we classify a cell's grid location (i, j) as a combination of its longest logic depth stemming from its primary input (i) and primary output (j). The grids on each level are assigned variation values that follow a normal distribution. The effective channel length for a particular gate is computed by the sum of the variations on each level of the grids to which the pertinent gate belongs. The total value of the target gate-level property as the sum of the variations on each level of the grids to which the corresponding gate belongs.

We show the quad-tree model for L in Equation (2.8), where ΔL_{ij} is the quantitative variation of the i-th level and j-th grid to which the gate belongs, and μ_i and σ_i are parameters of the normal distribution at level i.

$$\Delta L = \sum_i \Delta L_{ij}, \qquad where \quad \Delta L_{ij} \sim N(\mu_i, \sigma_i) \tag{2.8}$$

We adopt the Gaussian distribution proposed by Asenov et al. [65] for V_{TH} and is based on the physical simulation of random dopant distributions. Note that gates are not correlated in terms of V_{TH}.

CHAPTER 3

Gate Sizing in the Presence of Switching Activity and Input Vector Control

In this chapter, we introduce a novel gate sizing approach that considers both the gate switching activity (SA) and gate input vector control leakage (IVC). We first extract SA using simulation and find promising input vectors. Next, in an iterative framework, we interchangeably conduct gate sizing and refining the IVC. As dictated by the new objective function, our algorithm conducts iterative gate freezing and unlocking with cut-based search for the most beneficial gate sizes under delay constraints. We evaluate our approach on standard benchmarks in 45 nm technology, showing promising improvement, achieving up to 62% (29% avg.) energy savings compared to the traditional objective function.

3.1 Introduction

Gate sizing is a powerful optimization technique used to minimize power and/or area under strict timing constraints by altering the widths of transistors in gates. Gate sizing has been extensively studied over the past three decades [114][115][96] and several approaches have been proposed. Previous approaches, however, do not consider switching activity (SA) and the impact of input vector control leakage (IVC), which greatly impact the overall optimization strategy. Furthermore, gate sizing is often combined with dual or multi-threshold techniques which further increases the importance of accurate power and delay modeling. As a result, the

modern design flow imposes a number of modeling and optimization challenges

A major challenge is the simplification of timing and power models, which may lead to suboptimal solutions when mapping out to real designs. Accounting for accurate gate and interconnect delay and its dependencies on capacitive load slew are often ignored [96]. Additionally, nominal gate switching activity and/or average gate leakage are generally assumed in previous works limiting the potential improvements by accounting for realistic operating conditions. Moreover, previous approaches are either dynamic or leakage power-centric in their optimization flows, which do not address the varying application usage characteristics present in high-performance systems (e.g., data-centers, super-computing) to energy constrained mobile devices (e.g., sensors tablets, smart-phones).

Cell library-based optimization has emerged as the de facto standard for modeling power and delay of a circuit design [115]. Many previous approaches, however, utilize simplified timing models by assuming convex and/or linear delay and power models [116]. Empirical analysis has shown that accurate timing models are non-linear/non-convex. Furthermore, optimizing circuit designs using a discrete cell library, however, leads to solving an NP-Hard problem [97]. As a result, many heuristics have been developed in order to address the huge problem search space. A major drawback of these methods is that they require heavy parameter tuning and are difficult reproduce, since they are technology dependent, and are rooted to a set of sensitivity functions. These methods often perform iterative per-gate or per-group improvement are too compute intensive and are impractical to be applied on modern IC sizes, even with incremental updates. Furthermore, these approaches mainly perform local optimization (i.e., local-moves) and are susceptible to be trapped in local minimas [117].

The usage of modern cell libraries, however, have enabled the support various supply/threshold voltages, and drive strengths, enabling a rich performance and energy trade-off to address the potentially vastly differing device usage charac-

teristics. However, current tools do not account for realistic conditions into their objective functions (e.g., gate activity, duty cycle, input vector control), with respect to their applications, potentially impacting obtained results.

We improve state-of-the-art sizing methodologies by simultaneously considering gate switching activity (SA) and gate input vector control (IVC). One of our key contributions is that we demonstrate significant benefits of incorporating actual gate SA and gate IVC in the objective function over the equivalent approach that only uses nominal and average values for switching and leakage weights, and show how the obtained solution varies when accounting varying duty cycles.

The focal point of our approach is a scalable gate sizing algorithm that considers gate SA and IVC leakage. The steps are to: 1) extract the SA of gates based on simulation of real workloads; and 2) conduct IVC to obtain the input vector that induces the lowest total leakage energy across all gates, and 3) an iterative gate sizing approach freezes maximally-constrained gates (ones that are at high-power states as determined by SA and IVC) while searching for a sizing option that best improves the current picture. The objective function in step 3 to be considered at the iteration depends on the types of options available and their impacts on both delay and energy. The algorithm prevents the algorithm from reaching a local minima by freezing gates as they are sized until all gates have been frozen, then unfreezes all gates, re-conducts IVC (since new gates may be energy-dominant), and reiterates steps 2 and 3 until the solution converges or the delay constraint cannot be met.

We evaluate our approach on benchmarks included in ISCAS-85/89, ITC99 and arithmetic units. Our results indicate over 62% (29% avg.) energy improvement over a method that assumes nominal SA and IVC, demonstrating that gate SA and IVC play an major role in the guiding sizing decisions.

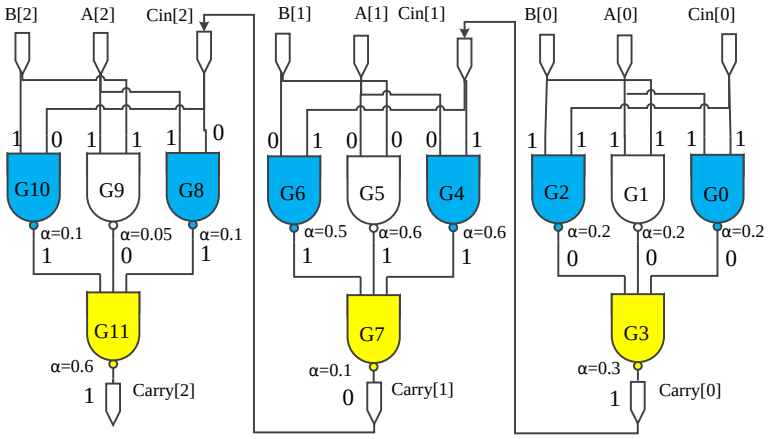


Figure 3.1: Carry propagation for 3-bit carry-ripple adder.

3.2 Motivation

We begin by providing a small realistic example demonstrating the importance of considering both SA and IVC in the gate sizing optimization process. Consider the carry propagation of a 3-bit carry-ripple adder, shown in Figure 3.1. Assume that 2- and 3-input NAND gates have input-dependent leakage power consumption values for two possible sizes, small (X1) and large (X2), shown in Table 3.1. Also assume that the given input vectors ($A = 101$, $B = 101$, and $C_{in} = 1$) are realized throughout the entire duration of the application. Figure 3.1 shows the input vectors to each gate. Therefore, the overall leakage power of the circuit is 288 nW. For simplicity of the example, ignoring load and slew dependencies, assume that all gates have delay of 10 ps at size X1 and 5 ps at size X2. Finally, assume that at the beginning of the optimization process, all gates are nominally sized to size X1. Therefore, there are eight nominal critical paths (colored), $\{G0, G2\} \rightarrow G3 \rightarrow \{G4, G6\} \rightarrow G7 \rightarrow \{G8, G10\} \rightarrow G11$, with nominal delay 60 ps. Consequently, total leakage energy of the circuit is 1.73×10^{-17} J.

As an example, consider a delay constraint of 55 ps, it is clear that one of gates G3, G7, or G11 should be sized up to X2, as all critical paths pass through these bottleneck gates and decrease the delay of each of these gates will decrease

the overall delay. A traditional approach to gate sizing would consider these gates equally. In other words, increasing the size of either would decrease delay and increase switching and leakage power by the same amounts. However, from Table 3.1, we see that the leakage power of a gate, due to transistor stacking, strongly depends on its applied inputs, with up to a 43X difference between the lowest-leakage state (input vector "100": 1.29 nW) and highest-leakage state (input vector "111": 55.8 nW) of a 3-input NAND gate. Furthermore, switching energy of a gate is directly proportional to its activity factor, or the likelihood that the gate will switch. Therefore, because the gates have both different applied input vectors and different activity factors, sizing up each one will have a different effect on overall power and energy consumption, so they should not be weighted equally in the optimization process.

First, consider the case where the duty cycle of the adder is low and therefore leakage energy dominates. We can determine from Table 3.1 that increasing the size of gates G3, G7, or G11 will increase leakage power by 9.96 nW, 167.42 nW, or 56.35 nW, respectively, while decreasing the overall delay by 5 ps. Therefore, the optimal decision is to increase the size of gate G3, which will have minimal impact on leakage energy, increasing leakage power to 298 nW and *decreasing* leakage energy to 1.64×10^{-17} J. Increasing the size of G7 would instead increase leakage power to 455 nW, *increasing* leakage energy to 2.50×10^{-17} J. Thus, considering IVC in this example in the optimization algorithm can improve the energy by roughly 60%.

Now, consider the high duty cycle scenario, where switching energy is the dominant factor. Again, for simplicity, assume that all gates consume 10 nJ and 20 nJ of switching energy at nominal activity factor 1.0 for a given application at sizes X1 and X2, respectively. Figure 3.1 shows the activity factors (α) for each gate. Therefore, overall switching energy consumption at the nominal size is 35.5 nJ. In this case, increasing the size of gate G7 is the optimal decision, since it has

Table 3.1: NAND gate leakage values (nW) for two sizes (X1, X2) based on input vector control (IVC) from a single threshold 45 nm cell library [118], where min and max leakage states are represented by bold and italicized fonts, respectively.

	NAND-3	
IVC	X1	X2
000	3.32	13.28
001	18.18	72.73
010	4.21	16.84
011	39.49	157.97
100	**1.29**	**5.15**
101	18.78	75.13
110	3.76	15.04
111	*55.8*	*223.22*

	NAND-2	
IVC	X1	X2
00	**3.48**	**13.93**
01	24.8	99.2
10	4.09	16.34
11	*37.21*	*148.83*

the lowest activity factor and consumes less switching energy than when up-sizing either G3 or G11. In fact, this decision results in a switching energy of 36.5 nJ, whereas increasing the size of G11 would result in a switching energy of 41.5 nJ. Therefore, the decision that considers SA performs roughly 14% better.

To present these motivations, we have made a number of assumptions that when relaxed make the optimization much more complex in practice. It is reasonable to assume that additional information (gate switching, input vector state) can be readily obtained by modern CAD tools and/or by implementing a simple gate-level simulator. Such information is beneficial since it enables the simultaneous consideration of low duty cycle and high duty cycle scenarios, as in real use cases at current and pending deep submicron feature sizes, leakage and switching may both have significant impacts on overall energy. For example, sizing up G7 in the high duty cycle scenario may in reality not be optimal, since its input gates have

higher values for α than, the input gates of G3, and thus their switching energies would increase by larger factors. Thus, this IVC depends on how the circuit is sized and its duty-cycle. Therefore, a feedback loop exists between gate sizing and IVC that must be addressed simultaneously during the optimization. The simple example here demonstrates that both IVC and SA are crucial considerations in gate sizing for energy optimization in the presence of delay deadlines.

3.3 Related Work

We now cover a set of related gate sizing approaches that have considered a variant of SA or IVC. Several approaches exist that address continuous and discrete gate sizing. Common methods to solve the gate sizing problem have been convex optimization [116], Lagrangian Relaxation [114][115][60], and gradient and sensitivity-based optimization [58][59].

Gate sizing methods have also been combined with V_{dd} and V_{TH} assignment to minimize power under various gate SA ratios [119][120]. These works, however, have only considered average leakage values when accounting for leakage and have not explored real application activity factors when considering gate switching activity. Leakage minimization using IVC is a popular technique for due to its strong dependency on the input vector state [99]. IVC and gate replacement techniques have also been combined [121] by replacing gates at their worse-case leakage state with equivalent gates with lower leakage power. The technique is further combined with circuit aging in pre- and-post silicon phases. [122][126]. IVC has also been explored in the presence of uncertainty [73].

To the best of our knowledge, we are the first to consider gate sizing in the presence of both SA, IVC, and duty cycle. Prior approaches have at most considered one or two terms accurately [125], and/or do not differentiate between the duty cycle with respect to switching and leakage energy weights, leaving many approaches

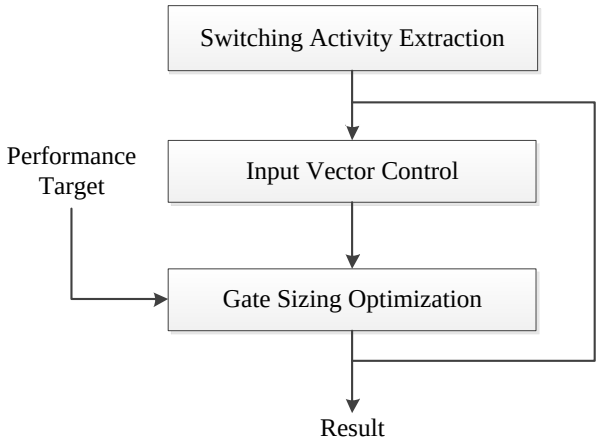


Figure 3.2: Gate sizing optimization flow.

to be either dynamic or leakage power-centric. For example, the state-of-the-art gate sizing contest considers only nominal leakage power [96]. Our technique minimizes total energy, such that both the switching and leakage energy components are accurately accounted for in accordance to their usage or duty cycle.

3.4 Technical Approach

Our gate sizing procedure is composed of three major phases (Figure 3.2). The first phase extracts gate switching activity factors (SA) for a given circuit by performing event-driven gate simulation from a set of input bit vectors. Figure 3.3 illustrates an example SA extraction for a carry-look-ahead unit (cla4) from two applications (mpeg2enc/dec). The second step identifies a primary input bit vector that places gates in low leakage states in order to minimize the total energy of the circuit, which accounts for leakage consumption for both active (obtained from SA) and idle periods. IVC techniques range from random simulation to satisfiability (SAT) and model counting-based formulations. The final component is the gate sizing algorithm, where the goal is to minimize total energy consumption under a delay constraint. The approach is iterative; at each iteration, gates are either frozen or unlocked based on their leakage (IVC) and switching (SA) impact,

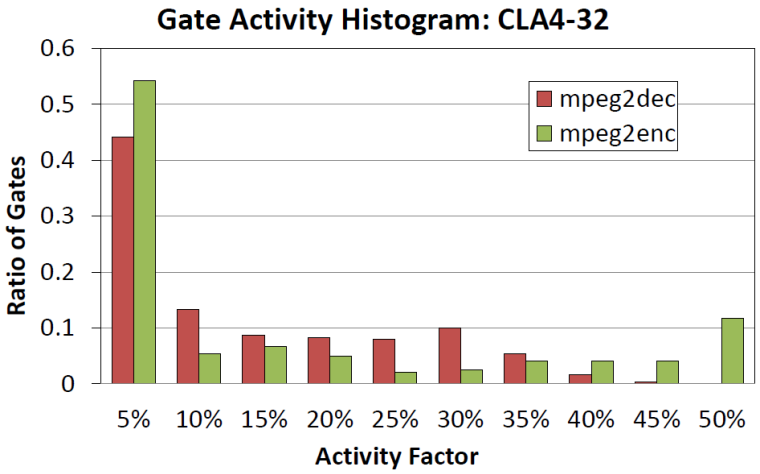


Figure 3.3: Gate switching activity for a 32-bit CLA circuit when using real mpeg2enc/dec application input stimulus. Shown are varying distribution of gate activity within a circuit and across two applications.

while a search is conducted for the most beneficial current *move*.

Our algorithm is sensitivity-based in nature in terms of determining which move or set of moves to perform. A gate sizing move can have 1 of 3 effects (increase, decrease, have no effect) on 2 parameters (energy and delay), leading to a total of 9 separate possible classes for a move. The algorithm classifies each move to a class and enforces a priority in terms of selecting a move that has higher precedence. There are three precedence levels, where level 1 is the highest priority. Moves that improve both parameters are at precedence 1, moves that improve just one parameter and do not affect the other are at precedence 2, and moves that improve one parameter at the expense of degrading another are of precedence 3. Note that moves that degrade both parameters are never selected. Each precedence level has its own objective function for selecting the best move: 1) the product of the respective improvements; 2) the single improvement; and 3) the normalized ratio of improvement and degradation.

The algorithm considers a cut of M gates at a time and restricts one gate to

be sized per group visit. Once a size move is committed, the gate is locked and is no longer considered within that phase. The completion of a phase is defined as having locked all gates, or having no more acceptable moves among sizable gates that improve the objective function. The algorithm terminates after the solution converges or if a target delay (D_{target}) can not be met after a number of phases. All gates are unlocked before the start of each new sizing phase.

The algorithm initially freezes the top K energy-critical gates by setting them to their minimal sizes at the beginning of the phase. We note that this initial set potentially restricts some delay critical gates, as improving the delay of these gates may be required in meeting a deadline. To relax this constraint (i.e., if a solution cannot be obtained), K is relaxed through a locking threshold ratio γ, where a new K is computed (e.g., $K = K^{'} \cdot \gamma$), thereby, enabling potentially more delay critical gates to be reduced. It is crucial to identify the top K energy-critical gate, which in turn depends on both SA and IVC; this maximally-constrained gate locking is one of the key innovations of the approach, and prevents being trapped at a local minima by encouraging global circuit optimization.

We utilize an epsilon tree to minimize circuit delay updates (ϵ_{path}), which consists of gates that were on the critical path during the last accurate delay computation (Figure 3.4a). Since the critical path may change during optimization, we also include the immediate fan-out gates of each critical gate (e.g., nodes 1 and 3), fan-in nodes may be added for greater accuracy as their slews may also impact timing propagation. The figure shows bold-outlined nodes (e.g., 7, 8, 9, 5 and 6) are the primary outputs (G_{out}), and are transitively connected to at least one node belonging to the critical path (e.g., 0, 2, 4, 8). Thus, the delay cost of sizing a gate on the ϵ_{path} can be estimated by the sum of its δ_i respect to the target delay (D_{target}) is used to estimate the delay impact of each move via a delay cost formula, as shown below:

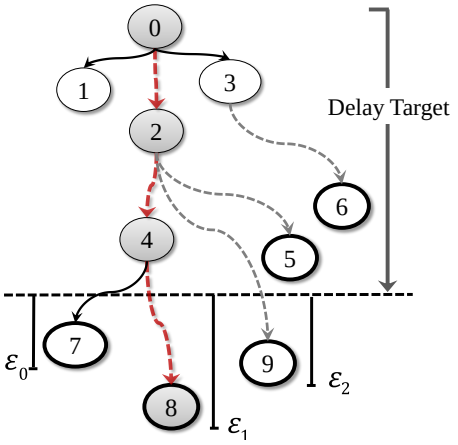
$$D_{cost} = \sum_{i}^{|G_{out}|} (\delta_i - D_{target})^2 \tag{3.1}$$

This formulation enables very efficient delay estimation by only considering the delay impact of a small subset of gates at a time. A drawback of this approach, however, is that a potentially new critical path may emerge. This remains to be a major challenge for existing gate sizing techniques that attempt to maintain delay accuracy during optimization [58][60][59]. To address this issue, the frequency of delay updates can be increased by adjusting M and γ to be larger values, as we have done. These parameters can be adjusted, to trade-off accuracy vs run-time. Our used values of M and γ achieved a delay accuracy to be within 5%, while achieving linear run-time scaling with respect to circuit size (Figure 3.4b).

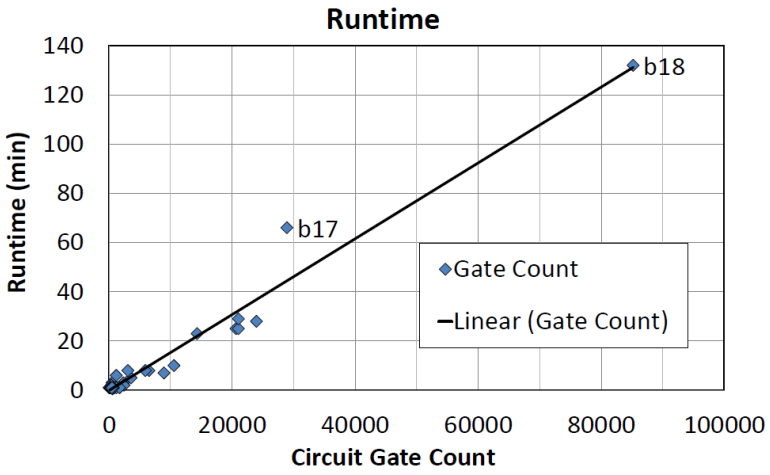
3.5 Experimental Results

We evaluated our gate sizing approach on a set of benchmarks in ISCAS-85/89, ITC-99 suites, as well as integer arithmetic units consisting of adders (carry ripple, carry-look-ahead, Kogge-Stone) and multipliers (array, Dadda). All units were synthesized using a single threshold (HVt) 45 nm open cell library from [118] under the typical cell configuration. An in-house timing/power engine was implemented in C++ and was correlated to an industrial tool, Synopsys PrimeTime, to be within 10^{-3}ps. All results were optimized using identical rules such as ensuring no slew or load violations exists in the final design, as presented in [96]. The only differences in our framework is the choice in cell library, which was done in order to enable accurate IVC computations, as well as the choice of circuit benchmarks. In handling slew and load violations, we adopt an iterative approach as proposed in [59].

The SA of gates and IVC for each circuit were obtained from simulation of random input vectors. However, real application switching activity factors were obtained from mpeg2enc/dec benchmarks from recorded operand values from each unit type, running ARM7TDMI mpeg2-enc/dec traces [123][124]. The initial sim-



(a)



(b)

Figure 3.4: (a) An example of ϵ critical path (ϵ_{path}); the critical path in red; transitive fan-out output nodes in bold outlines; and ϵ_i corresponds to the absolute delay difference w.r.t to the target delay used for estimating delay cost of a move. (b) The linear run-time of the new gate sizing approach.

ulation parameters set were, $K = 25\%$, $M =$ twice the length of average critical path, $\gamma = 0.2$, and were fixed across all benchmarks. The delay target for each circuit was set as the median between the achieved delay when all gates were set to their maximal size, and the achieved delay when all gates are at their minimal size. Five duty cycle scenarios (D0=10%, D1=20%, D2=33%, D3=50%, and

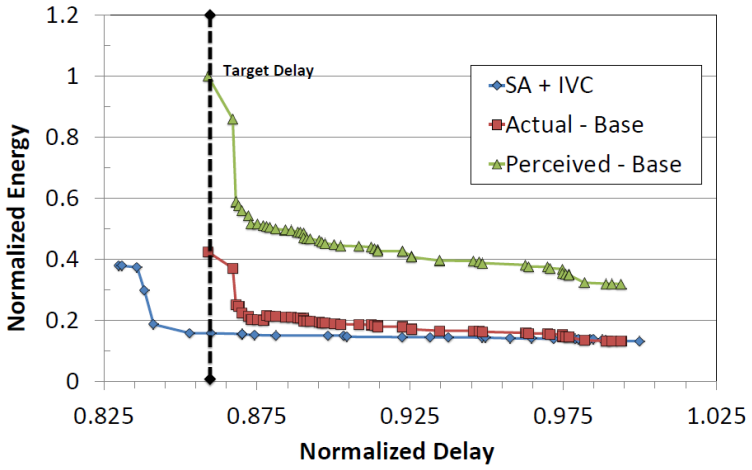


Figure 3.5: Energy vs delay plot of c2670. The $SA{+}IVC$ approach consistently outperforms the *Base* method.

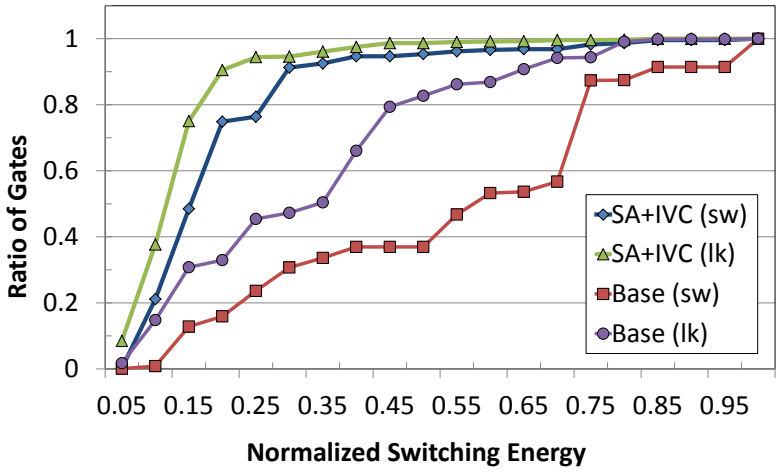


Figure 3.6: Cumulative distribution of leakage and switching energy after sizing for gates in c2670. The accurate $SA{+}IVC$ approach results in a higher percentage of gates at lower energy.

D4=100%) were considered.

We evaluate our sizing algorithm under two gate sizing assumptions: 1) SA+IVC, which considers gate switching activity factors and input vector control in the ob-

jective function; and 2) *Base*, where the objective function uses only nominal gate switching (50%) and average gate leakage values for total energy computation. Table 3.2 compares the two methods, where Max (%) savings corresponds to the maximum energy improvement achieved over the *Base* method across the five duty cycle cases (D0 to D4) for each circuit under the same timing constraint. As expected, the maximum improvement observed varies across duty cycles and circuits, motivating the advantage of utilizing accurate power and delay knowledge.

Table 3.4 provides overall energy improvements across the benchmarks suite. The results generated by the new approach achieved a max energy improvement of 62% for circuit c2670 and 29% average overall for the same delay. Figure 3.5 provides a normalized energy and delay plot for c2670, which illustrates the advantage of using more accurate power and delay information. A delay of 0.87 shows that the *Perceived* (green) energy deviates from the trend of the *Actual* (red) energy plot. In performing move-trace analysis, we noted that *Base* method caused the algorithm to over-size a few selected critical paths and encountered a timing wall much earlier, where as $SA{+}IVC$ was able to efficiently trade-off delay for an additional 0.05 delay units, as shown.

Figure 3.6 shows a cumulative distribution of gate switching and leakage energies of the max improved result for $SA + IVC$ over *Base* for circuit c2670. Our approach shows that accurate knowledge enabled the algorithm to efficiently guide the circuit to a lower energy state, as shown with higher percentage of gates falling under lower energy profiles for both leakage and switching energy. This is important to note since due to the difficulty of comparing gate sizing algorithms, many existing algorithms are sensitivity-based in nature, thus, the ability to guide an algorithm to determine more promising "moves" greatly impacts the optimization procedure.

Table 3.3 presents results comparing the minimal configuration found by $SA +$

IVC and the perceived minimal configuration obtained by $Base$. The minimum energy configuration determined was cla432 and dad8, optimized under the same timing constraints determined by the the multiplier. For these configurations, our approach shows additional savings in both leakage and switching categories where the majority of the savings for both cases (15% mpeg2enc, 25% mpeg2dec) were achieved by the multiplier circuit.

3.6 Summary

We present a new gate sizing approach that includes the switching activity (SA) and input vector control (IVC) to minimize overall energy. The new objective function has several ramifications on the optimization procedure, including the need for reiteration between gate sizing and input vector selection and freezing and unlocking of high-power gates. On a comprehensive set of benchmarks, from ISCAS-85/89, ITC-99, and arithmetic units, synthesized using 45 nm technology, we reduce average actual energy consumption by 30%. The approach is generic in the sense that thermal impacts and multi-V_{TH} can be easily addressed using the new optimization procedure.

Table 3.2: Energy improvements when considering gate SA and IVC during the gate sizing procedure over the *Base* method. The obtained switching and leakage energies are presented for the SA+IVC. The maximum energy deltas (Δ %), corresponds to the max difference in energy profile "perceived" by the *Base* method during optimization.

		Max Energy Improvement			$SA + IVC$					
Circuit	No. Gates	Total (%)	Sw (%)	Lk (%)	Sw (μJ)	Max. (Δ%)	Lk (μJ)	Max. (Δ%)	Duty Cycle	Delay (ns)
c880	383	8.14	8.16	8.05	125.11	31.22	15.75	3.01	D2	0.73
c1355	554	14.35	15.01	13.26	264.13	31.12	164.79	15.19	D1	0.74
c1908	932	13.41	13.72	9.41	405.7	29.57	32.06	6.75	D4	1.14
c7552	3568	43.65	44.02	41.32	1685	30.27	284.6	3.21	D2	1.20
c5315	2330	58.72	59.1	54.57	855.6	32.93	87.58	3.19	D4	1.34
c432	168	30.15	35.24	22.35	68.54	33.15	53.55	23.67	D1	0.62
c2670	1202	62.64	62.74	60.78	407.6	31.91	24.62	1.39	D4	0.85
c3540	1703	28.84	29.66	24.14	799.1	31.09	152.2	2.47	D2	0.79
s1488	698	46.38	46.49	44.81	182.6	35.54	14.09	12.05	D2	0.42
s1494	692	35.5	36.08	33.84	283.9	36.36	103.3	12.55	D1	0.42
s15850	10547	31.34	30.76	34.48	3803	33.69	662.1	4.98	D3	2.22
s838	473	32.49	38.89	27.17	139.3	31.76	199.8	10.84	D1	1.64
s5378	3054	16.76	30.08	15.06	826.6	36.2	7876	26.34	D0	0.68
s641	470	26.56	26.14	29.17	112.2	34.26	17.46	10.02	D3	1.90
s713	483	12.83	13.92	6.15	91.37	34.22	16.29	6.07	D3	1.98
s820	345	24.49	24.3	27.25	93.71	36.47	6.29	4.11	D2	0.29
s832	343	1.35	0.55	3.09	81.64	35.58	33.66	3.39	D1	0.29
s9234	5897	50.19	50.52	47.23	2041	32.51	242.5	4.43	D3	1.50
s953	477	14.26	14.38	13.02	144.6	34.45	14.84	12.34	D2	0.45
s38417	23963	53.2	53.85	46.17	6661	30.65	714.2	8.58	D3	1.29
s35932	21035	22.27	22.31	21.25	8317	29.28	332.1	37.26	D3	0.80
s38584	18161	29.01	31.52	28.75	5820	32.24	59720	4.77	D0	1.52
b01	54	55.43	55.41	55.84	13.57	30.68	0.83	61.96	D1	0.18
b02	33	14.23	14.07	17.23	10.54	30.17	0.53	54.57	D1	0.14
b03	190	58.83	58.94	44.38	36.46	30.99	0.38	66.37	D3	0.32
b04	738	47.38	47.65	38.1	174.9	31.43	6.01	65.87	D3	0.97
b05	631	22.52	22.64	20.38	122.2	32.54	6.72	55.68	D3	1.06
b06	57	23.32	27.23	13.48	13.28	31.19	6.29	55.03	D0	0.14
b07	493	18.28	26.9	15.23	89.02	33.71	291.5	62.41	D0	0.94
b08	203	35.99	33.73	37.66	38.07	32.01	48.38	63.25	D0	0.35
b09	198	33.72	41.65	23.39	54.04	32.56	54.41	61.36	D0	0.30
b10	204	44.69	45.06	32.43	40.56	34.45	1.55	54.83	D2	0.36
b11	633	35.53	35.79	31.74	250.9	32.37	18.77	61.52	D2	1.10
b12	1183	22.91	27.89	3.31	314.0	37.03	107.3	60.54	D1	0.61
b13	375	22.05	22.78	15.01	161.23	31.85	18.4	55.33	D1	0.33
b14	6498	28.42	28.65	26.21	3024	33.22	320.7	54.94	D2	1.31
b15	8920	27.62	27.7	26.75	1666	38.99	166.1	54.24	D3	1.58
b17	28911	21.25	25.29	20.79	8.53	38.76	80.58	55.99	D3	1.68
b18	85188	7.02	8.67	5.96	44.54	37.84	71.4	54.72	D1	2.10
b20	14322	27.85	28.57	24.07	3.66	33.2	0.74	55.8	D2	2.08
b21	20640	8.36	8.52	6.95	10.21	33.51	1.24	60.27	D2	2.02
cra32	225	38.32	45.14	37.76	6.72	44.90	92.22	32.09	D0	2.08
cla432	305	22.25	25.02	12.18	10.18	42.87	3.28	13.62	D2	0.35
ks32	611	24.2	23.18	24.63	17.64	44.76	40.83	14.6	D0	0.30
arr8	512	35.96	36.3	23.18	178.5	34.69	22.84	21.29	D1	0.81
dad8	542	30.35	30.99	36.33	101.3	35.83	9.12	11.98	D2	0.62

Table 3.3: Energy improvement of $(SA + IVC)$ over *Base* using extracted gate switching activity and input vector control from mpeg2enc/dec applications assuming a (D2) "33%" duty cycle. The units represent an single-adder (32b) and multiplier (8b) configuration of an ARM7TDMI core [124].

	Energy Improvement			cla432				dad8			
Application	Total (%)	Sw (%)	Lk (%)	Sw (μJ)	Sw Imp. (%)	Lk (μJ)	Lk Imp. (%)	Sw (μJ)	Sw Imp. (%)	Lk (μJ)	Lk Imp. (%)
mpeg2enc	15.31	17.15	8.55	742.7	4.73	628.7	1.09	2267	20.61	498.6	13.92
mpeg2dec	25.10	29.89	6.21	11.02	6.42	24.30	1.20	101.4	30.99	9.24	21.58

Table 3.4: Overall energy savings with respect to benchmark suite.

Benchmark Suite	Max Tot (%)	Avg. Tot (%)	Avg. Sw (%)	Avg. Lk (%)
ISCAS-85	62.64	29.70	30.58	26.74
ISCAS-89	53.20	28.33	29.99	26.96
ITC-99	58.83	29.23	30.90	24.15
Arith	57.19	30.48	33.23	33.15

CHAPTER 4

A Temperature-aware Approach for Simultaneous Delay and Leakage Optimization

Accurate thermal knowledge is essential for achieving ultra low power in deep submicron CMOS technology, as it affects gate speed linearly and leakage exponentially. In this chapter, we propose a temperature-aware leakage minimization technique that efficiently utilizes input vector control (IVC), dual-threshold voltage gate sizing (GS), pin reordering (PR), and gate replacement (GR). To the best of our knowledge, we are the first to consider these techniques simultaneously in a synergistic fashion with thermal knowledge. We evaluate our approach by showing improvements over each method when considered in isolation and in conjunction. Additionally, we study the impact of employing considered techniques with/without accurate thermal knowledge. We ran simulations on synthesized ISCAS-85/89, and ITC-99 circuits on a 45 nm cell library while conforming to industrial design flow. Overall results show leakage power improvements of up to 5.62X (2.26X avg.) when applying thermal knowledge over equivalent methods that do not.

4.1 Introduction

Power minimization continues to be one of the top design metrics in modern VLSI design [33][157][158]. For modern CMOS transistors, power has been primarily characterized into three main sources: 1) switching, due to the charg-

ing/discharging of load capacitance's; 2) short circuit, due to the momentary short circuit state between the pull-up/down of devices; and 3) leakage, which is further broken down into gate tunneling and sub-threshold leakage currents.

Sub-threshold leakage has been shown to be the dominant leakage portion for modern CMOS devices; it is strong function of input vector state and is exponentially affected by operating temperature. Gate delay is also thermally affected as rising temperatures contribute to decreased carrier mobility affecting propagation delays. However, current tools lack early and thermal analysis to better address modern and pending design issues affecting total power and energy consumption, circuit performance, and reliability. Conventional design tools utilize nominal gate switching activity, average gate leakage, and uniform operating temperatures. Optimization under these assumptions not only limits the effectiveness of proposed techniques, but also increases the likelihood of producing designs that violate intended design targets, such as power, area, and delay, due to the lack of thermal knowledge. Yang et al., reports around 70°C in operating processor temperature variation for different workloads [112]. Furthermore, it has been shown by Kumar et al., that temperature can degrade circuit delay by 57% [111]. To account for large differences, designers rely on to enforcing higher guard bands to account for corner cases.

We propose a temperature-aware synthesis methodology combining and improving gate-level pre-silicon synthesis techniques that utilize thermal knowledge during the optimization. A summary of the considered techniques are listed below.

- Dual-V_t gate sizing (GS) – utilize thermal knowledge to efficiently size and assign V_t to thermally impacting gates to minimize leakage power
- Input vector control (IVC) – find promising input vectors for a given thermal map by placing temperature critical gates to their minimal leakage states
- Pin reordering (PR) – improve the effectiveness of IVC by placing each gate

to its optimal leakage state, relaxing IVC-imposed constraints.

- Gate replacement (GR) – replace leakage critical gates with an equivalent lower leakage gates in the library and enhanced when combined PR.

The main contribution of our work is to the demonstrate the vital role temperature knowledge has on modern CAD optimization techniques using industrial imposed constraints. Until now, previous approaches have considered at most two of the mentioned techniques simultaneously with temperature (e.g., GS+PR, IVC+GR). Furthermore, these techniques often assume simplistic delay/power models that operate under nominal conditions (e.g., operating temperature, average leakage). Our goal is to show that a strong interdependence exists when considering all of the enabled techniques in light of utilizing the correct temperature knowledge during the optimization process. We demonstrate that the success of considered techniques heavily depend on each other due to interacting metrics we consider such as leakage and delay. Other contributions include further enhancements to input vector control and gate replacement by simultaneously employing pin reordering using thermal profile knowledge, while adhering to industrial imposed design constraints, such as load capacitance and slew limits [96]. The complete flow can be performed in an iterative fashion to obtain better solutions and can be easily integrated into modern CAD flow. We also show that input vector control optimization should be considered across diverse temperature profile scenarios requiring different input vectors per temperature assumption. Section 4.5 provides a more complete description of our contributions.

We evaluate our approach on a comprehensive set of combinational and sequential benchmarks included in ISCAS-85, ISCAS-89, and ITC-99. We also provide improvement with respect to each technique in isolation, in conjunction with others, and with/without thermal knowledge. Our results show dramatic leakage improvements over state of the art techniques using our novel enhancements to

GS, PR, GR, and IVC, with/without temperature knowledge, demonstrating the additional benefits of incorporating a coordinated thermal-aware optimization.

4.2 Motivation

This section highlights the advantages and limitations of several well-known high-level synthesis techniques that we consider in our work for minimizing leakage power.

Table 4.1 provides average gate propagation delay (t_p) and leakage profiles for each input vector state m for both 2-and 3-input *nand* gate under two operating temperatures (room temp. 25°C and hot 125°C), and at high (HV_t) and low (LV_t) threshold voltage, V_t, settings on a 45 nm cell library. The large differences in leakage power are clearly evident between the minimum leakage state (mls) and respective worst leakage state (wls), a factor of 10.4X at 25°C under HV_t. The differences are even greater when considering a dual-V_t design and temperature (e.g., 338X at 25° at HV_t vs 125°C at LV_t).

The significant leakage range between mls to wls, motivates the use input vector control techniques for minimizing a the leakage profile during idle periods. However, its improvements are limited due to difficulty of controlling internal nodes for circuits with long circuit depths and the requirement of potentially considering exponential number of input vector combinations. Gate replacement techniques and other internal control mechanisms have been explored to address this issue at the cost of area and delay. Figure 4.1 (a), shows an example *nand2* gate with an input vector $\vec{x}$ = "01" (24.8 nW). A standard gate replacement would use a *nand3* by adding a sleep signal $\bar{s}$ to place the gate in "001" (18.2 nW) state, improving standby leakage by 1.36X. Additionally, the overhead of using a larger gate and additional wires alters the circuit's structure (e.g.. interconnect capacitance's), impacting circuit delay and switching power as well. A less intrusive

approach is to optimally order the input pins to achieve its minimal leakage state. In the previous example, the pins can be reordered to achieve "10" (4.09 nW), achieving 6X leakage savings without applying gate replacement.

The addition of temperature drastically changes the optimization search space. Figure 4.1 (b) illustrates how temperature impacts IVC decisions under a hot circuit condition. Under the nominal temperature condition (left), the optimal configuration places the top two nand gates to their mls, while trading off the wls for the remaining output gate. However, making a decision to the right figure would result in a significant leakage penalty of over 6.8X. Thus, under a typical system that exhibits temperature fluctuations, it is key that the correct IVC is used as well as account for the delay alterations. The same idea can be readily applied early in the synthesis phase during gate sizing and V_t assignment. Under the scenario where standby leakage is dominant, the correct input vector plays a vital role in selecting optimal sizes and threshold voltages.

Pin reordering may also be leveraged to reallocate slack (T_{target}-T_{max}) to achieve delay targets, without incurring the potential overhead area from gate sizing and gate replacement, essential for performing simultaneous delay and power optimization when using accurate timing models. For example, replacing a 2-input nand gate with a 3-input incurs an area overhead from 0.80μm^2 to 1.01μm^2. Figure 4.1 (c-d) illustrates an example of maximizing slack by reordering the input pins. Four terms are shown for each net. The top two represent (rise arrival, rise slack), the bottom corresponds to (fall arrival, fall slack) in ps. For the sake of simplicity, ignore the effects of slew on input pins a and b for gate $N2$. Also assume that all gates are operating under nominal temperatures, are negative unate, and that a T_{target}=200 ps is set. Gate $N2$ has cell rise and fall delays of (25, 33) from $a \rightarrow o$ and (66, 84) from $b \rightarrow o$. Although, path $a \rightarrow o$ is shorter than $b \rightarrow o$, the rise and fall arrival times and slack of input b are (79, 46) and (163, -46), which leads to a timing violation (T_{max}=229 ps) (Figure 4.1 (d)). To maximize slack,

Table 4.1: Input vector state (m) dependent leakage power (nW) and average delay propagation (t_p) for high (HV_t) and low (LV_t) of 2-and 3-input nand gates at minimum size operating at room and hot temperature.

	NAND2				NAND3			
	25 °C		125 °C		25 °C		125 °C	
m	HV_t	LV_t	HV_t	LV_t	HV_t	LV_t	HV_t	LV_t
00	3.57	6.22	58.2	176	3.32	5.93	55.9	169
01	24.8	44.8	285	867	18.2	32.8	222	673
10	4.09	7.32	66.3	201	4.21	7.54	67.8	205
11	37.2	67.5	397	1205	39.49	71.6	415	1260
100	-	-	-	-	1.29	2.29	26.0	79.0
101	-	-	-	-	18.8	33.9	228	692
110	-	-	-	-	3.76	6.73	61.9	187
111	-	-	-	-	55.8	101	550	1670
t_p	0.33	0.29	0.38	0.33	0.35	0.31	0.39	0.33

the input pins of $N2$ can be reordered such that the path with minimum slack is connected to the path with the smallest arrival time. Thus, after slack reallocation, T_{max}=194 ps without violations (Figure 4.1 (c)). However, it is important to note that due to the temperature dependence on gate delay, the delay of certain paths may be totally different 4.1, thus, accurate temperature knowledge should be applied during synthesis flows.

4.3 Related Work

Leakage power has become increasingly important for modern CMOS devices. Input vector control (IVC), proposed in [102][103], has been applied to minimize leakage by applying minimum leakage input vectors to leakage critical gates. Finding the minimum input vector is also NP-Hard [121]. Leakage power is reduced

due to the strong dependence of sub-threshold currents (transistor stacking) with respect to a gates applied. The use of MUXes was explored in [99] to drive combinational circuit sections to their minimally achievable leakage states during idle periods. However, IVC improvements are limited for circuits with large depths due to the lack of internal node controllability. Several works cover gate replacement, which replaces gates with worst-case leakage (wls) states with an equivalent lower leakage gate [105][121]. Gate replacement has also been combined with V_t assignment in [110][107]. First proposed in [108], input pin reordering has also been applied in the context of FPGAs [109]. None of these works consider all IVC, GR, and PR together. In addition, temperature dependencies are also ignored.

Gate sizing have become effective techniques for addressing energy and performance metrics [58][115]. The gate width is scaled up/down achieving various drive strengths to enable circuit power and timing optimization. In the discrete domain, gate sizing is known as an NP-Hard problem [97] and several well known solutions have been proposed, such as Lagrangian relaxation [60], dynamic programming [115], combinatorial relaxation [98], and sensitivity-based optimizations [58][59]. Gate sizing under continuous sizing assumptions has also been proposed, and convex programming has been used [116]. Dual threshold voltage (V_t) combined with gate sizing has also been proposed [106][110][119][120]. Low V_t gates achieve greater speed, but at the expense of higher leakage and vice-versa for high V_t gates. Typically, low V_t cells are placed on critical paths to achieve performance and high V_t gates on non critical paths to minimize leakage power. Temperature-aware dual V_t and gate sizing was explored [104] and uses heuristics to place high V_t in hot regions; however, they do not consider the impact of IVC in their leakage models nor accurate cell timing (rise/fall) and load constraints.

More recently, the problem of gate sizing combined with multi-threshold optimization has spurred the interest of Intel researchers to hold a yearly design contest at ISPD on the topic [96]. One motivation was to introduce realistic

optimization design constraints such as gate load, and slew dependencies to the public. The contest objective function was to minimize leakage under specified timing constraints. A limitation of the contest, however, is that only average leakage values were used for each cell. In our work, we consider a gates input vector state to reference its leakage power consumption with respect to its operating temperature. As done in the ISPD-2012 design contest, we conform to identical industrial design constraints (capacitive load and slew limits).

4.4 Power and Delay Model

We employ a table lookup-based power and delay model from an open source 45 nm gate library [118]. In lieu of addressing realistic design challenges faced in industry [96], our model accounts for a gates load capacitance limits, and the rise and fall propagation and slew delays, modeled as function of a gate's driving load capacitance, and the input pin slew. The worst-case gate delay/slew propagation are propagated to the next stage and a negative unateness is assumed throughout the entire cell library. Leakage power is retrieved in a similar table-lookup fashion where its respective gate type/size and its corresponding input vector control state are used as its index. Since our library of choice is assumed to be operate at a single nominal room temperature with all cells at LV_t configuration, we utilize Markovic's model [154] to modify the lookup-procedure to account for both operating temperature and threshold-voltage setting (Section 4.5). We direct the reader to [96] for further information on table-lookup cell-based power and delay modeling.

4.5 Technical Approach

Our leakage minimization framework consists of three major steps, as shown in Figure 4.2, which include: 1) initialization of cell library and circuit thermal maps; 2) leakage minimization through finding minimal input vectors combined with pin ordering ($IVC+PR$); 3) leakage minimization through simultaneous dual-V_t gate sizing (GS), input pin reordering (PR), and gate replacement (GR). Steps 2-3 can be repeated to obtain additional savings. The following subsections cover each enabled technique in greater detail.

4.5.1 Thermal Map

Our work addresses circuit optimization during the pre-silicon phase. Thus, our method relies on circuit thermal simulations to generate thermal maps. Thermal maps can be generated using models found in *HotSpot* [113] where the functional-unit level temperature modeling can be extrapolated to support gate-level temperature modeling. Thus, actual gate-level activity switching factors (obtained either through gate-level or probabilistic simulations), and its cell physical placement information can be used to generate power densities. The resultant power densities can then be used to generate a circuit-wide thermal model. However, such a process would require actual correlations to actual hardware measurements in order to generate a reliable temperature profile for use [113]. Due to this limitation in our optimization search space, we assume a static chip-wide temperature profile for each netlist as a starting point. As temperature modeling in CAD tools mature, further improvements in designs may be possible, since input vector control, pin-reordering, gate replacement, and gate-sizing all impact power dissipation of the circuit (Sections 4.2 and 4.4). For our experiments, we generate temperature scenarios under two circuit-wide operating temperature assumptions (55°C as *cool* and 125°C as *hot*).

4.5.2 Input Vector Control and Pin Reordering

IVC is an essential technique for leakage reduction in idle modes since temperature critical (e.g., hot) gates may be placed in lower leakage states and traded-off with less critical (e.g., cool) gates be be placed in higher leakage states. We improve the conventional IVC by combining it with input pin reordering (PR). PR provides additional opportunity for leakage savings for each gate, since it relaxes the constraint imposed by the IVC setting of its transitive fan-in gates. Thus, a higher percentage of gates may be placed at or closer to their respective mls, making IVC+PR an effective technique for addressing circuits that exhibit large temperature variations.

Finding the optimal IVC is NP-Hard [100]. In our work, we utilize a statistical random-walk procedure of 10K randomly generated input vectors for obtaining a promising IVC to be used in later phases. This procedure is simple in nature and note that more sophisticated techniques can be used. However, our experience has shown this technique achieves relatively fast convergence with most designs converging before completion. To further reduce the number of computations in this phase, we modify the lookup-table entry for each gate to only consider the minimum leakage values for respective input vector permutations. For example, the minimum IVC for a 3-input nandgate "100" can represent its respective leakage profiles of "100" and "010" (Table 4.1).

4.5.3 Gate Configuration Selection

The next step in our flow is to perform iterative gate-level modifications for minimizing leakage power with respect to a given delay target. This phase combines gate replacement (GR), dual-V_t gate sizing (GS), and pin reordering (PR) techniques.

We introduce GR during this optimization phase, to maximize leakage savings

Algorithm 1 Gate Configuration Selection Algorithm

1: Initialize library, delay target D_{target}, and *netlist*
2: Obtain minimum input vector and pin reordering ($mIVC$)
3: Compute difficulty metric for each gate (power, delay)
4: Initially set all gates to High-V_t and minimal size
5: Compute difficulty and lock top K critical gates
6: Lock K maximally constrained gates G_{lock}
7: **repeat**
8: For each gate, find least constraining move m
9: Perform move m and lock chosen gate
10: If all gates are locked, unlock all gates $\in G_{lock}$
11: If no solutions found after L iterations then relax K
12: Recompute circuit difficulties
13: **until** $Converge$

of a given circuit configuration. Our gate replacement policy is similar in nature to [121], but is enhanced such that we combine it with input pin reordering. Our GR procedure replaces a gate's G and its input vector $\vec{x}$, with an available lower leakage gate $\tilde{G}$, with reordered pins $\tilde{\vec{x}}$ and sleep signal $\bar{s}$. Additionally, $\bar{s} \in \tilde{\vec{x}}$ and can be set to a value $\in \{0,1\}$. The resultant circuit after performing GR is still functionally equivalent. However, alterations of a gate type may result with very different power and timing characteristics of the entire circuit, potentially constraining or relaxing the optimization search space. Thus, we repeat the IVC+PR procedure in order to obtain more promising input vectors in future iterations.

With the addition of GR+PR during this phase, The number of available *moves* for each gate can be quite large. Thus, we restrict GR to be enabled only for nand gates. PR is performed simultaneously when deciding a gates size and threshold voltage and is enabled only for *epsilon* critical gates (ϵ-gates). Only

delay critical gates are enabled as PR candidates since this phase emphasis is on delay optimization, as most gates should have been placed in their minimal mls during the $IVC{+}PR$ phase. Thus, a gate configuration is comprised of its size, V_t, GR, and PR. Gate configuration selection is also permed while while conforming to the special restrictions mentioned above.

4.5.3.1 Selection Heuristic

Our gate configuration selection approach is based on *maximally* constrained, *minimally* constraining optimization paradigm. The procedure is constructive at each its step such that the most benefiting move in terms optimization criteria is performed. The maximally constrained principle attempts to assign the gate configurations to difficult gates early while there is still flexibility or slack in the design. In addition, the early assignment of the difficult or constraining gates early provide an accurate picture of actual consequent difficulties for future moves and is recognized as early as possible [86]. In a circuit where leakage power is localized to few hot spots, determining the optimal configuration of these gates early in the optimization phase is critical. The *minimally* constraining principle states that at each step we should determine a gates move (GS, GR, and PR) in such a way that the remaining gates are as least constrained as possible.

We first define the sources of difficulty or constraining metric in determining the best configuration for a particular gate. Gates are sorted in descending order during "difficulty computation" step (Algorithm 1) in decreasing precedence listed below:

1. Leakage power - temperature leakage impact factor
2. Slack - gate participation on the critical paths
3. Logic Depth - gate participation on longest paths

4. Fan-out - affect on its transitive fan-out gates

5. Fan-in - affect on its transitive fan-in gates

As shown above, the leakage profile of a gate is considered as the main source of difficulty, followed by its ability to potentially impact the circuit delay. Our main objective is to minimize leakage consumption, thus, we first identify the top K leakage critical gates and lock them to their minimally impacting configuration.

Algorithm 1 highlights our gate configuration selection procedure. Line 1 and 2 performs all required pre-processing steps including thermal simulation to identify critically temperature impacting gates and IVC to obtain the minimum leakage input vector combined with its corresponding optimal pin-reordering structure. The key idea in this step is to maximize achievable savings by $IVC + PR$ before performing gate-level adjustments. In other words, an accurate picture of the circuits leakage profile is determined early, such that any subsequent optimization (e.g., iterative refinement) lead to a better solution. Line 3 initializes the gate sizing framework such that the most constrained or high leakage impacting gates are locked in initial minimal leakage configuration in order to minimize overall leakage power. For our purposes, we set K to be equal to the number of gates predicted to be temperature critical. Lines 7-13 performs iterative gate-level adjustments based on its move classification (next subsection). This procedure is repeated until all gates configuration have been set. Note that once a configuration is determined for a gate, the gate is locked (frozen) (excluding gates in initially the locked gate set G_{lock}) until the start of the next iteration have been determined. The locking principle prevents the algorithm from getting stuck into a local minima by requiring all gates configurations to be determined before a subsequent phase is performed.

4.5.3.2 Move Classification

Gate moves (GS, GR, and PR) are classified into three groups with respect to our delay-constrained objective. For each gate potential gate move, three ideal scenarios are considered:

1. Leakage power and delay reduction
2. Leakage power reduction, constant delay
3. Leakage power reduction and delay increase

It is important to account for valid moves (no load or slew violation). These valid moves are assigned priorities in the precedence class order of i, ii, and iii. Moves that benefit both leakage and delay (class i) are always selected over moves belonging in classes ii and iii, and are compared against other moves within its own class as the product of leakage and delay savings. If no class i moves exists, then class ii moves are selected by the maximum leakage energy improvement. If only class iii moves are found, the move that produced the maximum $\frac{benefit}{cost}$ is selected. Note that the above objective concepts may be applied inversely when the objective is set (e.g., power-constrained delay minimization).

One challenge during our gate-level configuration selection approach is that the initial step requires the locking of K top critical gates (line 5). Note that a condition may exist where the target delay was not achievable due to locking constraints placed on sizable gates. Under these cases, where after a specified number iterations have passed where no valid solution has been found, a gate is chosen in the locked set (G_{lock}) to be unlocked using the maximally constrained and minimally constraining principle. The most constrained locked gate is defined using: 1) maximum frequency being on the critical path, and 2) leakage power. These frequency statistics may be recorded when performing accurate delay computation.

4.5.3.3 Epsilon Critical Tree Extraction

Another major difficulty encountered in modern gate sizing flows is the ability to scale to larger circuits. To address this issue, we employ an epsilon critical tree structure to enable our algorithm to scale linearly with respect to circuit size. The requirement of comparing and determining gate configurations while maintaining accurate delay pictures, is the major challenge faced in sensitivity-based algorithms. A single move may require delay re-computation of the entire circuit. Thus, performing per-gate-wise delay update results in quadratic run time. We develop an efficient epsilon tree structure that performs delay updates when it is detected that gates along the critical path (or within some tolerance) are updated. The cost for acquiring accurate delay values of the entire circuit is significantly reduced, while minimally impacting accuracy. To further improve run-times, groups of gates may be sized at a time as done in [58].

An epsilon tree (ϵ_{path}) provided in Figure 3.4a and consists of gates that were within $\epsilon - delay$ of the critical path during the last accurate delay computation (shaded nodes in Figure 3.4a). The bold-outlined nodes are primary outputs (P_{out}) transitively connected to at least one node belonging to the critical path. The delay impact of a gate is accounted by its transitive relation to the ϵ_{path}. For example, a gate that is either on the critical path or an output gate of a critical path gate would cause a δ delay with respect to all the primary output nodes in the transitive fan-out of the critical gate. The δ is used to estimate the delay impact of each move via a delay cost formula, defined as the sum of the squared difference with respect to each transitively related primary output's output time, ϵ_i, with respect to the delay target (Figure 3.4a). Using an ϵ_{path} enables the following very efficient delay estimation:

$$D_{cost} = \sum_{i}^{|P_{out}|} (\epsilon_i - D_{target})^2 \tag{4.1}$$

Significant reduction in run time can be achieved since the percentage of gates

that make up the critical path is relatively small compared to the total gate count ($\leq$ 5%). However, there can be an exponential number of paths that need to be taken into account. Thus, in order to maintain reasonable accuracy, we update the propagation and slew delays (rise/fall) after assigning a gates configuration of the current gate under inspection, as well as its immediate fan-in/out connections. We also note that circuit violations (load and slew) are also fixed during accurate delay computations. We employ a similar technique as in [59] for fixing violations. Figure 3.4b provides the run-times achieved of our method with respect to circuit size.

4.6 Simulation Framework

We evaluate all considered leakage minimization synthesis techniques on 14 industrial benchmarks included in ISCAS-85/89, and ITC-99, and were synthesized using Cadence Encounter in order to retrieve net/wire capacitances. The Nandgate 45 nm cell library [118] is set as our base library and 3 gate sizes are assumed (1X, 2X, and 4X). We extend the cell library to support dual-V_t optimization by using EKV formulas in [154] to fit against the base library and set (LV_t=0.55V, HV_t=0.6V). For generate power and time results, we implement an in-house power and delay timer C++ correlate within 1E-3 within the Synopsys PrimeTime industrial tool. We extend our lookup table model to support continuous temperature indexing so that both leakage power and delay can be obtained using the gates driving load capacitance, size/type, and rise/fall input slews, and temperature. Leakage power is indexed using its input vector state, which is dependent on its applied IVC. The minimum leakage IVC is obtained through simulation (Section 4.5). We assume two chip-wide thermal operating settings: 1) 55°C as *cool* and 2) 125°C as *hot*. We limit the gate configuration selection procedure up to 3 iterative refinement phases and report their results in the next section; additional

Table 4.2: Leakage power improvement factors for ISCAS-85 circuits when optimized under actual (Act.) and predicted (Pred.) temperatures with respect (row-wise) to the addition of each enabled technique and (column-wise): (O1) IVC+GS; (O2) IVC+GS+PR; and (O3) IVC+GS+PR+GR.

Circ.	Act.	Pred.	Lk. Pwr. (uW)			Lk. Imprv. (X)			% of gates. (mls, wls)			Clk
	(°C)	(°C)	O1	O2	O3	O1	O2	O3	O1	O2	O3	(ns)
c1355	Nm.	Nm.	43.4	28.3	9.79	1.10	1.28	1.42	-	-	-	17.0
		Hot	47.8	36.2	13.9	-	-	-	-	-	-	
	Hot	Nm.	69.6	45.0	28.6	1.45	1.70	1.66	17, 17	17, 17	67, 0	19.4
		Hot	48.1	26.4	17.3	-	-	-	11, 8	11, 8	75, 5	
c2670	Nm.	Nm.	95.1	88.8	72.5	1.23	1.15	1.01	-	-	-	15.7
		Hot	117	102	73.6	-	-	-	-	-	-	
	Hot	Nm.	522	499	412	3.70	3.99	4.08	24, 25	26, 21	41, 15	17.1
		Hot	141	125	101	-	-	-	37, 17	41, 11	59, 11	
c3540	Nm.	Nm.	135	124	99.8	1.20	1.18	1.08	-	-	-	28.2
		Hot	162	146	108	-	-	-	-	-	-	
	Hot	Nm.	556	532	492	2.91	3.10	3.58	40, 31	47, 27	66, 17	28.3
		Hot	191	172	138	-	-	-	41, 23	50, 17	71, 14	
c7552	Nm.	Nm.	155	152	136	2.03	1.85	1.58	-	-	-	24.8
		Hot	315	281	216	-	-	-	-	-	-	
	Hot	Nm.	734	722	644	1.92	2.13	2.35	33, 27	39, 25	58, 18	25.3
		Hot	382	339	274	-	-	-	38, 23	41, 17	68, 15	

phases resulted with marginal improvements with respect to simulation runtime.

4.7 Experimental Results

We evaluate the effect of utilizing temperature knowledge during the optimization process when considering input vector control (IVC), dual-V_t gate sizing (GS), input pin reordering (PR), and gate replacement (GR). We report the leakage improvements across three enabled optimization modes corresponding to their enabled techniques: 1) O1 (IVC+GS), 2) O2 (IVC+GS+PR); and 3) O3 (IVC+GS+PR+GR). The optimization objective is to minimize leakage consumption (stand-by mode), while meeting delay targets.

Table 4.2-**??** shows the impact of using temperature knowledge under all considered leakage optimization techniques. Results are grouped (row-wise) with respect to the circuit, and further sub-grouped (row-wise) with respect to the

Table 4.3: Leakage power and improvement factors for ISCAS-89 circuits (row-wise) shows results when optimizing under actual (Act.) and predicted (Pred.) temperatures with respect to the addition of each enabled technique (column-wise).

Circ.	Act.	Pred.	Lk. Pwr. (uW)			Lk. Imprv. (X)			% of gates. (mls, wls)			Clk
	(°C)	(°C)	O1	O2	O3	O1	O2	O3	O1	O2	O3	(ns)
s820	Nm.	Nm.	8.82	7.78	7.75	1.30	1.33	1.24	-	-	-	9.91
		Hot	11.4	10.3	9.91	-	-	-	-	-	-	
	Hot	Nm.	57.3	53.3	53.2	2.33	2.37	2.39	23, 11	35, 11	56, 3	9.91
		Hot	24.6	22.5	22.2	-	-	-	23, 6	41, 3	56, 3	
s838	Nm.	Nm.	11.2	10.9	9.93	1.58	1.59	1.62	-	-	-	47.1
		Hot	17.8	17.4	16.1	-	-	-	-	-	-	
	Hot	Nm.	86.6	81.3	75.8	2.70	2.62	2.69	36, 24	38, 18	48, 8	47.1
		Hot	32.0	31.0	28.2	-	-	-	42, 6	50, 6	64, 4	
s953	Nm.	Nm.	8.55	8.10	7.43	2.09	2.06	2.14	-	-	-	19.5
		Hot	17.9	16.7	15.9	-	-	-	-	-	-	
	Hot	Nm.	50.5	48.4	43.5	1.68	1.71	1.66	24, 30	32, 26	60, 6	20.4
		Hot	30.1	28.2	26.2	-	-	-	38, 10	46, 10	64, 4	
s1488	Nm.	Nm.	45.6	43.2	39.4	1.04	1.02	1.01	-	-	-	7.44
		Hot	47.3	43.9	40.0	-	-	-	-	-	-	
	Hot	Nm.	209	194	184	2.81	2.79	2.87	28, 13	40, 9	78, 4	7.44
		Hot	74.3	69.5	64.2	-	-	-	34, 11	45, 2	84, 1	
s15850	Nm.	Nm.	604	584	531	1.12	1.07	1.01	-	-	-	44.5
		Hot	674	627	539	-	-	-	-	-	-	
	Hot	Nm.	3680	3570	3330	4.05	4.14	4.26	40, 30	41, 30	53, 25	45.1
		Hot	908	863	780	-	-	-	61, 23	63, 22	80, 18	

correct and wrong temperature assumption (columns 4 and 5), respectively. The correct temperature knowledge is used when the *actual* temperature under "Act." matches the *predicted* temperature "Pred." For example, consider benchmark $c2670$ where a *hot* temperature scenario is considered. The leakage power achieved when making the correct "Hot" temperature knowledge is 141 uW in contrast to using the wrong temperature knowledge that resulted with a leakage power of 522 uW. Subsequent leakage improvements are provided for the remaining techniques enabled under O2 and O3. For benchmark $c2670$, using temperature knowledge achieved leakage improvements (leakage reduction factor) 3.70X (O1), 3.99X (O2), and 4.08X (O3). As expected, additional leakage reductions were achieved as more techniques were enabled.

The impact of temperature knowledge in placing gates in their minimal leakage

Table 4.4: Leakage power and improvement factors for ITC-99 circuits (row-wise) shows results when optimizing under actual (Act.) and predicted (Pred.) temperatures with respect to the addition of each enabled technique (column-wise).

Circ.	Act.	Pred.	Lk. Pwr. (uW)			Lk. Imprv. (X)			% of gates. (mls, wls)			Clk
	(°C)	(°C)	O1	O2	O3	O1	O2	O3	O1	O2	O3	(ns)
b11	Nm.	Nm.	15.7	13.9	9.16	2.18	2.01	1.55	-	-	-	22.3
		Hot	34.2	27.9	14.2	-	-	-	-	-	-	
	Hot	Nm.	114	96.6	60.8	2.08	2.20	2.78	15, 25	17, 23	56, 6	22.3
		Hot	54.6	43.9	21.9	-	-	-	18, 14	20, 14	60, 2	
b12	Nm.	Nm.	19.7	17.3	11.6	2.01	1.86	1.71	-	-	-	17.5
		Hot	39.6	32.2	19.7	-	-	-	-	-	-	
	Hot	Nm.	129	115	77.9	2.13	2.37	2.30	30, 22	30, 22	51, 11	17.5
		Hot	60.5	48.3	33.9	-	-	-	30, 8	32, 7	64, 4	
b13	Nm.	Nm.	2.00	1.78	0.977	4.05	4.54	5.62	-	-	-	6.82
		Hot	9.10	7.22	5.49	-	-	-	-	-	-	
	Hot	Nm.	17.2	15.0	10.9	1.08	1.11	1.10	13, 31	18, 31	36, 13	6.82
		Hot	15.9	13.5	9.90	-	-	-	26, 8	34, 8	60, 5	
b17	Nm.	Nm.	191	171	139	1.41	1.16	1.13	-	-	-	40.6
		Hot	269	198	157	-	-	-	-	-	-	
	Hot	Nm.	1230	1110	864	2.43	2.84	3.20	25, 27	27, 26	55, 9	40.7
		Hot	506	390	270	-	-	-	28, 21	32, 20	62, 8	
b18	Nm.	Nm.	235	213	158	1.72	1.61	1.57	-	-	-	43.6
		Hot	404	343	249	-	-	-	-	-	-	
	Hot	Nm.	1350	1260	917	2.54	2.79	2.67	35, 22	39, 21	60, 8	49.9
		Hot	530	449	343	-	-	-	38, 13	41, 12	65, 5	

state can be clearly observed by determining the % of gates (post optimization) in their minimum leakage state (mls) and worst leakage state (wls), listed under the "% Gates." columns. Due to their exponential dependency on temperature, only *hot* operating conditions are listed (col. 12-14). It is important to note that wls and mls are only two of the 2^{fi} leakage states considered, where fi corresponds to gates fan-in size.

Using accurate temperature knowledge enables superior solutions over an equivalent methods using the incorrect temperature assumption (Table 4.2-4.4). For example, the result for "c2670" shows that the correct temperature knowledge enabled 37% gates to be placed in mls in contrast to 24% when the wrong temperature knowledge is used. Additionally, using the correct temperature knowledge placed a lower percentages of gates in their wls (17% vs 25%). As the number

Table 4.5: Overall leakage power savings for each enabled optimization (O1, O2, and O3) for each benchmark suite under nominal and hot operating conditions.

Circ. Suite.	Temp.	O1	O2	O3
ISCAS-85	Nm.	1.65X	1.63X	1.56X
	Hot	2.28X	2.47X	2.62X
ISCAS-89	Nm.	1.43X	1.41X	1.41X
	Hot	2.71X	2.73X	2.80X
ITC-99	Nm.	2.37X	2.14X	2.31X
	Hot	2.05X	2.27X	2.41X
Avg. Nm.		1.82X	1.73X	1.76X
Avg. Hot		2.35X	2.49X	2.61X
Max Nm.		4.54X	4.05X	5.62X
Max Hot		4.05X	4.14X	4.26X

of available gate configurations increases (from O1 to O3), more gates were able to be placed in their mls, and less in their wls. Improvements using the correct temperature knowledge are greater since they enable more gate selection candidates to be selected among temperature-leakage critical gates, thus, more gates are able to be placed in their mls, and less in wls. For instance, circuit the optimization of "c2670" under the correct temperature prediction (59% mls, 14% wls) outperformed the wrong temperature prediction (41% mls, 15% wls).

Table 4.5 shows overall leakage improvements when using the correct temperature knowledge during optimization with respect to the considered benchmark suites and correct temperature setting. On average, leakage improvements are greater under the hot circuit condition achieving 2.35X, 2.49X, and 2.61X leakage improvement factor over when using the wrong temperature assumption. This is expected since under the *actual* nominal temperature case, all gates are assumed to be operating at cooler temperature. Therefore, the impact of making a

wrong optimization decision (e.g., input vector, gate size, V_t setting) is lower, since all gates operate at nominal operating temperature (55°C). However, under the *hot* (125°) operating scenario, the impact of making the wrong decision is much greater since magnitude between the gate leakage state across available gate configurations, is increased when temperature is accounted. Therefore, knowing the best decisions with considering the gate size, minimum input vector, and threshold voltage assignment, becomes more crucial during optimization, especially in light of constrained optimization when a number of ideal moves are limited, and knowing trade-off between slack and leakage power is critical. Additional reductions should be expected under more realistic operating conditions where accurate temperature variations are available.

4.8 Summary

We have developed a synergistic temperature-aware delay and leakage optimization approach using enhanced synthesis techniques including: input vector control (IVC), pin reordering (PR), dual-threshold voltage V_t gate sizing (GS), and gate replacement (GR). We study the impact of temperature knowledge using these techniques under two assumed operating temperatures (*cool* and *hot*) and report significant leakage improvements averaging 2.61X (4.26X max) when utilizing the correct temperature knowledge. We evaluate our approach on a comprehensive set of benchmarks included in ISCAS-85/89, and ITC-99 on a 45 nm dual-V_{TH} cell library while conforming to industrial imposed constraints.

As future work, we aim to study the impact of using accurate thermal knowledge under more accurate operating conditions. We believe additional improvements can be achieved due to ability to leverage our considered techniques to specific *hot* and *cool* regions on a chip. The consideration of duty cycle also plays an important role, since total power and energy is both a function of switching

and leakage power and energy components. Therefore, identifying promising input vectors to minimize stand-by leakage, while also minimizing switching and leakage during active move, is required.

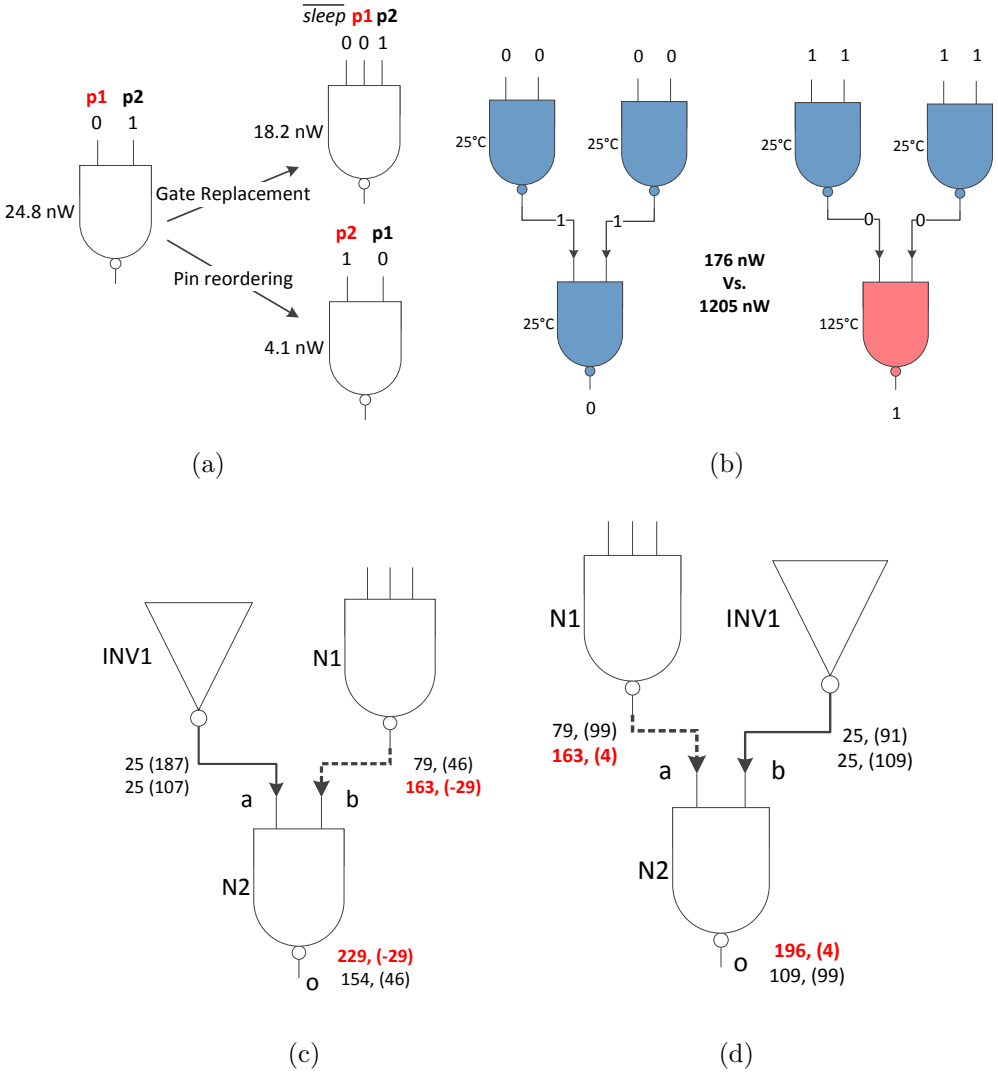


Figure 4.1: Pin reordering vs gate replacement (a); input vector control under two temperature settings (b); pre-slack (c) and post-slack (d) re-allocation via input pin reordering. For (c-d), the critical path is represented as bold-red with arrival time (left) and slack (right) terms provided.

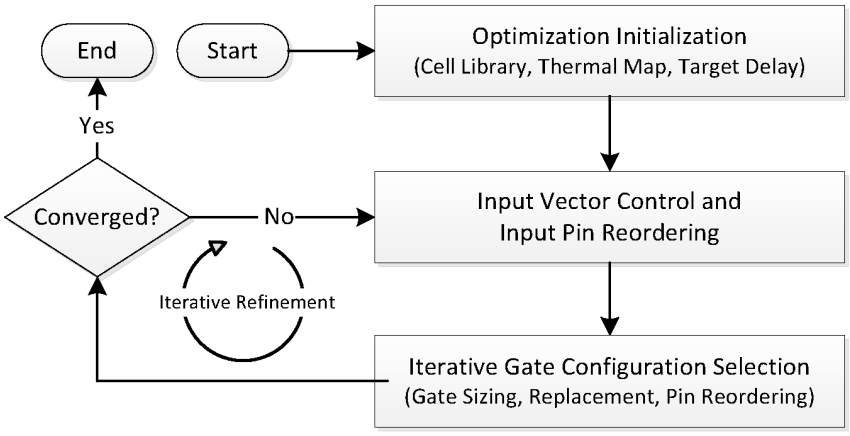


Figure 4.2: Temperature-aware synthesis flow

CHAPTER 5

Maximizing Yield in the Near-Threshold Regime in the Presence of Process Variation

Near-Threshold Computing (NTC) shows potential to provide significant energy efficiency improvements as it alleviates the impact of leakage in modern deep submicron CMOS technology. As the gap between supply and threshold voltage shrink, however, the energy efficiency gains come at the cost of device performance variability. Thus, adopting near-threshold in modern CAD flows requires careful consideration when addressing commonly targeted objectives. In this chapter, we propose a process variation-aware near-threshold voltage (PV-N_{vt}) gate sizing framework for minimizing power subject to performance yield constraints. We evaluate our approach using an industrial-flow on a set of modern benchmarks. Our results show our method achieves significant improvement in leakage power, while meeting performance yield targets, over a state-of-the-art method that does not consider near-threshold computing.

5.1 Introduction

Power and performance continue to be the top design metrics for optimization in modern and pending IC technologies. The near-threshold computing (NTC) paradigm has been shown to provide significant energy efficiency improvements of 10X by scaling supply voltage (V_{DD}) to comparable levels to the nominal threshold voltage (V_{TH}). However, several challenges must be addressed when incorporating

NTC in modern CAD flows, such as their significant performance cost and high susceptibility to performance variations due to the lessening gap between supply and threshold voltages [61]. As devices continue to scale into the deep submicron regime, these issues will be further compounded by process variation.

Process variation (PV) is an unavoidable side product of modern and pending silicon implementation technologies. As a ramification of PV, each transistor, gate, and wire on each integrated circuit that realizes a particular design has unique physical (e.g. channel length) and manifestational (e.g. power and delay) properties [66][67][68]. When considering such implications in variations for a near-threshold voltage (N_{vt}) design where the gap between V_{DD} is reduced, the effects on respective leakage and delay components are magnified.

PV may eliminate most of the potential gains from one technology generation or design optimization [67]. For example, due to the impact of PV, the classic design approaches that aim to optimize delay and power, such as [69], would not produce optimal solutions. Therefore, there is an increasingly pressing need for the design practice to switch from a fixed deterministic domain to a usually infinite probabilistic and statistical domain, in order to reflect the changes brought about by the existence of PV [70].

In this paper, we consider an IC design optimization problem of performing a process variation-aware near-threshold voltage (PV-N_{vt}) gate sizing method. Our goal is to optimize the yield of a given IC design (i.e., to maximize the total number of ICs that meet a certain set of power and delay specifications). We achieve this goal by gate sizing and selecting threshold voltage settings for the gates on the circuit in such a way that the maximum number of ICs can meet the power and delay requirements. In order to reflect the impact of PV, we use a scenario-based approach by creating a set of scenarios (IC samples) that are representative of the PV model.

First, we perform simultaneous delay and power optimization of a circuit to

meet a target performance and power budget. Once a specified threshold is satisfied, the next step is to optimize the yield of the set of scenarios. For this step, we have developed an iterative heuristic algorithm that efficiently identifies the most problematic sections of the circuit (through our partitioning scheme) in terms of the objective function target (e.g., delay and leakage power). The critically determined sections are then made more resilient to PV in order to maximize yield. Each partition is optimized such that the observed maximally benefiting configuration (e.g., size and V_{TH}) is selected, such that the overall yield is improved, while taking into account of the global circuit optimization search space. We validate our approach using statistical re-sampling techniques on various generated scenarios. The procedure is constructive in nature and can be repeated to improve the design at the expense of additional run-time.

5.2 Motivation

We begin by providing a small realistic example demonstrating the advantages and challenges encountered when considering an NTC-enabled design. Consider Figure 5.1a, which shows a small representative circuit composed of eight cells (six inverters, one 2-input nand gate, and one 4-input nand gate). Special considerations must be made when enabling NTC for optimizing the circuit under conflicting objectives, such as delay and power. Figure 5.1b and 5.1c presents delay vs. leakage power (log scale), when performing four separate of individual cell V_{TH} modifications on N1 and INV5, independently. A plot of 1000 generated circuit instances is provided against each V_{TH} circuit configuration with delay variations following a normal PV model.

As shown in Figure 5.1b, the performance (delay) variability is significant when all cells are at a NTC setting, ˜200 ps vs ˜50 ps for non-NTC. However, the gains in leakage power reductions are significant (exponential). Thus, selecting cells

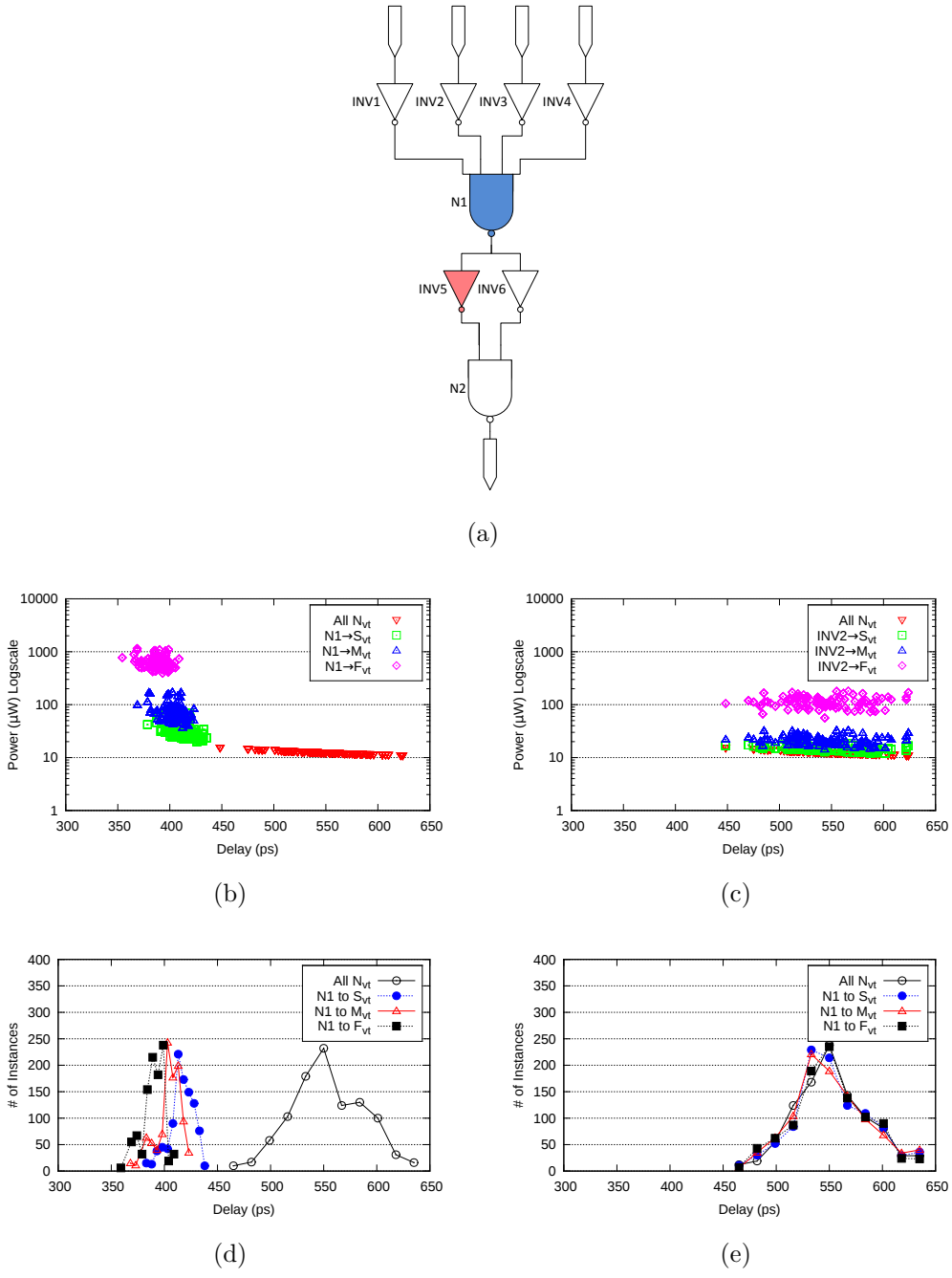


Figure 5.1: Example circuit (a) with inverters (INV1 to INV6) and nand gates (N1 and N2); (b, c) distribution of 1000 circuit instances and achieved delay when decreasing V_t starting from $N_{vt} \rightarrow \{S_{vt}, M_{vt}, F_{vt}\}$, selectively for gate N1 (b) and gate INV5 (c); (d, e) circuit instance distribution vs circuit delays when performing V_{TH} adjustments on gates N1 (d) and INV5 (e).

with lower V_{TH} addresses performance variation in a design, at the cost of exponential leakage power overhead. Selecting the proper cell to be altered, however, requires identifying cells which maximally reduces variation. For instance, altering INV5 provides no additional benefit in addressing the performance variability, as paths that pass through INV6 are still affected by PV (Figure 5.1c). In contrast, altering N1 minimizes the delay variations from the four inputs (INV1 to INV4), and outputs to INV5 and INV6, simultaneously. Therefore, when attempting to maximize yield, it is key to identify cells that participate on many paths in order to suppress performance variation in a design, as shown in Figures 5.1d over 5.1e.

The significant variations that are more susceptible in an NTC-enabled design motivate the requirement of handling other uncertainty factors, such as dealing with spatial correlations. For example, due to an arbitrary model with complex correlations, identifying the cells that maximally suppress variation cannot be modeled through standard statistical approaches. In order to address the issue, a scenario-based approach is utilized to generate instances to optimize based on a given PV model.

5.3 Related Work

5.3.1 Process Variation

Worst case analysis (WCA) is widely used in industry to deal with the impact of PV [71]. WCA considers the worst-case parameter values due to PV, environmental, and aging effects [89][90]. WCA is used in both verification and design processes. In the verification process, WCA is used to verify the target IC against the specification in the worst case, which is helpful in reducing technical risk and improving system reliability. However, in the design process, WCA produces overly conservative designs that over-estimate the impact of PV. Such over-estimation may complicate the design process and, more importantly, result in unnecessary

performance degradation.

In order to solve the WCA issues, researchers have been promoting statistical circuit modeling in IC design and analysis [70]. The goal of statistical modeling is to search for alternatives to WCA that provide more accurate representations of PV. As the demand on performance, power, and density continues to increase in modern IC design, statistical modeling plays an increasingly important role in achieving greater gains compared to worst-case design.

Recent work has focused on the manifestation properties of an IC under the impact of PV. Sarangi et al. propose a timing error model resulting from systematic and random PV effects [67]. In [72], an analysis of the leakage power distribution due to PV is given and used to predict the CDF/PDF of the total chip leakage. Alkabani et al. propose an approach for post-silicon leakage power reduction through input vector control (IVC) that takes into account PV [73]. Wei et al. have developed gate-level characterization techniques for quantifying PV effects in addressing hardware security [91]-[95].

5.3.2 Scenario-based Analysis

Scenario-based approaches [74][75][76] have been used to solve design optimization problems with uncertain constraints, referred to as chance-constrained problems. The main idea is to use a sampling of the constraints to approximate the infinite space of variable constraints caused by the variations. Calafiore et al. prove that the solution of the scenario problem maps approximately to the original problem with uncertain constraints; this work also provides an explicit bound on the number of samples that are needed to obtain a specified levels of robustness [74]. [75] obtains a convex approximation to the chance-constrained problem and extends it to an ambiguous set of constraints, where the random distribution of the variations belongs to a convex compact set instead of a fixed distribution.

5.3.3 Gate Sizing

Gate sizing has been a crucial task for accomplishing simultaneous optimization of delay, power, and area since the very early beginnings of CAD [77]. In the mid-80s, Fishburn and Dunlop proposed a provably optimal approach to transistor sizing [69]. It proposes an optimal gate sizing scheme to meet the delay constraint by using convex programming, but it does not consider the impact of process variation. More recently, there have been several efforts to optimize manifestational characteristics of ICs for a design of interest in the presence of PV using gate sizing. Zhu et al. proposes a gate sizing and clustering approach to optimize leakage energy adaptive body bias, which takes into consideration the process variations in different instances of ICs [78]. [79] discusses a gate-sizing algorithm to minimize the number of failing chips considering process variation. It compares the variation-aware design with the worst-case approaches and confirms the gain obtained from the former. [80] proposes a geometric programming-based heuristic approach to gate sizing. More recently, researchers from Intel have held yearly discrete cell sizing contests to expose the challenges encountered in an industrial flow [96]. However, only leakage power is accounted with no PV model assumed. Additional improvements when accurate operating conditions such as temperature, gate switching, and input vector state leakage computations are accounted for[87][88], however, PV is not considered in their models as well.

Other works use statistics-based approaches to capture the uncertainty stemming from PV [81][82][83]. The spatial correlations in L_{eff} inherent in the PV model, however, are too complex to be captured by simple statistical models. Our scenario-based approach is simple, generic, and flexible; it can be applied to any number of optimization tasks because it does not rely on assumptions about the uncertainty model. Instead, the uncertainty model is used to generate scenarios to be used as a training set for optimization, with the idea that if the training scenarios are representative, then the optimization will work well for any set

of instances. Calafiore and Campi prove that for convex programs the scenario approach provides a solution that satisfies most constraints with high probability given enough samples, establishing a theoretical bound [74]. Therefore, the scenario approach has the same theoretical underpinnings as traditional convex optimization methods. However, our main contribution is to extend the scenario approach to essentially optimize known NP-hard problems (e.g. discrete gate sizing [97]) well, by combining it with generic optimization techniques (e.g. iterative improvement) and statistical analysis. Although a mathematical result can no longer be theoretically proven in this case, a statistical interval of confidence for various sample set sizes can be established.

5.4 Technical Approach

Figure 5.2 highlights the several steps in our NTC-enabled PV-aware yield optimization framework. Each step is discussed in the next following subsections.

5.4.1 Cell Switching Activity

As the gap between V_{TH} and V_{DD} is reduced in NTC (Eq. 2.2), switching power starts to become the dominant power (Eq. 2.3). Therefore, accurate knowledge in determining which cells have high switching activity (SA), is crucial for maximizing energy efficiency for a given design, especially for NTC-enabled designs. As a pre-processing step, we perform gate-level event simulation by applying a set of 100K randomly generated input vectors to the primary inputs of a given circuit and record the SA for each respective net (wire) of the design. Accurate input stimulus from actual applications may be used instead to improve accuracy. Only the SA of nets are recorded for computing switching activity since each gate in our studied benchmarks is driven by only one net. Therefore, the total switching power for a given design can be computed as the sum of all net switching power

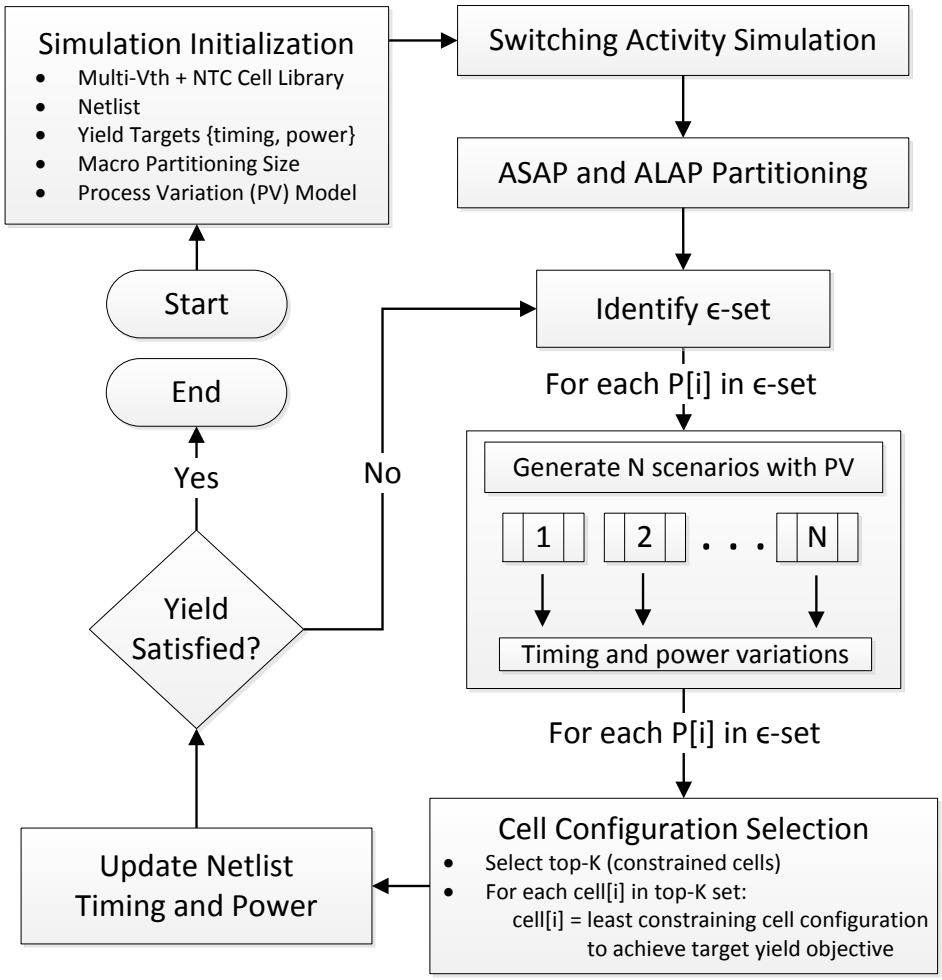


Figure 5.2: Process variation-aware NTC-enabled design flow.

using Eq. 2.3, where the $C_L = C_{net}+C_{pin}$. Here, C_{net} is the interconnect and C_{pin} is the sum of driven outgoing pin capacitance's.

5.4.2 Logic-depth Indexing

We employ an *as-soon-as-possible* (ASAP) and *as-late-as-possible* (ALAP) cell indexing principle for partitioning a circuit into distinct groups. The ASAP index represents the maximum number of logic stages an input signal is required to propagate (primary input source) before it is considered valid at its output pin. The ASAP index for a particular cell is the max ASAP value among its inputs.

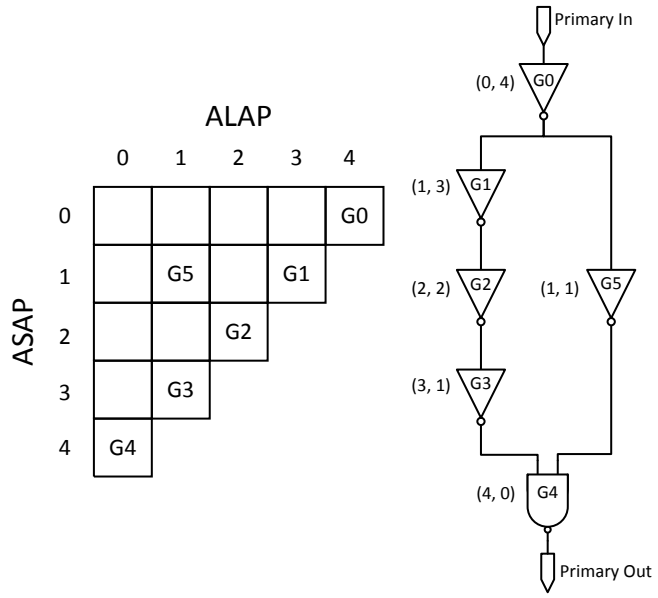


Figure 5.3: *As soon as possible* (ASAP) and *as late as possible* (ALAP) gate logic mapping.

The ALAP index represents the maximum number of stages the output signal of a cell must propagate before reaching the inputs of all its transitive primary outputs. As a result, cells in a given design can simply be indexed (grouped) by its respective ASAP and ALAP index (Figure 5.3).

The ASAP and ALAP index values can be used to provide useful structural knowledge of a given circuit, as well as properties useful for constructing circuit partitions. Figure 5.3 illustrates a simple circuit comprised of six gates. The right figure shows the two possible paths that the output signal from G0 is required to propagate (P1 = {G0, G1, G2, G3, G4} and P2 = {G0, G5, G4}) before in route to the primary output. The left figure shows a triangular-matrix with each box containing the cell(s) with matching the ASAP and ALAP indexes. The max depth is 5 (0 to 4) and can be represented by the P1 (or the diagonal-edge of triangular-matrix), which is more likely to form the critical path over P2 since the number of stages that a signal must propagate (5 vs 3) is larger. It is important to note, however, that although P2 is less critical in terms of depth than P1, the

ASAP and ALAP indexing scheme cannot guarantee that P2 is less critical (e.g., delay) than P1. However, we foresee that under the most general case, the ASAP and ALAP indexing can provide key insight in identifying which cells are more likely to participate in the critical paths, thus, enabling efficient power and delay trade-offs to be performed during optimization.

5.4.3 Circuit Partitioning

A major challenge in circuit partitioning is identifying the metrics to group cells by as well as the group size. We define the following objectives in establishing our circuit partitioning scheme: 1) partitions are configured such that each independent optimization captures and maintains the global optimization picture accurately; 2) partition size is kept small (e.g., hundreds of gates) to improve simulation time, reducing the number of computations normally required when performing sensitivity analysis (e.g., power vs delay trade-off) across the entire circuit; and 3) multi-threaded support by optimizing partitions independently across a set of available threads. We group cells with similar delay affinities by using their respective ASAP and ALAP indexes as a partitioning mechanism. Our ASAP and ALAP grouping analysis showed that most circuits provide enough sparseness in group sizes, with most individual ASAP and ALAP groups ranging from less than 1% to 5% for a given circuit, translating to few tens to few thousands of cells.

We define seven micro-partitions in defining the building blocks for circuit partitions (Figure 5.4a), which can be used to form larger macro-partitions of larger sizes (Figure 5.4b). The partition shapes are chosen carefully such that they any circuit depth (ASAP, ALAP) can be constructed by the elementary micro-partitions. Each micro-or macro-partition can be independently treated as its own circuit. However, it is important to maintain coherency across its transitively affected input and output groups; this can be achieved by preserving the cell configurations of connecting inputs and outputs to the group. For example,

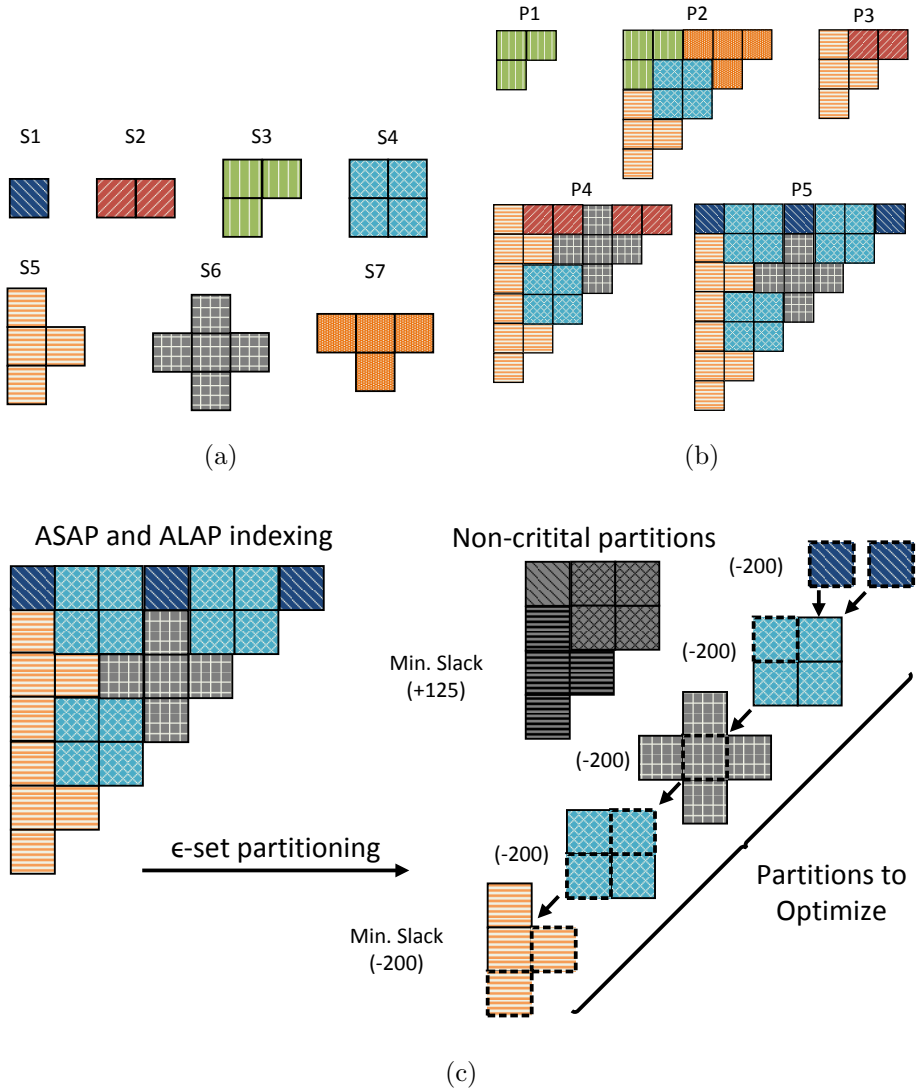


Figure 5.4: ASAP and ALAP circuit partitioning : (a) 7 enabled micro-partitions; (b) 5 example macro-partitions using micro-partitions; c) ϵ-set of macro-partitions satisfying ϵ-delta slack.

the cell drivers of a partition can be treated as the primary input virtual drivers with fixed configurations, and the load capacitance driven by its primary outputs as virtual loads. Delay and power coherency is maintained by invoking a circuit update procedure after each global optimization phase. The update procedure updates affected partitions during the last iteration with the latest configurations of any fan-in/fan-out neighbors (e.g., input drivers and output loads).

ASAP and ALAP indexing also provide a useful property that can be utilized for constructing circuit partitions composed of only critical partitions. A dependency graph may be formed by connecting macro-partitions (from primary-in to primary out) via ASAP and ALAP indexing. For example, a given partition at ASAP stage i and ALAP stage j can only have input sources from ASAP and ALAP indexes who's index satisfies $\leq i-1$ and $\geq j+1$, respectively. For instance, referring to Figure 5.3, cell G3 cannot have a source from G5 since it violates this property. This property useful for constructing ϵ-sets, explained later, which are composed of macro-groups that are within an ϵ-delta slack (Figure 5.4c).

5.4.4 Cell Configuration Selection

We adopt a maximally constrained, minimally constraining heuristic for selecting cell configurations. The maximally constrained principle states that the configurations for the most difficult cells (or groups) are determined early in the optimization while there is still slack in the design with respect to the desired objectives (e.g., timing and power). The minimally constraining principle states that the configuration is chosen such that it minimally impacts the optimization search space (e.g., available future moves) [86].

5.4.4.1 Identifying Critical Partitions

We first compute the accurate delay, slack, and power of the entire circuit *without PV* and identify partitions that are within a predefined ϵ-delta of the critical slack. These cell ASAP and ALAP micro-partitions are used to form an ϵ-set of macro-partitions with paths leading from the primary inputs to primary outputs (Figure 5.4c). The next step is to identify partitions which are more likely to be impacted by process variation. This can be achieved by generating 100 circuit scenarios (instances) to quantify the impacted of PV in regards delay and power yields. In our experiments, we found that the number of iterations can be kept small if the number of cells per group is small with high-confidence. With this method, the variation of a macro-partition can be quantified by the following equations:

$$P^j_{var} = \sum_{i}^{|P^j|} Cell^i_{var} \tag{5.1}$$

$$Cell^i_{var} = \sum_{k}^{|F_{in}|} Std_i \cdot Slk_i \tag{5.2}$$

The delay variation of a partition is the sum of the partitions primary input/output delay cell variation ($Cell_{var}$) for each cell $i = 1$ to $|P^j|$, where $|P^j|$ represents the number of cells in partition j (Eq. 5.1). The delay variation of a cell is simply the sum of the products between each input pin $k = 1$ to $|F_{in}|$ and its corresponding slack Slk_i. Note that only computing the $Cell_{var}$ of a partitions primary inputs/outputs is necessary since they connect to paths to other fan-in/fan-out partitions, effectively preserving the variational impact on the global circuit delay and power.

In order to reduce the number of required computations, only groups included in ϵ-set are considered during an optimization phase, since the rest are considered non-critical in terms of achieving the desired yield objective (e.g., delay and/or power). The ϵ parameter may be adjusted to consider more groups, enabling more partitions to be optimized simultaneously towards a desired objective.

5.4.4.2 Quantifying Group and Cell Difficulty

Given ϵ partitions, the next step is to identify the top K-difficult cells within a partition. This is quantified by the formulation below:

$$Cell_{diff} = |F_{out}| \cdot |F_{in}| \cdot (-Slk) \tag{5.3}$$

$|F_{out}|$ and $|F_{in}|$ represents the fan-out and fan-in length of cell, and Slk the slack of a given cell and represents the minimum Slk value across a cell's input pins. We, therefore, identify the most difficult gates within a particular group as the top-K cells via $\max(Cell_{diff}^{i=1\,to\,|P_j|})$, where $|P_j|$ represents the number of cells in partition j. We use Eq. 5.3 to model an individual cell's difficulty, since the number of paths potentially affected by a given cell is attributed by the number of signals that must propagate through it, accounted by the number of fan-in and fan-out connections. Slk is used in order to identify cells within a group that are likely to participate in the critical paths.

Given the top-K most difficult cells within a partition, the last step is to determine the cell configuration or move (e.g., size, V_{TH}) with an incremental/decremental offset of *one* that benefits the target objective. As an example, assuming under delay-constrained optimization, we can classify the result of a considering a valid move for each cell into three categories:

1. Power and delay reduction.
2. Power reduction and constant delay.
3. Power reduction and delay increase.

It is important to account only for valid moves; moves that lead to load or slew violations are disregarded. Valid moves are then assigned priorities in the precedence class order of *i*, *ii*, and *iii*. Moves that benefit both power and delay (class *i*) are always selected over moves belonging in classes *ii* and *iii*, and are

compared against other moves within its own class as the product of power and delay savings. If no class i moves exists, then class ii moves are selected by the maximum total power improvement. If only class iii moves are found, the move that produced the maximum $\frac{benefit}{cost}$ is selected. The above objective concepts may be applied inversely when a power-constrained delay minimization objective is set.

To prevent the algorithm from being stuck in a local minima, only K-cells are configured during an optimization iteration. Based from our experiments, we set K=10% of the total group cell count, which can be adjusted to affect convergence rate. Once a cell configuration is chosen, it is locked and cannot be altered until the completion of an optimization iteration. An optimization iteration is complete once partitions belonging in the ϵ-set have been visited. Optimization iterations are repeated until a convergence criteria is satisfied (Figure 5.2).

5.5 Simulation Setup

We evaluate our approach using industrial benchmarks included in the ISPD Design Contest 2012 suite [96]. Each design was optimized in accordance to industrial imposed constraints, satisfying as max load capacitance and input slew limits. Power and timing results were generated using an in-house timer implemented in C++, which we correlated in good spirit against Synopsis PrimeTime to be within 1e-3 error. We extend the original cell library to support near-threshold cells (N_{VT}), which were generated using analytical models from Markovic et al. [154]. A variation factor ($3\delta/\mu$) of 30% was used to model the variation of a standard inverter.

We perform Monte Carlo simulations to obtain the best fitting parameters that minimized error against the standard cell library. We obtain fitting parameters independently for both delay and leakage with respect to each cell table entry: 1) delay type {rise, fall} per {delay, input transition}; and 2) capacitance and input

Table 5.1: Target clock (delay) for each benchmark (col. 2); the achieved delays (1000 instances) when considering PV (col. 3); and adjusted target clock with PV for using [60] (col. 4).

Circuit	Target Clock (ps)	Avg. !NTC-PV	Adj. !NTC-PV
dma	1800	1878	1500
pci_bridge32	1400	1626	1200
vga_lcd	1400	1689	1250
des_perf	1600	1754	1450
b19	4300	4546	3800

{rise, fall} transition index. The fit was performed across the three V_{TH} (S_{vt} = 0.33V, M_{vt} = 0.27V, and F_{vt} = 0.20V), and V_{DD} = 0.70V [96]. The final model resulted with error less than 5% per cell delay (e.g., rise/fall input transitions), and less than 1% in leakage. The fitting parameters were used to generate an NTC cell library configured with N_{vt} = 0.68V. Note that the corresponding cell library is no longer used when applying PV factors into the design since the affected factors we consider in our PV-model (Section 2.3). The obtained fitting parameters are applied directly during the delay and leakage table look-up. To validate our approach, we simulate each design against the look-up-table model and achieved power and timing errors within 8% to the reference industrial tool using the original and NTC-generated cell library for the circuits we consider here.

5.6 Experimental Results

We compare our approach NTC against an Non-NTC (!NTC) multi-V_{TH} gate-sizing method proposed by Li et al. [60]. Their approach achieved competitive results against solutions obtained from the ISPD 2012 design contest [96] and [59]. Due to a NTC cell library compatibility issues with their tool, we only com-

pare their method using the original ISPD multi-V_{TH} library (non-NTC library). Additionally, circuits *leon3mp* and *netcard* are also omitted due to tool issues.

Due to performance impact that is incurred when enabling NTC, for timing comparisons, we relax the target delay constraints (2X slower) used in the original ISPD design contest suite (Table 5.1, col. 2). To ensure fair timing and power analysis when considering PV, we used the our in-house timer to report timing and power results under PV (see Section 5.5) for solutions obtained by both methods. Due to space constraints, we omit reporting detailed simulation run-times of our approach, but note that our approach achieved 1.2X to 3.5X run-time increase over the compared method in [60]. This is expected since our approach requires additional computational overhead for performing tasks described in Section 5.4.

To identify equivalent timing and power constraints for fair comparisons, we first generate solutions using the method reported in [60] to achieve reference clock targets shown in Table 5.1. Next, the obtained solution is fed into our PV-aware timer to obtain the actual PV-enabled delay and power results. Timing violations are expected, as shown by non-PV-aware result, which are on average 8% (20% max) slower than the original intended target delay. Therefore, new target clocks are determined that achieves 100% yield (Table 5.1, col. 4) under the non-NTC method. We found experimentally that the original target clocks were required to be scaled lower by 12.1% on average.

The non-NTC result for each circuit is considered as the base comparison assuming all circuits (1000 instances) that achieve 100% yield in timing and power. We use this as the target yield objective for our NTC approach, since setting all cells to non-NTC configurations would naturally minimize variation in a design. Therefore, the objective of our approach is to optimize each benchmark using our PV-aware NTC-enabled framework to attain similar yields to that of the non-NTC approach for the given reference clock targets.

Table 5.2: Average (avg.), max (+), min(-) delays when optimizing under NTC-enabled (NTC) non-NTC-enabled (!NTC) [60].

Circuit	NTC				!NTC			
	Avg.	(+)	(-)	Std.	Avg.	(+)	(-)	Std.
dma	1475	1791	1403	48.07	1728	1780	1701	12.6
pci_bridge	1358	1583	1307	31.0	1479	1497	1458	7.16
vga_lcd	1212	1399	1150	40.8	1377	1399	1353	11.7
des_perf	1449	1554	1406	23.5	1475	1504	1450	11.1
b19	4167	4273	4059	36.4	3902	3940	3865	14.7

Table 5.3: Total power results when optimizing under NTC-enabled (NTC) and non-NTC (!NTC) [60]: Shown are the average (avg.), max (+), and min(-), results corresponding to total, leakage, and switching power values. The ratio $\frac{!NTC}{NTC}$ represents the total power reduction factor.

Circuit	Comb.	Total Power (mW)						Ratio	Leakage Power (mW)						Switching Power (mW)					
		NTC			!NTC				NTC			!NTC			NTC			!NTC		
	Cells	avg.	(+)	(-)	avg.	(+)	(-)	avg.	avg.	(+)	(-)	avg.	(+)	(-)	avg.	(+)	(-)	avg.	(+)	(-)
dma	23109	44	46	43	138	141	135	**3.13**	8.2	9.5	7.3	115	118	110	35.8	36	35.7	23	23	25
pci_bridge32	29844	36	38	35	113	115	110	**3.13**	5	6.7	4.1	87	88	84	31	31	30	26	27	26
vga_lcd	147812	115	118	110	509	529	485	**4.42**	16.1	17.2	15.5	420	430	400	98	100	94	89	99	85
des_perf	102427	175	200	164	533	554	526	**3.02**	10.2	12.6	8.7	381	402	330	164	187	155	152	152	196
b19	212674	260	297	251	733	762	722	**2.81**	21.4	23.1	20.2	546	550	542	238	273	230	187	212	180

5.6.1 NTC-enabled PV-aware Optimization

Table 5.2 presents the actual timing achieved by NTC and non-NTC. As shown, the achieved standard deviations by NTC were found to be up to 4.3X (3.2X) larger than the achieved standard deviation from the non-NTC delay results. Thus, in order to achieve 100% yield targets when enabling NTC-design, larger guard bands for timing and power should be enforced. To understand how delay is affected accross a large set of scenarios, Figure 5.5 presents the achieved max, mean, and min delays normalized to their respective reference clocks in Table 5.3. Clearly, non-NTC results achieves significantly lower performance variation over NTC. However, it is important to note that the mean result for each benchmark is closer to the minimum result. Thus, this means that the majority of circuit instances lie closer to the minimum result.

Table 5.3 compares the total power (avg., max, min) for both methods. The results were acquired from 1000 generated circuit instances using our PV model. As shown, NTC achieve significant total power reduction of up to 4.42X (3.30X avg.) over the non-NTC method. The power improvements result from the savings in leakage power, achieving 37.4X max reduction (24.1X avg.) over non-NTC. To achieve this under NTC, more cells have to be up-sized to larger cell configurations in order to meet timing constraints, thus, increasing switching power up to 1.55X (1.24X avg.). However, the total power is still reduced using the NTC approach due to the dominant leakage power component. For example, under the non-NTC approach, leakage power made up 77% (avg.) of the total power budget vs. 12.1% (avg.) for the NTC.

Figures 5.6a-5.6b indicate how delay target yields are achieved through successive iterations using our approach. Additional iterations show improvements in both the number of circuit instances that satisfy the target clock of 1400 ps (100% for !NTC). A 90% yield is achieved after performing 3 design iterations, achieving

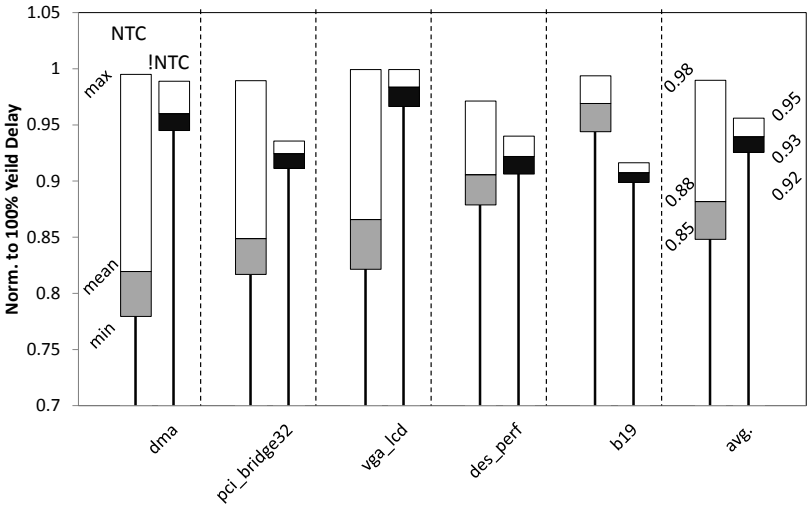
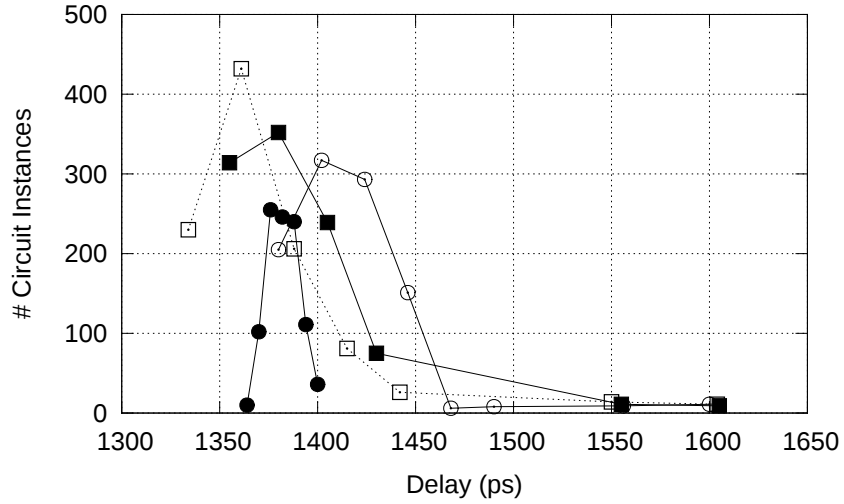


Figure 5.5: Max (top-tier), mean (mid-tier), and min (bottom-tier) target delay ratios among 1000 generated instances with respect to NTC (left) and !NTC (right) [60].

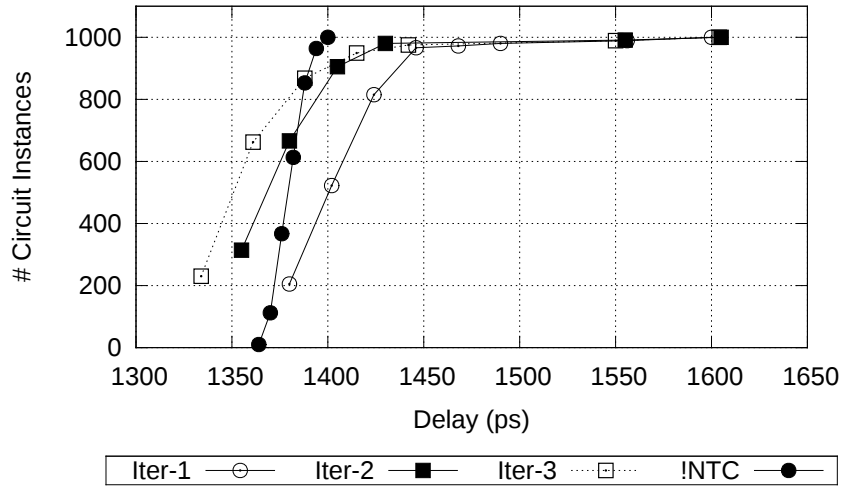
an overal 3.13X (avg.) total power reduction for all valid circuit instances.

5.7 Summary

We have presented a framework for maximizing performance and power yield constraints for near-threshold computing-enabled designs, under the presence of process variation (PV). We apply an NTC-enabled paradigm using popular optimization techniques, such as gate-sizing and multi-threshold cell assignment, while strictly adhering to industrial imposed design constraints, such as load and slew limits. The focal point of our framework is a logic-depth circuit-level partitioning scheme for efficiently characterizing and identifiying critical circuit sections related to timing and power variations of a given circuit. We utilize a scenario-based approach by generating a large set of circuit instances, which is used to identify which cell configurations in an NTC search space maximally benefit solution towards the target yield objectives.



(a)



(b)

Figure 5.6: Timing yield optimization for circuit pci_bridge32: (a) frequency distribution graph compares 3 iterations using our NTC approach against a !NTC ; (b) cumulative dist. graph.

We compare our approach against a state-of-the-art non-NTC approach and show significant reductions in total power of up to 4.4X (3.3X avg) for the same timing and power yield constraints. Consequently, the savings in power results with signficant performance variations that should be addressed when optimizing for maximal yield. The standard deviation in performance for NTC-enabled designs are shown to be 4.3X max (3.2X avg.) than that of non-NTC approach. However, we show that on average, only a few circuits (less than 5%) make up instances that violate imposed timing yield constraints for NTC-enabled designs.

CHAPTER 6

Sequential Circuit Unfolding for Energy and Yield Optimization

In this chapter, we propose a novel gate-level sequential circuit unfolding approach for energy, performance, and yield optimization in near-threshold computing (NTC) systems. Our approach simultaneously addresses several major challenges in NTC systems, such as performance and process variation (PV), while also maximizing its advantages, such as low leakage energy. Three main benefits are enabled by our technique. First, unfolding can be leveraged to improve performance, as the delay per operation is reduced. Second, energy efficiency can be improved by placing a larger ratio of cells at lower energy configurations where there exists unfolded slack, enabling greater savings when conducting popular circuit optimization techniques, such as gate sizing and threshold voltage (V_{TH}) selection. And third, unfolding can improve circuit resiliency against the impact of process variation. This chapter presents a synthesis methodology that coordinates unfolding with gate sizing and V_{TH} selection to maximize energy savings for NTC systems while addressing performance and PV concerns. We evaluate our approach on the ISCAS-89 benchmark suite using a near-threshold cell library in accordance to an industrial flow. We show significant improvements of up to 15.4X in energy and 1.8X in throughput with respect to identical timing yield constraints when applying our technique.

6.1 Introduction

As the operating gap between supply voltage (V_{DD}) and threshold voltage (V_{TH}) shrink with each device technology generation, standard techniques (e.g., gate sizing, supply voltage scaling) become less effective, while existing challenges (e.g., leakage power, performance variation) become more pronounced. Fortunately, this area has gained increased attention and opportunities have emerged showing promise in addressing energy, performance, and yield (e.g., near-threshold computing) [61][62][63][66]. However, CAD frameworks lack the ability to fully leverage the benefits of these opportunities, as well as address the aforementioned challenges. Therefore, in order to properly optimize modern and pending VLSI circuits, CAD tools must evolve to match the continuously changing physical landscape that accurately accounts for power (e.g., switching, leakage) and performance (e.g., delay, throughput), and how each of these characteristics are impacted by process variation (PV) [61].

Recently, near-threshold computing (NTC) has been proposed to enable ultralow energy operation of digital circuits. The benefits of NTC include up to a 10X reduction in energy as compared to super-threshold operation, but come at cost of a 10X increase in delay along with higher susceptibility to process variation. Furthermore, in NTC operation, load imposes a much larger scalar impact on delay.

In this chapter, we present a novel gate-level unfolding technique for NTC sequential circuits that simultaneously reduces the impact of process variation through deep logic reconfiguration and minimizes delay through load reduction. We leverage our approach in concert with popular circuit optimization techniques, including gate-sizing and V_{TH} selection, to produce energy efficient designs that are resilient against PV with respect to yield. We highlight our contributions in the following areas in which our circuit unfolding technique enables modern CAD

flow optimizations for NTC circuits:

- **Performance:** Traditionally, unfolding accomplishes performance gains through the removal of intermediate flip-flops between success results, thus eliminating incurred register timing requirements between successive operations. Unfolding also increases the number of non-epsilon critical path gates. We capitalize on this feature through gate sizing to redistribute load from epsilon critical path gates to non-epsilon critical path gates, subsequently reducing critical path delay. By relaxing constraints on the critical path in this manner, we also simultaneously reduce their sizing requirements, thus reducing their energy consumption rates.

- **Optimization**: Although unfolding incurs additional area overhead due to cell replication, the increased logic depth provides additional optimization opportunities when performing gate sizing and V_{TH} selection. The longer logic depth allows additional choices when performing delay and power trade-offs. This can become essential when conducting optimization across a wide-array of circuit considerations.

- **Yield**: Increased logic-depth is leveraged to improve resiliency against the impact of PV on circuit delay. By increasing the logic depth using unfolding, the likelihood that a single negatively-impacted (slow) cell severely affects the overall delay is reduced since there is an equal likelihood that another cell on the same path would be positively-impacted (fast). Ultimately, the impact of individual cells is averaged out over long chains.

We evaluate our approach on a set of sequential ISCAS-89 benchmarks using an industrial flow on a multi-V_{TH} cell library that includes fast and high-leakage fast threshold (FV_{th}) cells, as well as ultra-low leakage and slow near-threshold (NV_{th}) cells. We incorporate our unfolding techniques with our gate sizing and threshold

selection algorithm and demonstrate energy reduction by up to 8X against our benchmarks.

6.2 Motivation

We first introduce a simple circuit unfolding example as shown in Figure 6.1. Given the original circuit, as shown in Figure 6.1a, composed of flip-flops (FF1-FF4), combinational blocks (C1 and C2), primary inputs (I1 and I2), and primary outputs (O1 and O2), the act of unfolding consists of replicating all combinational elements (C1 and C2) and required flip-flops (F1 and F3), as shown in Figure 6.1b.

Conducting circuit unfolding on a circuit enables several advantages over its non-unfolded counterpart. One major advantage is the reduction in clock period, as shown in Figure 6.1a to 6.1b. The clock period is reduced by removing the timing delays (e.g., setup, hold, and propagation delay) of intermediate flip-flops, as well as the required combinational delay by connecting the more critical combinational block, C2, with the less critical, C1, in the subsequent stage. This phenomena is present in many of the sequential circuits we study here. Assuming that all flip-flops incur a delay of 1 time unit, and C1 and C2 delays of 2 and 4 units, respectively, then producing two results on the original circuit (Figure 6.1a) requires 10 time units ($(1+4)\times 2$), whereas a 2X unfolding circuit requires only 7 time units ($1+4+2$). Here, performing unfolding alone yields 30% improvements over the original non-unfolded circuit (Figure 6.1c). Improvements in delay come at additional cost in area due to an increased number of gates from replicating C1 and C2 combinational blocks. However, the area overhead of flip-flops is less since only F1 and F2 are required to be replicated and also fall under non-critical paths, thus, minimally impacting delay.

The structural transformations from performing circuit unfoldings provide designers with additional optimization freedom in conducting popular circuit opti-

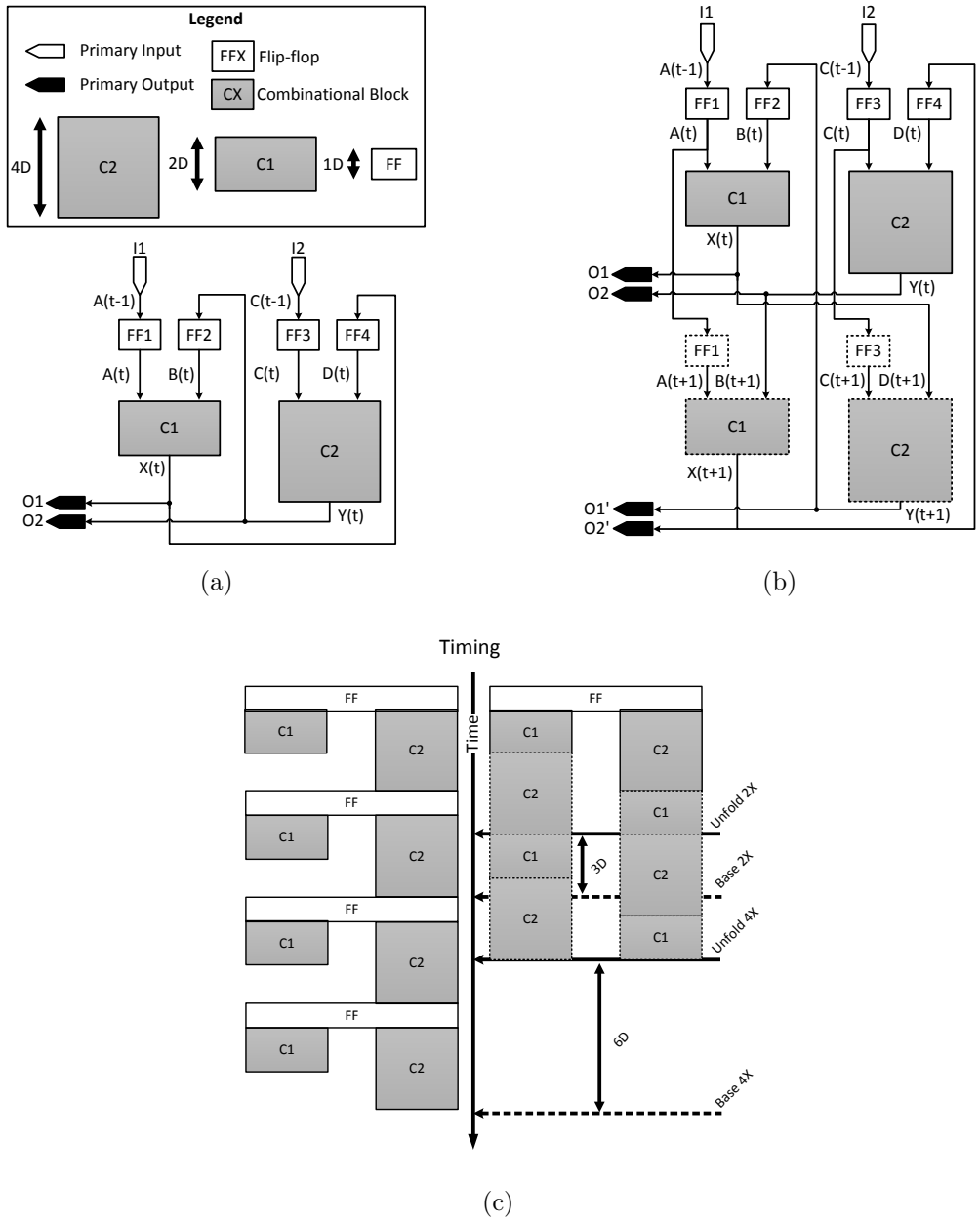
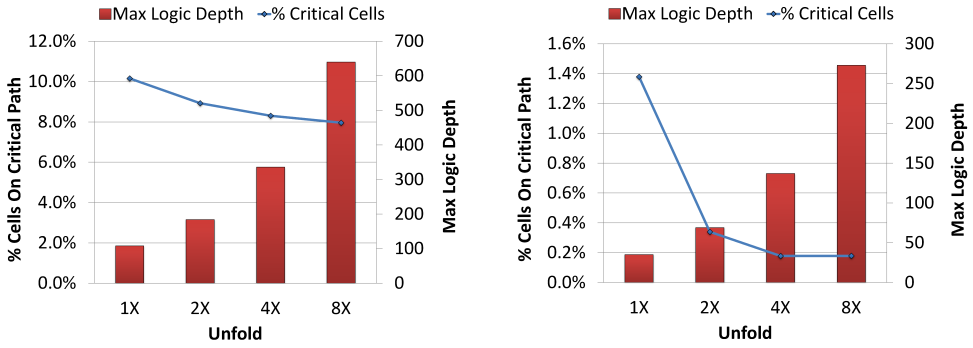


Figure 6.1: A circuit unfolding example showing original circuit (a); 2X unfolded circuit with replicated elements represented with dashed-borders (b); and timing diagram comparing the time required for up to 4 results of a non-unfolded circuit (left) and up to 4X unfolded circuit on the right (c). Sequential elements (flip-flops) are shaded green, primary input (output) elements are non-shaded I1 and I2 (shaded O1 and O2) buffers.

mizations, such as gate-sizing and V_{TH} selection. The increased optimization opportunity stems from the increased delay slack from performing unfolding alone. Therefore, the additional delay slack can be reclaimed for energy (power) savings while still maintaining a specified target delay through gate-sizing and/or V_{TH} selection. Furthermore, the number of non-critical cells increases per additional unfolding, thus, enabling them to be placed at lower power configurations. This phenomena bodes well for unfolded circuits, since most often the critical path consists of a small percentage (1 to 10%) of the total gate count of the original circuit; unfolding these circuits only further compounds this effect resulting in an even smaller percentage of cells that make up the epsilon critical paths.

Figure 6.2a-6.2b depicts the impact each additional unfolding has on a circuit with respect to the most critical paths. Here, a less than perfect scaling is shown per additional unfold. For example, circuit s1423 resulted with 95 logic stages for the original circuit (1X), 155 vs 190 at 2X, 290 vs 380 at 4X, 645 vs 760 at 8X. Simultaneously, the percentage of cells that make up the critical path is reduced per unfold, with a 7X reduction factor (1.4% to 0.2%) for the much larger circuit s38417.

Unfolding may also be leveraged to improve design yield, such as maximizing the percentage of circuits that satisfy a target delay. Unfolding a circuit enables it to become more resilient to the effects of process variation (PV). By increasing the logic-depth through unfolding the PV impact of a single cell on the entire design is reduced with respect to delay. Because unfolding increases the logic depth size, the overall delay variation decreases with respect to its original non-unfolded circuit. Furthermore, since the circuit delay is likely to be made up of the longest paths, and the percentage of gates that participate on these critical paths decrease per unfold, we improve yield by focusing gate-sizing and threshold selection techniques on the gates along these epsilon critical paths.



(a) (b)

Figure 6.2: Cell count distributions with respect to each respective unfolding (1, 2, 4, 8X) for (a) s1423 and (b) s38417. Unfolding sequential circuits reduces the ratio of the number of critical cells with respect to total cell count.

6.3 Related Work

First proposed by Parhi et al. [56], unfolding is a well known behaviorial transformation technique, which replicates functional blocks (e.g., adders, multipliers) in order to increase throughput performance, while mainting functional correctness. Behaviorial unfolding has been studied extensively and is a popular optimization approach that uncovers hidden concurrency in DSP applications. At the bit serial-level, Song et al. have explored for bit-serial/word-level architectures [57]. These approaches, however, are performed at a higher abstraction level and primarily focus on the performance benefit of unfolding. Our work explores unfolding at a gate/logic-level and leverage its benefits when performing energy, delay, and yield optimization.

Gate sizing and threshold-voltage (V_{TH}) selection are popular techniques for conducting power, area, and timing optimization. We study a discrete optimization search space using a standard cell library. A cell library provides for each cell type (e.g., inverter, nand, etc.) a set of sizes across multiple V_{TH}-levels, each with varying delay and power characteristics. Careful adjustments must be per-

formed and it comes to no surprise that in the discrete cell-based domain, gate sizing and V_{TH} selection is NP-hard when optimizing a circuit [97]. Gate sizing methods are largely sensitivity-driven, which apply a set of sensitivity functions for determining the most beneficial local cell configuration to choose. Approaches include, Lagrangian relaxation-based, dynamic programming [60][115], and sensitivity driven techniques [58][59].

Near-threshold computing (NTC) has shown promise in achieving significant energy efficiency [61][62][63].However, several challenges exist that complicate an NTC-enabled design, that includes: 1) performance degradation; 2) process variation impact; and 3) increased area cost. performance targets and restrict their optimization methods to gate-sizing and V_{TH} selection only. We explore multiple delay targets and demonstrate how circuit unfolding can be leveraged in achieving these targets while conducting circuit optimizations on NTC designs.

6.4 Preliminaries

6.4.1 Circuit Unfolding

Given a synchronous circuit (G), which is composed of a set of registers (R), combinational elements (C), and their respective primary inputs (PI) and outputs (PO), we present the following algorithm for unfolding G to any arbitrary length. The first step in unfolding a circuit requires a complete replication of all elements (R, C, PI, and PO). Next, the output connections of the original circuit (before unfolding) are connected as inputs to the replicated circuit. To enable two operations per cycle, the intermediate registers are removed; this breaks the sequential dependency, as shown in Figure 6.1a with replicated registers $FF2$ and $FF3$. The corresponding primary inputs and outputs are subsequently connected. Finally, the corresponding outputs of the replicated circuit are connected as inputs to the original circuit.

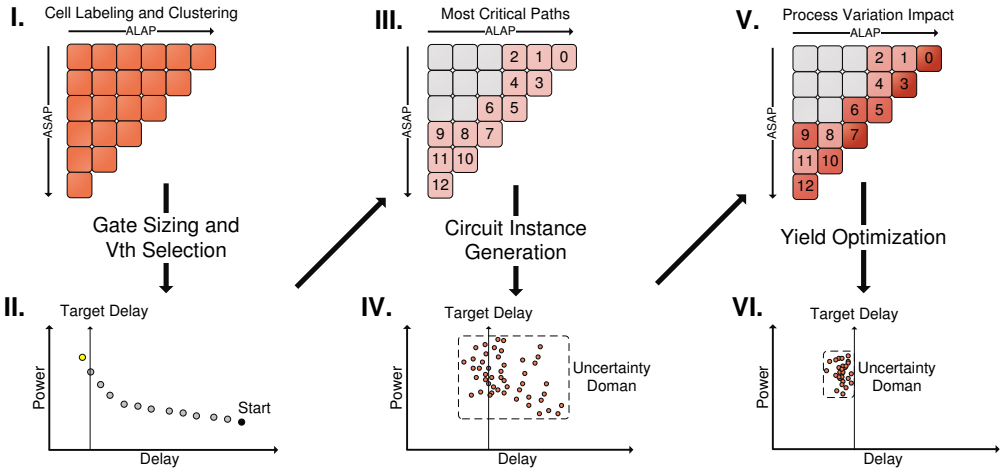


Figure 6.3: Gate sizing optimization flow.

6.5 Technical Approach

This section describes the major steps we include in our approach, as depicted in Figure 6.3. Given a pre-unfolded circuit as a starting point, the circuit undergoes a series of phases (I. to VI.) in route to producing a final circuit that satisfies performance and yield targets. We describe each phase in detail in the following subsections.

6.5.1 Gate Sizing and V_{TH} Selection

Our gate sizing and V_{th} selection begins with all cells initially set to their minimal power configuration (Step II). To address the scalability issue, at each cell sizing/V_{th} phase, only a small subset of gates (cells) that fall within ϵ-critical slack are set as candidates for optimization. Usually, this subset consists of 1-10% of total gates in the circuit. Once a configuration is determined for each cell, it is locked and cannot be re-evaluated until all gates in a sizing phase have been sized appropriately. Only when all ϵ-cells have been configured, are all modified cells unlocked and circuit delay is re-computed.

We introduce a cell difficulty ($Diff$) ranking measure in order to rank which

cells are configured first during each sizing iteration. Cell difficulties can be quantified utilizing the following properties: 1) rise/fall slack (S); 2) fan-in(out) length (F_{in}, F_{out}); 3) $ASAP$, $ALAP$ depth (i, j); 4) absolute depth (i+j); and 5) switching activity ($\sum_{k=0}^{|F_{in}|} \alpha_k$).

$$Diff = \sum_{k=0}^{|F_{in}|} \alpha_k \cdot w_a + F_{in} \cdot j \cdot w_b + F_{out} \cdot i \cdot w_c - S \cdot (i+j) \cdot w_d \tag{6.1}$$

In Equation 6.1, i refers to ASAP depth and j refers to ALAP depth. Weights w_{a-c} are experimentally determined. The first term, w_a, considers the switching power due to the switching activity of a given cell's input nets, w_b (w_c) considers the cell's potential impact on cells upstream (downstream), and w_d quantifies the impact on delay.

Equation 6.1 is applied to the selected ϵ-critical cells obtained during the last accurate delay and timing computation. Cells are configured based on their $Diff$ value in descending order. For example, if we assume that the w_d-term is the largest, then the cell that has the smallest slack will be configured first. During this configuration, only local updates of one degree are executed. Once the entire $\epsilon - set$ has been configured, then an accurate circuit-wide update is performed. By employing local updates during these individual configurations we are able to reduce computation cost while still maintaining accuracy.

6.5.2 Optimizing for Yield

In our yield optimization steps we aim to optimize those cells that are on epsilon critical path that also happen to be the most impacted by PV. Cells that are most impacted by PV will be those with higher threshold voltages since they are closer to V_{DD}. For example, a fluctuation in threshold voltage (due to PV) for a nominally high threshold voltage cell will have a greater impact on delay than the same scalar fluctuation has for a nominally lower threshold voltage cell.

We define the cell delay variational factor (Var_{delay}^{cell}) as the ratio between the

cell delay standard deviation and mean ($\frac{\sigma}{\mu}$) computed from a representative set of PV-impacted circuits. A similar method can be performed to quantify the power variation factor (Var^{cell}_{power}). Var^{cell}_{delay} determines which cells in the top ϵ-critical paths (pink shaded boxes in Step III) are likely to impact delay yield. The ϵ-critical path is defined as cells that are within a specified threshold of the critical delay, and can be obtain by performing STA across a set of PV-impacted circuits as well. The Var^{cell}_{delay} can be expressed as follows:

$$Var^{cell_i}_{delay} = max((\frac{\sum_j^{|F^j_{in}|}\sigma_{i,j}}{\mu^{rise}_i})^{rise}, (\frac{\sum_j^{|F^j_{in}|}\sigma_{i,j}}{\mu^{fall}_i})^{fall}) \tag{6.2}$$

$|F^j_{in}|$ represents the fan-in j of cell i. Thus, the cell Var^i_{cell} is the maximum variational factor of its rise and fall delays. This pre-processing step is conducted in Step IV before optimizing the circuit further. Once the Var^{cell}_{delay} has been obtained for the top ϵ-critical paths, then each $ASAP$ logic depth is assigned a Var^{ASAP}_m difficulty value, which is the ratio of the average Var^{cell}_{delay} from cells within a $ASAP$ stage m over the total number of cells belonging to $ASAP$ depth m. Similarly, corresponding variables are defined and computed for $ALAP$. A larger Var^{ASAP}_m value indicates that the current $ASAP$ depth m is projected to have a higher likelihood to impact delay (Step V), as represented by the darker shaded boxes (Step V). Thus, improving the delay resiliency of these cells is expected to improve yield of the final result shown in Step VI (Figure 6.3).

6.6 Simulation Setup

We utilize the ISPD design contest cell library and follow their industrial flow [96]. The cell library (V_{DD}=0.7V) is extended to support near-threshold NV$_{th}$ cells (V_{TH}=0.67V) by using Markovic's EKV formulas [154]. We sweep V_{TH} from 0.23V to 0.67V to obtain fitting parameters, which are applied against each cell V$_{th}$ type to generate the corresponding near-threshold cell (NV$_{th}$). The resultant

near-threshold cell library is fitted on average within 5% of the original cell library values with respect to each cell configuration (per size and V_{TH}). An in-house timer, implemented in C++, is used in order to account for both optimization flexibility and performance purposes. For timing validation, we validate our timer against the Synopsys PrimeTime^{TM} Design Compiler and are accurate within 0.1%.

We generate 1000 circuit instances per benchmark to account for the impact of process variation (PV) on yield. A unique variability ratio pair (V_{th}^{var}, L_{eff}^{var}) is generated for each cell using a 3σ distribution with respect to each cell's nominal threshold voltage (V_{TH}) and gate effective length (L_{eff}). Our PV-aware timing and power computation replaces the values returned by the look up-table with EKV formulas from [154] and returns a PV-impacted value utilizing respective ratios.

Benchmark	Total Cells				Maximum Logic Depth				Ref u1 Delay (ns)				Ref u1 Energy (mJ)				Ref u1 Area (mm^2)			
	u1	u2	u4	u8	u1	u2	u4	u8	mdp	medp	mep	mes	mdp	medp	mep	mes	mdp	medp	mep	mes
s1423	1073	2072	4070	8066	109	185	337	641	3.04	5.53	7.36	3.04	27.63	2.79	3.17	27.63	43.76	14.66	12.66	12.66
s1488	1577	3148	6290	12574	32	64	124	246	1.44	1.85	4.60	1.44	29.32	7.92	10.63	29.32	114.77	52.89	40.59	40.59
s1494	1587	3168	6330	12654	33	65	125	247	1.49	1.97	4.46	1.49	28.43	9.86	11.35	28.43	105.20	51.57	43.38	43.38
s5378	3523	6867	13555	26931	33	57	99	189	0.84	1.84	2.55	0.84	237.70	4.33	4.76	237.70	400.65	18.68	16.13	16.13
s9234	7564	14917	29623	59035	84	117	209	393	2.05	2.20	2.75	2.05	316.81	181.97	91.45	316.81	329.92	300.21	240.17	300.21
s35932	22977	44226	86724	171720	36	70	138	274	1.73	2.05	2.19	1.73	580.06	139.01	146.80	580.06	1908.16	497.50	467.22	467.22
s38417	29317	56998	112360	223084	57	86	168	332	1.89	2.37	4.17	1.89	339.28	224.87	149.04	339.28	703.41	202.06	155.88	155.88
s38584	31088	60750	120074	238722	77	124	218	406	2.14	3.45	5.45	2.14	335.90	208.58	129.52	335.90	461.62	455.39	364.32	455.39

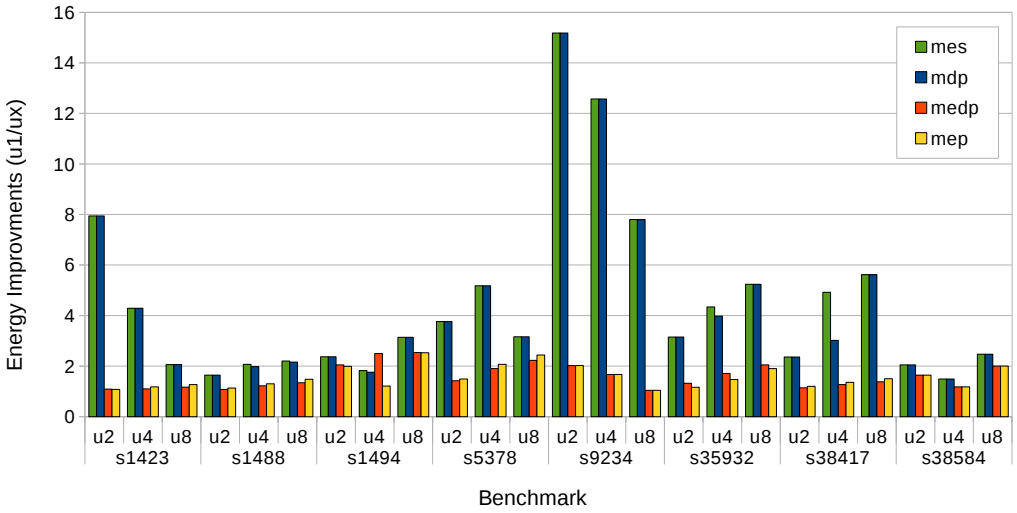
Figure 6.4: ISCAS-89 benchmark characteristics (cell count, logic depth) across 1, 2, 4, and 8X unfolds ($u1$, $u2$, $u4$, and $u8$). Also shown are the representative reference delay, energy, and area for each considered performance points: 1) minimum delay point (mdp); 2) minimum energy-delay product ($medp$); 3) minimum energy point (mep); and 4) maximum energy saving (mes).

6.7 Experimental Results

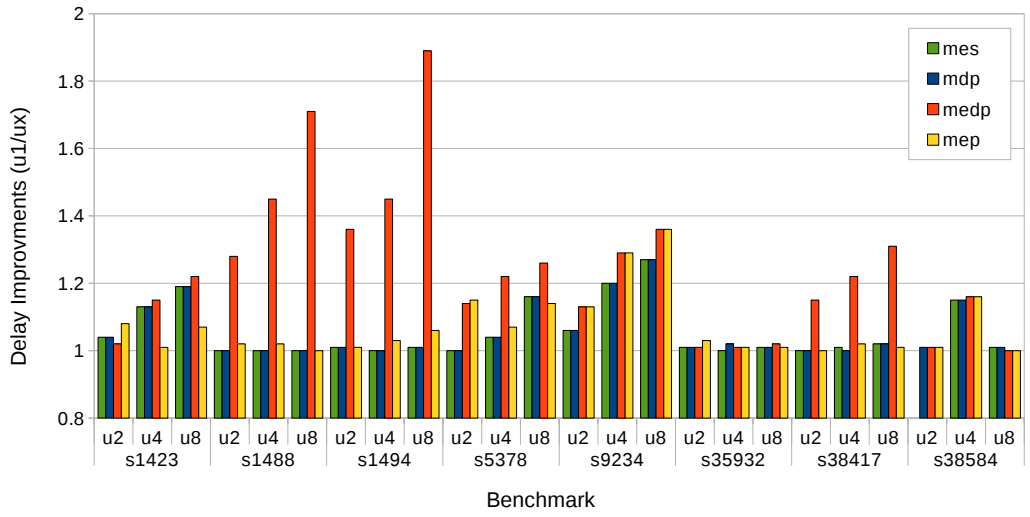
We compare energy and delay results of a non-unfolded circuit ($u1$) against 2X, 4X, and 8X unfolds ($u2$, $u4$, and $u8$) with respect to four delay references: minimum delay (mdp), minimum energy-delay product ($medp$), minimum energy point (mep), and maximum energy savings (mes) for each respective $u1$ circuit. Figure 6.4 presents benchmark characteristics used when comparing each $u1$ reference point, respectively. To compare results fairly, the energy and delay result of $u1$ is scaled to each respective ux unfolding throughput (ops/cycle). For example, comparing the energy of a non-unfolded circuit u1(mdp), to the best satisfying 2X unfolded circuit u2(mdp), requires multiplying the energy and delay of the $u1$ reference point by 2 (i.e., 2 operations per cycle). Then, the minimum energy point of a corresponding $u2$ circuit that satisfies the delay of u1(mdp) is used.

Figure 6.5a shows the energy reduction factor $\frac{u1}{ux}$, achieved at 2, 4, and 8X unfolds for each benchmark. Up to a 15.4X in energy reduction is achieved at u2(mdp) for circuit s9234. Large savings are due to the tight delay constraints applied on the $u1$ circuit, requiring a larger ratio of cells to be configured at larger and/or lower V_{TH} configurations for achieving high-performance delays. Through our coordinated unfolding and gate-sizing techniques we are able to relieve the load within this constrained circuit and obtain a substantial amount of energy savings. Figure 6.5b provides the delay reduction factor, achieving up to 1.9X performance improvement and 2.8X energy reduction for s1494's u8($medp$) point.

Efficient area scaling is shown across most unfolded circuits as shown in Figure 6.6a. Most benchmarks achieved better than perfect linear scaling ($\frac{area(ux)}{area(u1)}$) with respect to its unfolding setting. Circuit s1423 achieved less than 2X area increase across mdp, $medp$, and mep delay points for unfolds $u2$, $u4$, and $u8$; in fact, u8 achieved the best area scaling of 1.10, 1.2X, 1.15X with respect to its $u1$ reference. Efficient scaling is achieved by the large range of cell sizes to choose from (1X to

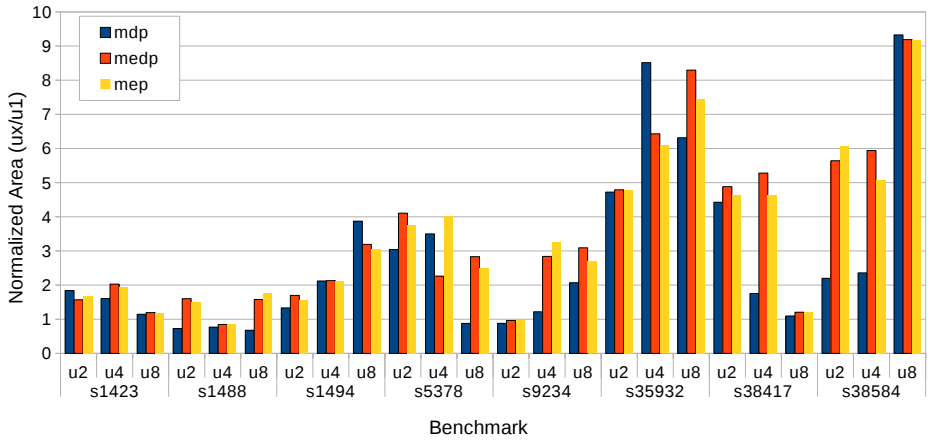


(a)

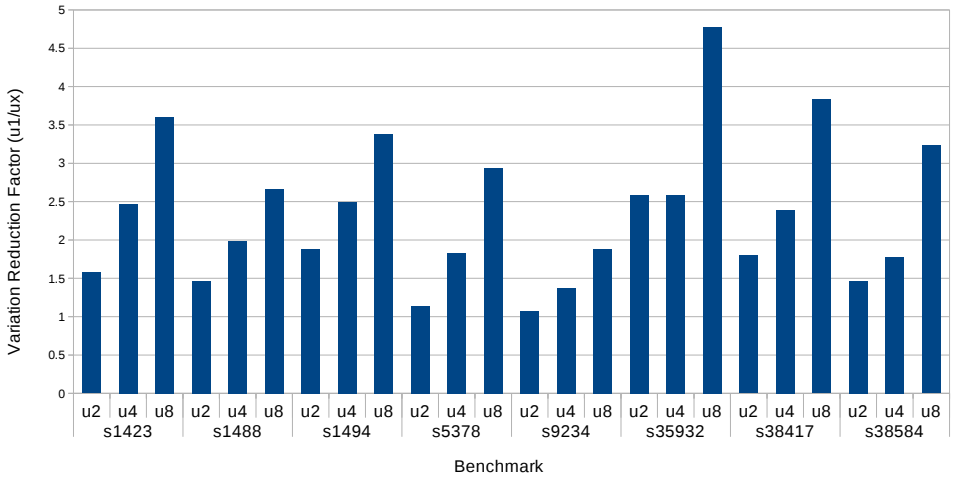


(b)

Figure 6.5: Experimental results showing: (a) achieved energy and (b) delay improvements achieved per unfolding with respect to performance targets (*mdp*, *medp*, *mep*, and *mes*).

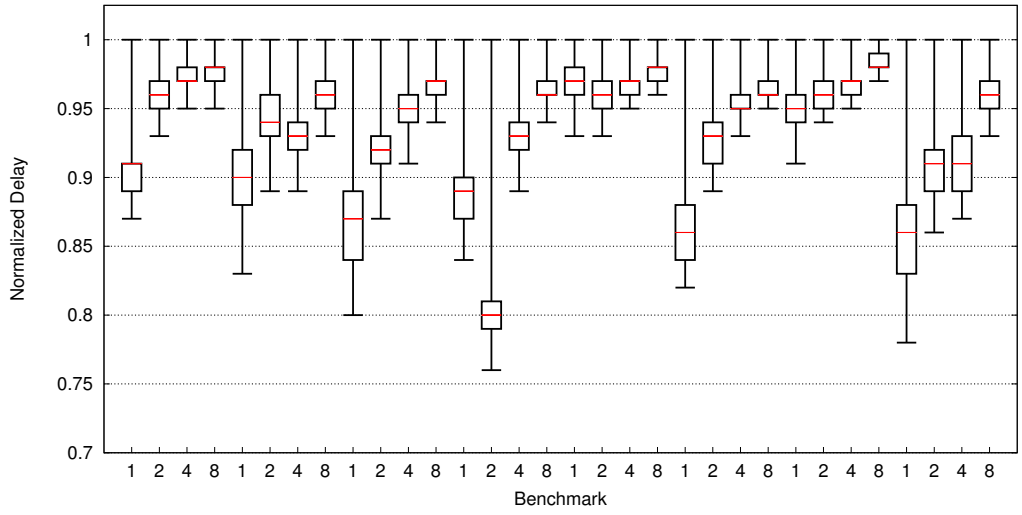


(a)

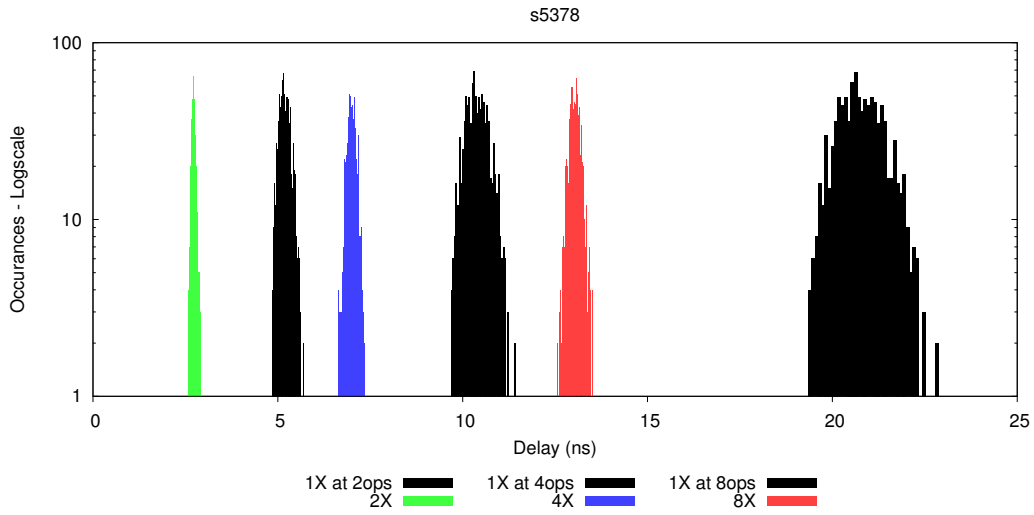


(b)

Figure 6.6: Experimental results showing: a) normalized area with respect to non-unfolded reference ($u1$); b) delay variation reduction factors achieved by unfolding for *medp* delay reference point.



(a)



(b)

Figure 6.7: Experimental results showing: a) quartile graph showing increased resiliency against process variation (PV) for *mep* delay reference; and b) s5378 delay distribution from 1000 PV-impacted using *mep* delay reference point.

512X) in the cell library. Therefore, a reference $u1$ circuit may have a higher ratio of cells set to larger sizes in order to achieve a high-performance delay. In contrast for larger unfolded circuits, the relaxed timing constraints achieved through the performance improvement of larger unfolded circuits causes the gate sizing procedure to require less gates at larger sizes.

Figure 6.6b presents the delay variability reduction factor for respective edp points only. The reduction factor is acquired by computing the delays of the 1000 PV-impacted circuit instances, and then computing the standard deviation and mean (σ/μ) ratio. Delay variability is reduced across all benchmarks as unfolding increases, achieving a max 4.75X reduction for s35932. Figure 6.7a shows a quartile graph for circuit with respect to achieved delays when running each benchmark across its respective 1000 PV-impacted circuits. As shown, the range impacted delay range is reduced per additional unfold. Figure 6.7b presents a histogram on the achieved u1(mep) delays for benchmark s5378 across $u2$ (green), $u4$ (blue), and $u8$ (red) and compares it against its scaled $u1$ reference result (black). The graph shows that the unfolded circuit consistently achieves significant performance improvements per unfold, while also improving delay resiliency to PV.

6.8 Summary

We have presented a novel sequential circuit transformation technique for achieving significant energy and performance improvements. We employ unfolding to increase gate sizing opportunities for load reduction and critical path delay minimization. The focal point of our gate sizing approach is an efficient lock and cut-based gate sizing and threshold-voltage V_{TH} selection algorithm that leverages structural properties (e.g., logic depth, fan-in/out, etc.) for efficiently determining optimal cell configurations in a scalable fashion. Furthermore, we use

deep logic configuration to mitigate the effects of PV. We present an extension to our algorithm for efficient delay yield optimization, which identifies a maximal set of PV-impacted cells to be configured (size, V_{TH}) accordingly, while minimizing overall energy costs.

Experiments were performed using an industrial flow on a representative set of ISCAS-89 benchmarks using a near-threshold standard cell library. Energy reductions of up to 15.4X and 1.8X in performance are achieved across 2X, 4X, and 8X circuit unfolds. We have show that these savings can be achieved with minimal area costs over a representative non-unfolded circuit, while also dramatically improving circuit resiliency against the impact of process variation.

CHAPTER 7

Retiming for Dual-Supply Voltages

Energy is one of the most important design metric in the modern integrated circuit design. We optimize the required energy using a dual-supply voltage (dual-V_{DD}) approach. In order to achieve improved energy efficiency, we use a new type of retiming that reduces the requirements in assigning gates to a higher voltage. We first apply a retiming approach for achieving minimum delay. Next, using a maximum-flow/min-cut strategy, we simultaneously reduce the required number of flip-flop logic and while maintaining delay in such a way that enables efficient V_{DD} assignment in the subsequent stage. We then conduct a new dual-V_{DD} cell assignment technique on the retimed circuit for efficient energy and delay optimization. The overall approach is tested on a set of standard ISCAS89 and ISPD2012 Design Contest benchmark suite using an industerial cell library. Significant energy savings between 8 to 57% (33% avg.) under a specified delay is observed.

7.1 Introduction

Energy has emerged to be the most important design metric in the last 25 years. The two most popular types of systems, mobile phones and data centers are in particular sensitive to energy. For example, battery technology improvements have not kept pace with the rapid device scaling that has continued to follow Moore's Law. For these platforms, mobile phones and tablets are limited by the amount of the energy that can be stored in the battery and, therefore, their energy efficiency

is important. On the other hand, computers in data centers are limited by the amount of energy that can be supplied and distributed for the ultra large scale calculations with modern processors. Furthermore, energy consumption directly impacts temperature to the creation of hotspots as well as frequent and significant fluctuations in temperature can have direct ramification on the circuity reliability.

Logic transformations are changes in the structure of the circuit and are performed in such a way that the functionality (e.g., relation between the inputs and outputs) are not altered [56]. Logical transformations has been widely used for power and energy minimization. For example, retiming has been used to minimize the energy by minimizing the amount of glitching in the integrated circuit. It has also been used to minimize energy by the reduction of the supply voltage by leveraging the additional slack delay introduced by retiming under delay minimization. We have proposed to use retiming with the use of an objective function where the number of sequential elements is minimized under the constraint of keeping the minimal possible delay using retiming [128]. The intuition behind our objective to minimize the number of flip-flops is to enable the effective usage of multiple supply voltages while reducing the energy and area overhead impact of introducing additional level converters in the circuit. For example, we initially assign all flip-flops at higher supply voltages and combinational gates are then structured in such a way that they receive inputs from either flip-flops and/or other gates that are in a higher or equal supply voltage. The requirement for the voltage level converters are also significantly reduced, since fewer cells initially are required to be set at higher supply voltages. The requirement of level converters is at most equal to the amount neccessary to drive cells that are initially set at higher suppy voltages. Therefore, it is important that the width of the circuit is small at the positions immediately after the flip flops, so that fewer gates can be place at higher supply voltages while significantly reducing energy in satisfying the delay constraints.

Thus, in order to create the overall approach, in the first stage, we retime a circuit in such a way that the minimal delay is guaranteed, while we heuristically minimize the total number of the required flip flops. We can achieve two benefits from our retiming: (i) the delay is minimized which can be used to reduce the voltages used in all the gates if timing slack exists; (ii) the assignment of high voltage gates is minimized while delay benefits are maintained. To the best of our knowledge, this is a new formulation of retiming in enabling effective multiple voltage technology. Our approach for assigning multiple voltages is at the gate-level and our cell supply voltage selection strategy is such that we efficiently identify gates to place at higher voltages and is done in such way such to reduce optimization execution time. Therefore, a multiple voltage assignment strategy can be conducted for even a large benchmark circuit with over 100k gates can be completed within reasonable amount of time. The overall approach is generic in a sense that it can be easily further improved by employing circuit unfolding as a pre-processing step and by considering simultaneously multi-supply voltages (muli-V_{DD}) and multi-threshold voltages (multi-V_{TH}) for given gates in the design. In this paper, we focus in only a retiming and multi-V_{DD} approach.

We have presented a motivational example in Figure 8.1. Given the original and post-retimed circuit at the upper left corner that achieves initial delay of 400 ps and requires 20 fJ, the task is to achieve a delay target of 250 ps using dual-V_{DD} assignment while minimizing energy consumption. The figure at the bottom left shows a valid circuit configuration that used a clustered dual-V_{DD} assignment technique and requires a total of 13 cells (4 flip-flops + 9 standard cells) to be placed a higher V_{DD} to achieve the target delay, while consuming 65 fJ. However, if we first conduct a minimum flip-flop retiming strategy on the circuit, and then apply the same clustered dual-V_{DD} approach, as shown in the bottom right figure, then, only 7 cells (2 flip-flops + 5 standard cells), are required to achieve the same target delay. The new configuration consumes only 45 fJ (vs 65 fJ), achieving an

energy reduction of 30%. Therefore, the advantages of applying min-cut + dual-V_{DD} techniques is clear, as it has enabled energy reductions through reduction of the number of flip-flops without impacting timing (as shown by the critical path in top figures), which consequently reduced the number of cells to be placed at higher V_{DD}.

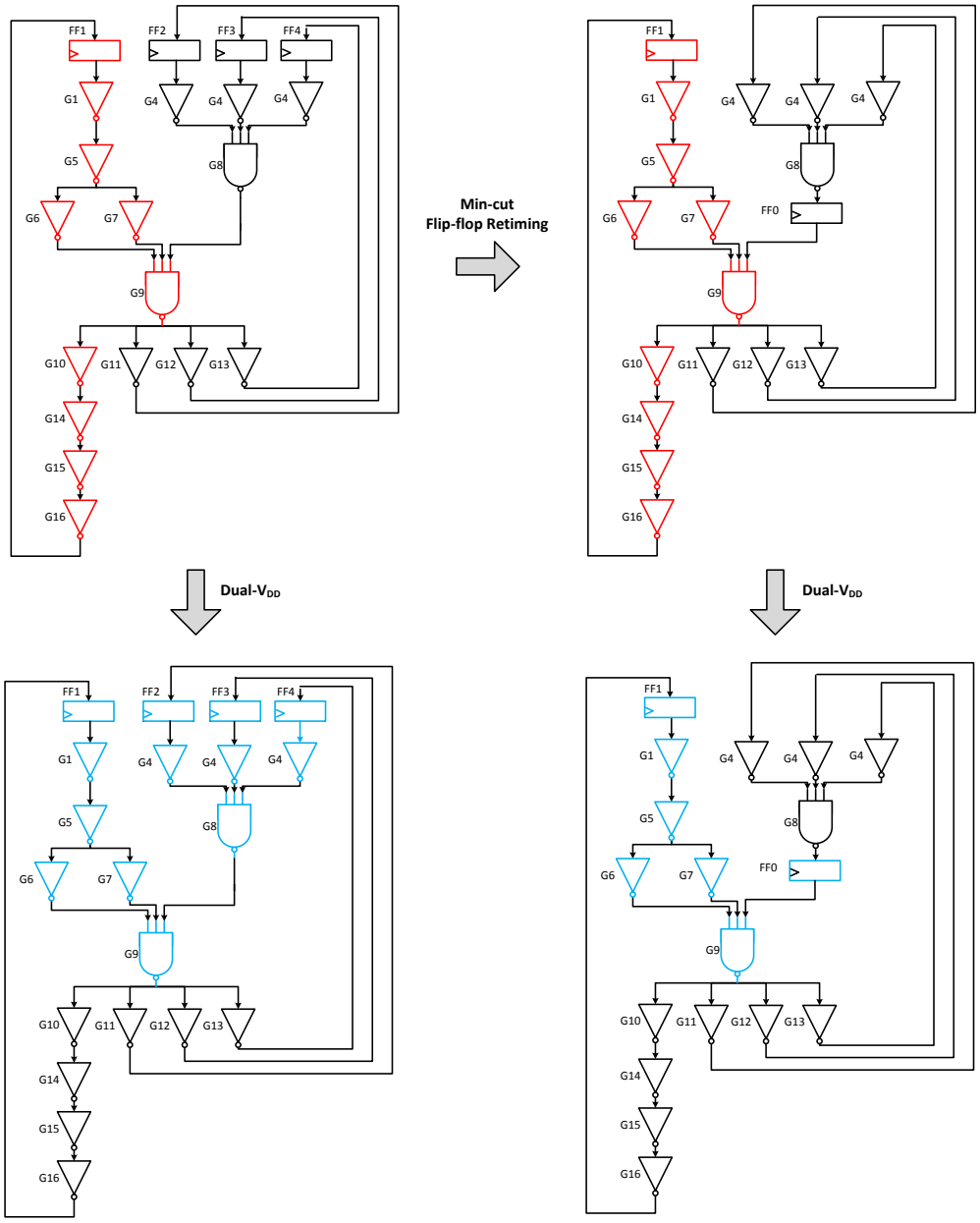
The rest of the paper is organized in the following way. We first cover prior work in Section 7.2 in relation the techniques we propose in this paper. Section 7.3 provides a background on retiming and minimum register (flip-flop). Section 7.5 covers our proposed retiming formulation and is followed by our experimental setup and results in Sections 7.6 and 7.7. Section 7.8 concludes our work.

7.2 Related Work

In this section, we briefly survey the most direct related work, we start by summarizing the most important retiming efforts and logic synthesis and the gate level use of retiming for minimize circuit delay and area. Next, we present published effort for power minimization using retiming. Finally we conclude this section by presenting multiple supply voltage design technology as well as pointing out the difference between the research presented in this paper and the surveyed retiming and dual-voltage technology.

Leiserson and Saxe [128] were first to study retiming and they present polynomial time algorithm that minimize either delay or the number of use of sequential elements. Consequently, several research groups introduce technology that combines the effectiveness of retiming and algebraic transformation technology. Berkeley CAD group [129] introduce a technique for optimizing a sequential network by moving the registers to the boundary of the network using an extension of retiming. [130] and [131] introduce the technology that restructure the circuit in such a way that the retiming bottleneck is removed and significant reduction

Figure 7.1: The orginal post-retimed for delay circuit is shown at the upper left. The red lines represents the critical path in the original circuit. The blue line represents the gate to put in high supply voltage.



in delay time is accomplished.

Mishchenko and his co-authors integrated not only the logic synthesis but also technology mapping and retiming [133]. Soha and Ebeling used conceptual simi-

larities between the retiming and the pipeline to optimize the latency-constrained circuit [134]. Cong and his group at UCLA design technology to combine retiming with the physical planing [132]. [135] developed technology that employ retiming to enable more efficient testability. Kuehlmann and Baumgartner proposed transformation-based verification using generalized retiming [136]. Finally, it is important to mention efforts to create efficient retiming implementation or efforts that use retiming when realistic delay are employed [137][138][139].

Two types of efforts have been proposed to optimize power using retiming. The MIT research group presents a retiming method that targets the power dissipation of a sequential circuit[140]. Princeton CAD group propose the reduction of dynamic power by employing the retiming for delay reduction and consequently voltage scaling[141]. Sapatnekar and his group at UMN developed a series of algorithms that minimize the number of used flip flips in the area of the circuit by using retiming and the corresponding delay models. They also develop an algorithm that simplify the structure of the circuit to minimize the run time of the retiming algorithm [142][143].

Kimiyoshi Usami was the first to propose the use of multiple supply voltage as a way to reduce energy[144][145]. Salil and Sarrafzadeh have applied multiple supply voltages at behaviour level for energy minimization[146]. A gate level approach was presented by Chi et al., to minimize power under timing constraints [156]. Dynamic programming technique for solving the multiple supply voltage scheduling problem in both non-pipelined and functionally pipelined data-paths is proposed in [147]. [148] and [150] have developed the low power technology on the FPGA platform using dual-Vdd/dual-Vt fabrics. [149] has proposed the level converter required for dual-vdd systems. [151] introduce technology that minimize both switching and static power using simultaneous supply and voltage assignment. Chang and his group study the use of dual-vdd by consider the requirement for power-network planning[152]. Finally, [153] has demonstrated

the effectiveness using dual subthreshold supply voltage for energy minimization in CMOS circuits.

To the best of our knowledge, we report the first approach that combines retiming and the dual-voltage assignment for the reduction of energy. The approach employs a new technology that minimize simultaneously both the delay and the number of the flip flops used by retiming. Our dual-vdd assignment only focus on the situation that voltage converters are not required for combinational logic. Under such assumption, our algorithm guarantees significant energy minimization using dual-vdd assignment. The most important observation is that retiming is both used for power minimization as well as to enable a sequence of application of dual-vdd assignment. The effectiveness of our approach in this generalized technology can be further combined with unfolding and other sequential and combinational transformation.

7.3 Retiming and Minimum Register

First proposed by Leiserson and Saxe, retiming is a powerful circuit transformation technique that involves the restructuring of a circuits sequential elements (e.g., D-type flip-flops, latches) in order to minimize the delay in such a way that the output functionality is preserve. Other problem variants have also been proposed such as finding the minimal number of flip-flops, fan-out optimization, etc. A given sequential circuit can be represented as a multi-graph $G = (V, E, d, e)$, where V and E represent the set of vertices v with weighted propagation delays $d(v)$, and edges e with weighted register count edges $w(e)$. Solving retiming requires the evaluation of two matrices D and W where $D_{u,v} \in D$ Retiming is often used to obtain the minimal delay of a circuit and a method solvable in $O(|V||E| + |V|^2 lg(V))$ is shown in [128].

Achieving the minimal number of registers was also explored in [128] which

have been conducted for area and energy (power) optimization. Finding the minimum number of flip-flops can be solved using a maximum-flow/minimum-cut method, where the minimum weighted edge cut from node u to v is equal to the maximum flow of the graph network.

7.4 Work Flow

We show our work flow in Figure 7.2. We start from the characterisation using the cell library, afterwards, we use the gate-level simulation to quantify the delay, switching power, and leakage power, etc. properties about the circuits. The next step we do is to retime the circuit to achieve the minimal delay, based on the retimed result, we apply our minimal-cut to minimize the number of flip flops for further dual-vdd assignment. Finally, we use our dual-vdd algorithm on the current circuit configuration to achieve the final results. We use the delay of the original circuit as our target delay, hence, in each step, we compare the energy consumption given the target delay.

7.5 Technical Approach

In this section, we summarize our contribution on two algorithms, respectively RTMF and Dual-vdd optimization.

7.5.1 RTMF

The pseudocode for the Retiming for minimum flip flops (RTMF) is illustrated in Algorithm 2. The core idea is to apply the minimum-cut retiming on the circuit, meanwhile not to increase the circuit delay. Note that here the critical path delay we want to reserve is the delay after the optimal retiming. Therefore, we only use the minimum-cut retiming on the flip flops that will not influence the circuit

Algorithm 2 Retiming and Minimum flip flops Optimization (RTMF)

Input: C_0 - original circuit.

Input: CP_0 - critical path on C_0.

Input: FF_0 - flip flops on C_0.

Input: FF_{fix} - is a vector that contains all the flip flops that are fixed.

1: $FF_{fix} = \varnothing$
2: $(C_1, CP_1, FF_1) = Retime((C_0, CP_0, FF_0))$
3: **repeat**
4: **for all** ff_i in FF_1 **do**
5: **if** ff_i is in CP_1 **then**
6: FF_{fix}.append(ff_i)
7: **end if**
8: **end for**
9: $C_{pre} = C_1$
10: $(C_1, CP_1, FF_1) = Min - cut((C_1, CP_1, FF_1 - FF_{fix}))$
11: **until** $CP_1 ! = CP_{pre}$
12: Output: C_1

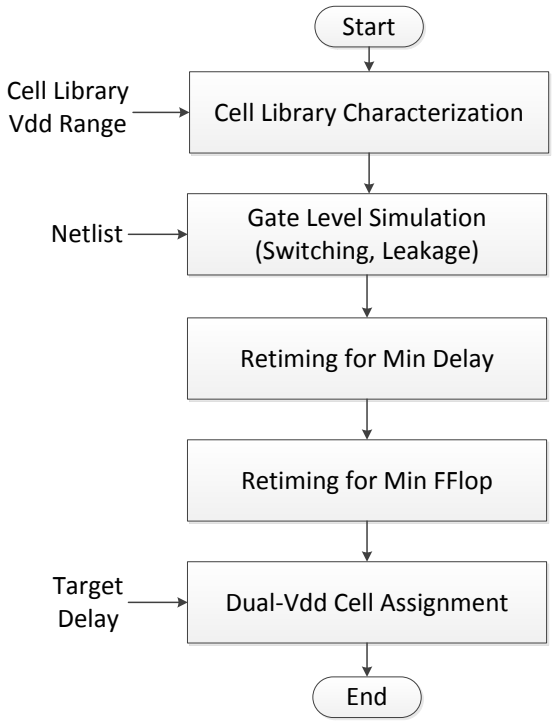


Figure 7.2: Overall work flow of our simulation.

delay. The way to find such flip flops is that after retiming, we fix the flip flops in the critical path (fix means not to change the positions of the flip flops in the procedure of RTMF), apply the RTMF and check whether we have a new critical path. If yes, we go back to the original circuit design after retiming and add the flip flops in the new critical path to our set of fixed flip flops, afterwards, we apply the minimum-cut again. If not, we stop our algorithm and use the current configuration. By using our algorithm, we guarantee that the circuit delay after retiming is not influenced by the algorithm of minimum-cut. The overall approach targets at retrieving the optimal minimum-cut on the circuit in the premise not to change the circuit delay. Starting from this, fewer flip flops and gates need to be put in high voltages.

Algorithm 3 Dual-vdd Optimization (DV)

Input: C - original circuit.

Input: Vdd_0 - original supply voltage on C_0.

Vec_{high} is a vector that contains the gates with high vdd.

Vec_{low} is a vector that contains the gates with low vdd.

1: $Vec_{high}= \varnothing$.
2: $Vec_{low}=$ all gates in C_0.
3: $Vdd_{high} = Vdd_{low} = Vdd_0$
4: **repeat**
5: $CP_0 = CriticalPath(C, Vdd_{low}, Vdd_{high}, Vec_{low}, Vec_{high})$
6: $POW_0 = Power(C, Vdd_{low}, Vdd_{high}, Vec_{low}, Vec_{high})$
7: **for all** $Gate_i$ in CP_0 **do**
8: Vec_{high}.append($Gate_i$ with smallest ASAP)
9: Vec_{low}.erase($Gate_i$ with smallest ASAP)
10: **end for**
11: **for all** possible $Vdd_{newlow} \leq Vdd_0$ **do**
12: binary search $Vdd_{newhigh}$
13: so that $CP_0 = CriticalPath(C, Vdd_{newlow}, Vdd_{newhigh}, Vec_{low}, Vec_{high})$
14: $POW_1 = Power(C, Vdd_{lownew}, Vdd_{highnew}, Vec_{low}, Vec_{high})$
15: **if** $POW_1 < POW_0$ **then**
16: $Vdd_{low} = Vdd_{newlow}$
17: $Vdd_{high} = Vdd_{newhigh}$
18: $POW_0 = POW_1$
19: **end if**
20: **end for**
21: **until** $Vec_{low}!= \varnothing$
22: Output: C

7.5.2 Dual-vdd Optimization

When applying dual supply voltages to the circuit, two essential questions need to be answered. One is what voltages should be used as the multiple voltage combination, the other is which part of the circuit should be put in high voltage (or low voltage). Starting from the two questions, we have proposed an algorithm shown in Algorithm 3 to heuristically approximate the best voltage pairs and the corresponding coverage. We assume that in our design, only the gates with high supply voltage can drive the gates with low voltage, thus we do not need to use the level converters in our circuit. In each iteration, we choose one gate in current critical path with the smallest ASAP and put it in the group of gates with high supply voltage. Afterwards, given the current circuit configuration, we use the binary search to traverse the pairs of voltages that meet the given target delay and find the pair that can achieve the smallest power consumption. The iteration continues until all the gates in the circuit are in the group of high voltage. From there, we choose the lowest point of energy consumption from all the history iterations. In real practice, the minimal energy is normally achieved when only a small subset of gates are put in high voltage, therefore, the algorithm can be stopped when no more energy can be reduced within some number of iterations. The algorithm provides a heuristic way to approximate the best pair of supply voltages as well as the corresponding circuit configuration to achieve the minimal energy consumption.

Figure 7.3 depicts an example on the performance of our algorithm. The tested circuit DMA has the following initial configuration: the total number of gates is 25301, the initial supply voltage is 0.7, the critical path delay is 10737ps, and the power is 52337μw. In each iteration, one gate is switched from low voltage to high voltage. To read from Figure 7.3, the minimal power is achieved at the 3217 iteration, with high vdd=0.75, low vdd=0.60. A detailed data analysis indicates that the power (or energy) by the 3217 iteration achieves 22.69% reduction. The

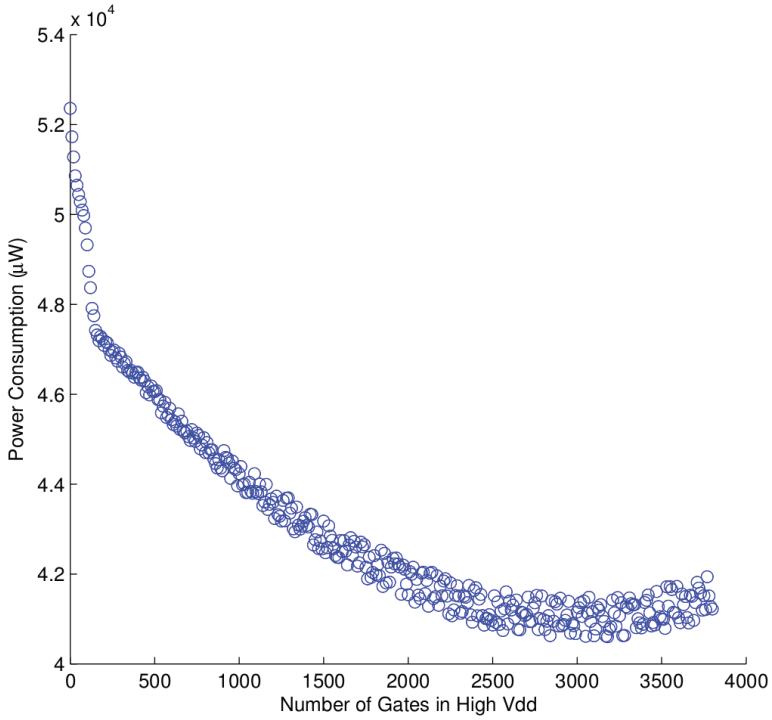


Figure 7.3: An example of the performance of the dual-voltage optimization algorithm on the DMA circuit.

reason that the initial part of the iteration causes more energy reduction is that as more gates being put on high voltage, the circuit is getting balanced, thus the effect on the energy reduction using dual-voltage is reduced. Note that in the whole process, the delay of the circuit is not changed, only the voltages are scaled to meet the delay constrains.

7.6 Experiment Setup

We adopt the ISPD2012 standard cell library [155] and fit accordingly to Markovic's EKV formulation for enabling dual-V_{DD} optimization [154]. The nominal V_{DD} is set to 0.7V and V_{TH} 0.3V. For our dual-V_{DD} approach, we consider V_{DD} within the range of 0.35V to 0.70V. For the scaled voltage approach, we enable the 1mV precision in V_{DD} scaling, where as 50mV precision for our dual-V_{DD} technique.

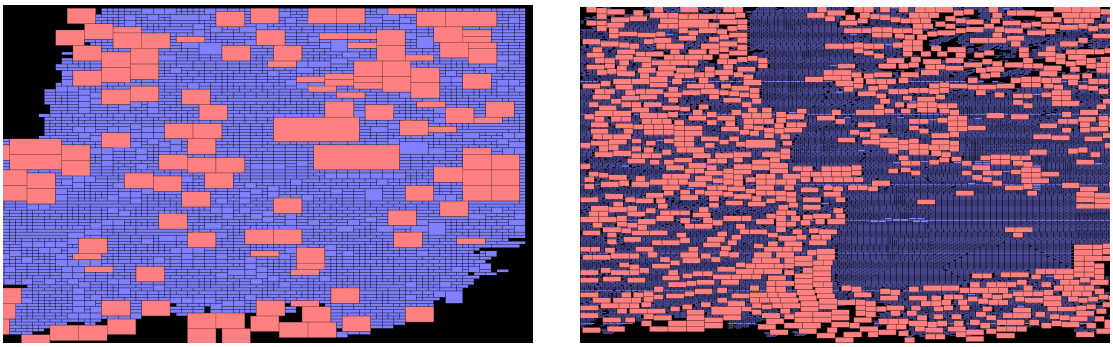
We evaluate a subset of ISCAS89 benchmarks and synthesize each netlist using Cadence Encounter for generating parasitics capacitances for all considered netlists. We developed an in-house timer in C++ for flexibilty and robustness in computing load and slew dependent delays and correlate to an the Synopsys PrimeTime Compiler to be within 1% in both timing and leakage computation. Each design was optimized in accordance to the ISPD2012 design contest suite in satisfying each slew and cell load restrictions.

7.7 Results

Table 7.1: Energy savings when using standard retiming (RT), minimum flip-flop (MF), and dual-V_{DD} (DV). Energy consumption is relative to each method satisfying a target delay achieved by the original configuration operating at the nominal supply voltage V_{DD} (0.7V). Also presented are the scaled supply voltages using RTMF in column 6 and the best dual-V_{DD} pair (RTMF+DV) in columns 7 and 8. The bottom table provides a max, avg, and min summary of the energy savings when applying RTMF (column 1) and RTMF+DV (column 2). Column 3 summarizes the additional savings achieved our dual-V_{DD} approach over RTMF.

Circuit	Delay	Energy (uJ)			Supply Voltage (V)			Energy Savings (%)	
	(ns)	Original	RTMF	RTMF+DV	Scaled	(V_{min}, V_{max})		RTMF	RTMF+DV
s27a	0.44	0.04	0.04	0.03	0.60V	0.55V	0.65V	5.30 %	19.57 %
s208a	0.86	0.52	0.33	0.27	0.48V	0.40V	0.55V	37.47 %	49.27 %
s382	0.74	1.01	0.72	0.55	0.56V	0.45V	0.60V	29.49 %	45.38 %
s386a	1.42	1.53	1.44	1.41	0.65V	0.60V	0.70V	6.06 %	8.18 %
s400	1.52	1.63	0.76	0.70	0.43V	0.35V	0.45V	53.18 %	56.72 %
s420a	1.13	1.25	0.61	0.53	0.41V	0.35V	0.55V	51.37 %	57.56 %
s510	0.82	1.12	1.02	0.93	0.61V	0.50V	0.65V	9.46 %	16.69 %
s526a	0.66	1.33	1.09	0.77	0.58V	0.45V	0.60V	18.12 %	41.96 %
s641	4.54	6.91	5.04	4.07	0.54V	0.35V	0.60V	27.08 %	41.17 %
s713	4.78	7.25	5.00	4.04	0.52V	0.35V	0.60V	31.07 %	44.34 %
s820	1.26	2.54	2.34	2.10	0.61V	0.35V	0.65V	7.72 %	17.00 %
s832	1.26	2.57	2.32	2.11	0.61V	0.35V	0.65V	9.50 %	17.90 %
s953	0.99	2.20	2.08	1.87	0.65V	0.55V	0.65V	5.07 %	14.89 %
s1423	5.69	8.45	7.15	5.26	0.56V	0.35V	0.60V	15.35 %	37.71 %
s5378	1.48	21.81	16.61	15.20	0.58V	0.55V	0.65V	23.84 %	30.31 %
s9234	3.99	13.36	9.63	8.19	0.57V	0.45V	0.60V	27.92 %	38.70 %
s35932	1.86	10.06	9.45	7.65	0.64V	0.55V	0.70V	6.06 %	23.96 %
s38417	3.04	20.40	19.20	15.17	0.64V	0.50V	0.70V	5.88 %	25.64 %
s38584	3.63	28.00	25.40	21.33	0.61V	0.50V	0.65V	9.29 %	23.82 %

	RTMF %	RTMF+DV %	+RTMF %
Max	53.18 %	57.56 %	23.84 %
Avg	22.00 %	33.24 %	11.24 %
Min	5.07%	8.18 %	2.12 %



(a) (b)

Figure 7.4: Circuit layout after applying the dual-voltage (RTMF+DV). The red cells are the gates/flip-flops in high supply voltage and the ble cells are the ones in low supply voltage for benchmarks s5378 (a) and s35932 (b)

We present results comparing the energy consumption against the original circuit under identical timing constraints (Table 7.1). Energy savings are presented in an incremental fashion with respect to the achieved energy and delay of the original circuit operating at the nominal set supply voltage of 0.7V. The techniques we consider independently are minimum delay optimization through retiming and minimum flip-flop minimization with supply voltage (V_{DD}) scaling (RTMF) such that the final delay after voltage scaling satisfies the delay target (e.g., $D_{current} \leq D_{target}$). Finally, we evaluate our approach which combines minimum retiming and minimal flip-flop with our dual-V_{DD} technique (RTMF+DV).

Table 7.1 presents all results in relative terms to the original circuit's achieved energy and delay. The first result is achieved through first retiming the original circuit and then conducting a maximum-flow/minimum-cut algorithm for flip-flop minimization. Retiming involves re-arranging the sequential elements (e.g., fflops, latches, etc.) of the circuit to achieve the minimum delay while preserving the functionality of the design. We observed that between 6.25% to 50% delay reduction (32% avg.) was achieved when conducting RT alone. A major drawback of retiming is the significant flip-flop overhead costs needed in order to achieved

Table 7.2: Impact of retiming on circuit Delay and flip-flop count. The bottom table includes a summary of the savings in delay (max, avg, min) achieved through RT only (column 1) and savings in the number of flip-flops through RTMF over RT (column 2).

Circuit	(%) Delay Savings	No. of FFlops			(%) FFlop Savings
		Orig.	RT	RTMF	
s27a	44.44 %	3	3	3	0.00 %
s208a	46.67 %	8	17	17	0.00 %
s382	50.00 %	21	65	54	16.92 %
s386a	12.50 %	6	8	8	0.00 %
s400	57.14 %	21	68	57	16.18 %
s420a	52.63 %	16	42	38	9.52 %
s510	7.69 %	6	8	8	0.00 %
s526a	33.33 %	21	57	51	10.53 %
s641	31.37 %	19	51	51	0.00 %
s713	31.37 %	19	52	43	17.31 %
s820	11.11 %	5	10	8	20.00 %
s832	11.11 %	5	10	8	20.00 %
s838	50.88 %	32	91	73	19.78 %
s953	18.75 %	29	46	46	0.00 %
s1423	31.19 %	74	103	95	7.77 %
s5378	36.36 %	179	435	394	9.32 %
s9234	27.3 %	228	299	253	3.05 %
s35932	5.2 %	1732	2199	2123	3.05 %
s38417	6.7 %	1636	1685	1657	1.67 %
s38584	7.1 %	1452	1486	1466	1.34 %

	RT (delay)	RTMF (fflop)
Max	57.14 %	26.92 %
Avg	32.43 %	9.75 %
Min	6.25 %	0.00 %

reduced delay, requiring up to 73% max 46% additional flip-flop logic (Table 7.2). However, since the optimal delay may not be unique, we additionally conduct a minimum flip-flop procedure (RTMF) to further driver energy costs lower, while ensuring that the target delay is preserved, which saves up to 26% (9%) flip-flop from RT. A final step in RTMF involves reducing the supply voltage of the entire design uniformly across all cells to reclaim slack (increase delay). The minimum energy V_{DD} is presented under RTMF. such that the minimum energy V_{DD} is recorded

Significant energy reductions is achievable through RTMF alone for most benchmarks. As shown in Table 7.1 RTMF achieves betwee 5-53% (22% avg.) energy reduction over the original configuration. As shown, a lower uniform V_{DD} can be set for designs that achieved significant delay improvement from RTMF alone. However, these results assume 1mV scaling precision in V_{DD}. In contrast, limited energy improvements ($< 10\%$) via RTMF can be observed for 6 benchmarks (s27a, s510, s713, s832, s820, and s953) due to the small differences in V_{DD} with respect to the nominal V_{DD} of 0.7V. Reduced V_{DD} indicate minimal delay reductions through RT and potential overhead (e.g., additional flip-flops). Furthermore, RTMF may also improperly balance the fan-out of outgoing edges of flip-flops. Larger load capacitances can significantly impact the circuit critical path. In order to fix load violations, cells are required to be up-sized in order to support imposed design constraints (e.g., load and slew). For this process, we conduct a similar violation procedure as in Li et al. [60]. Without proper consideration of load, improvements via RTMF may be reduced.

Our solution RTMF+DV achieves between 8%-57% (33% avg.) energy reduction over the original design under the same delay target. These savings translate to an additional 11% average (23% max) energy reduction in energy is achieved over RTMF, which indicates the potential benefits of utilizing a dual-V_{DD} platform. Large improvements using RTMF+DV can be attributed by large differ-

ences (V_{maxx}-V_{min}) with respect to the observed singly-scaled V_{DD} in RTMF; this can be observed with 7 benchmarks (s27a, s208a, s382, s526a, s641, and s1423) that achieve an additional 10% over RTMF. Furthermore, the flexibility in assignment non-critical cells at lower V_{DD}, while simultaneously maintaining the circuit delay on critical paths achieves additional improvements; RTMF is limited in this regard, since only a single V_{DD} is enabled and all cells, critical or non-critical, are set to this configuration.

Reduced energy savings are observed for RTMF+DV in some benchmarks where a small delay improvement is achieved through RTMF alone. Additionally, smaller improvements with RTMF+DV over RTMF indicate that a smaller ratio of cells are placed in lower V_{DD} configuration with respect to the scaled V_{DD} set for RTMF, negating potential the benefits through the use of a dual-V_{DD} appoach. For example, although RTMF achieves 51% energy savings, only an additional 2% (53% overall) in energy reduction alone is achieved conducting RTMF+DV for s420a that uses a dual-V_{DD} platform. Furthermore, the delay impact of a single cell increases at lower V_{DD} along the near-threshold regime. Therefore, the ability to apply lower V_{DD} for cells at near-threshold voltage levels is a challenge due it its potential to significantly impact the critical path. In contrast, for instances where minimal slack is achieved through RTMF, our dual-V_{DD} achieves the greatest energy savings. Therefore, our dual-V_{DD} approach is ideal when higher supply voltages are required to achieve higher performance (shorter delay), while also leveraging the available slack from non-critical cells to be set at lower V_{DD}. Additionally, the costs of placing a higher number of cells at higher V_{DD} is reduced by both the simultaneous reduction of flip-flops that are placed at higher V_{DD} and through efficient restructuring of logic cells during our retiming approach, which is performed in such a way enable greater efficency using dual-V_{DD} technology.

Figure 7.4 presents the final standard cell layout using our RTMF+DF on

benchmark circuits s5378 (3523 cells) and s35932 (22977 cells). The red cells include all flip-flops and combinational cells placed at higher V_{DD} and blue corresponds to the lower V_{DD} pair. For benchmark s35932, 39% of the cells were placed at higher V_{DD} and 11% of cells for s5378%. Although we do not consider cell placement optimization in this paper, in addition to energy minimization, applying an RTMF+DV-aware procedure can be used in generic cell placement tool for power grid optimization for ensuring circuit reliability across the power network of a design. Under these scenarios, the location of higher voltage cells directly impacts the placement of high-voltage power rails of a design.

7.8 Summary

We have developed a new approach for energy minimization that employs retiming as an essential step for the consequent application of dual supply voltages. The new retiming approach minimizes the number of sequential elements under the constraint that the retimed circuit has provably minimal delay achievable by any retiming. The minimization of the number of flip-flops is a heuristic measure that maximally reduces the number of gates that require placement on high supply voltage to achieve further reduction of the delay of the pertinent circuits. This delay reduction in turn reduces the energy required by the gates that use low supply voltage. The approach is generic in a sense that can be easily used in standard synthesis flows and can be further extended to cover transformations such as unfolding and other degree of optimization freedom such as multiple thresholds. We have evaluated the new approach on a wide spectrum of circuits and observed energy reduction by 33% on average.

CHAPTER 8

Provably Minimal Energy using Coarse-grained Hardware Adaptation

Both energy and execution speed can be greatly impacted by clock and power gating, nonlinear voltage scaling, and leakage energy. In this chapter, we address the problem of coordinated power gating and dynamic voltage scaling (DVS) to minimize the overall energy consumption of an application under user-specified timing constraints. We prove that a solution provided by our convex programming formulation that uses at most two versions of hardware, where each version uses its own constant voltages, is optimal. Comprehensive evaluation of the new approach demonstrates energy improvements over traditional DVS and DVS and power gating techniques by factors of 1.44X–2.97X and 1.44X–2.82X, respectively.

8.1 Introduction

There is a wide consensus that energy is a premier design and operational metric for various computing systems ranging from data centers and desktop computers to smart phones and sensor networks [167]. However, energy minimization in these modern and pending systems for time sensitive tasks is an increasingly complex and intractable problem because of several interwoven degrees of freedom, such as supply voltage management, leakage currents, and unit clock/power gating. For example, dynamic voltage scaling (DVS) is by itself complex because the various components (e.g. functional units and cache memory) have highly differing energy-

speed of execution trade-offs. Therefore, the overall system execution speed is a nonlinear function with discontinuities.

Additional challenges for energy minimization include the significant overhead (ranging from hundreds to even thousands of clock cycles) induced by voltage and frequency adjustments. Leakage energy alone depends on several factors, including the allocated hardware, supply voltage, processs variation, and temperature [44][88]. It is well known that clock and power gating are very effective energy saving techniques with relatively low time and energy overheads. However, careful power gating and DVS settings are still necessary when considering the entire task schedule, since their aggregate overheads may greatly impact the execution time.

We address the problem of minimizing the total energy required to complete a computational task within a specified allocated time. We have developed a new approach that combines the effectiveness of static power gating and DVS techniques using a convex programming-based procedure. It has been a long standing common wisdom that a single voltage should be used for the execution of a task. It was proven that this strategy is optimal when the energy-speed dependency is convex. Recently, it was demonstrated that when the relationship is non-convex, it is often more advantageous to employ multiple voltages [18]. However, unless there are drastic differences in speed and speed-energy products for a given application, the benefits of the technique are limited. We create these differences by realizing N versions of the pertinent hardware using static power gating.

The essence of the approach is illustrated in Figure 8.1, which shows energy-speed trade-offs for eight versions of hardware used to realize a synthetic application. In Figure 8.1a, there are eight points that correspond to eight power gating options of the available hardware, operating at a single voltage. The base hardware platform corresponds to the maximal-resource hardware allocation 8 (blue hexagram). Allocations 7 (purple pentagram) through 1 (blue square) are allo-

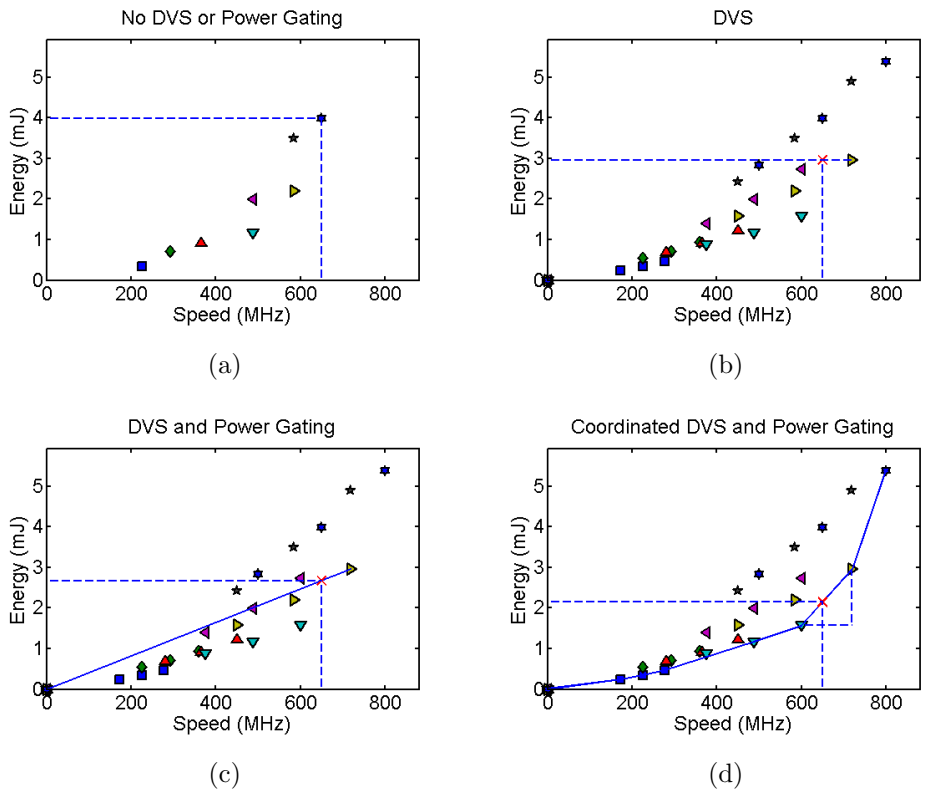


Figure 8.1: Motivating example with 8 hardware allocations at 3 voltages: (a) no DVS or power gating; (b) DVS but no power gating; (c) DVS and power gating; and (d) our approach, coordinated DVS and power gating.

cations with decreasing hardware resources. If our time constraint, for example, requires a clock rate of 650 MHz, the best individual configuration (without utilizing DVS or power gating techniques) is allocation 8 at nominal supply voltage, requiring 4 mJ, as shown in Figure 8.1a.

In Figure 8.1b, there are eight sets of three points that correspond to three operational voltages of each hardware allocation. In this case, for the same timing constraint, the optimal single configuration, where a configuration is comprised of a hardware allocation and operating voltage utilizing DVS, is allocation 6 (yellow right-arrow) set to the highest voltage, consuming 3 mJ (25% savings). Note that the selected configuration is the lowest energy point that exceeds the speed

requirement. Figure 8.1c demonstrates how power gating can be used to reduce the energy consumption by turning off the unused hardware after the completion of the application. This can be done because the speed of the chosen configuration is faster than required. In this example, the energy consumption is now 2.7 mJ (32.5% savings).

Finally, Figure 8.1d demonstrates our new approach that uses coordinated DVS and power gating to provide provably minimal energy. The first step is to find the convex enclosure of the energy-speed points, consisting of piecewise linear segments such that all points are either on the enclosure or above and to the left of it. The intuition behind computing the convex enclosure is that we can use a combination of the two closest points on the enclosure that surround the target speed requirement to execute the task using the minimum amount of energy. In this example, the highest-voltage setting of allocation 6 (yellow right-arrow) can be used for 42% of the required time ($\frac{650-600}{720-600}$), followed by a context switch to allocation 4 (teal down-arrow) for the remaining 58% of the execution time. This approach requires 2.2 mJ (45% savings). Note that in this case changing the voltage between the two optimal points was not required, but in general DVS can be utilized, if needed, to switch to a different voltage.

It is crucial to note that switching between allocations is realized by power gating subsets of microarchitectural units to, for example, reduce the number of units or the cache sizes. Thus, using n sets of power gating circuitry, 2^n hardware allocations can be realized. Our coordinated DVS and power gating procedure is surprisingly effective, since schedules utilizing only two voltages and two versions of an implementation (hardware allocation) are used and can be determined at compile time. There is only one context switch, where either a subset of microarchitectural units are power gated to realize a different allocation, or voltage is scaled, or both. Therefore, the additional storage overhead is small and the overhead for task management is negligible.

To summarize, the overall flow of the optimization proceeds as follows: (1) create the gating structure; (2) characterize the application of interest; (3) calculate the DVS impact; (4) calculate the convex enclosure; and (5) select the two best allocations at the two best voltages (two configurations total).

8.2 Related Work

Dynamic voltage scaling methods have been proposed since more than two decades ago. They address energy minimization by only altering the supply voltages. Those methods cover various sets of scenarios and system specifications ranging from continuous to discrete supply voltages to achieve energy efficiency, but these have mainly focused on dynamic power consumption [46][52]. Researchers have also studied the DVS techniques in conjunction with peripheral management and further generalized the method to multiprocessors [45][53].

Leakage energy consumption and its impact on the overall system energy dissipation have significantly grown because of the continuous scaling of CMOS to miniature feature sizes. In fact, for some 35-nm processes, leakage could potentially be even larger than dynamic energy [44][48][87]. Therefore, a MIT group has studied techniques to address leakage energy, where leakage current is modeled at different levels of abstraction [47]. Furthermore, DVS has been extended to control threshold voltages (V_t) and therefore leakage current through adaptive body biasing [51]. Various methods for simultaneous supply voltage and threshold voltage scaling are also covered [42][5].

Power gating at the microarchitectural level has been proposed to achieve further reductions in leakage, by gating units when idle periods of 10 cycles or more are detected [32]. They achieve significant leakage savings, but only consider one or two units. Furthermore, the impact of leakage current on energy and the usage of various hardware have led to more general and non-convex energy-speed models

[47][49]. Corazao et al., applied template matching to reduce power consumption [54].

Our approach differs from all previous techniques because we assume non-linear trade-offs between energy and speed of execution with an arbitrary number of discontinuities. Furthermore, we show that partial power gating and DVS can be combined in a provably optimal way to minimize energy.

8.3 Preliminaries

In contrast to the energy and delay models presented in Chapter 2, this section presents respective models at the functional unit granularity.

8.3.1 Energy and Delay Model

We adopt the energy and delay models used by Dabiri et al. [18]. For a hardware with M_s resources, switching capacitance is a function of the resources used, $C(M_s)$, where one iteration of the schedule takes R_s clock cycles, and the clock period is $T = \frac{1}{f}$. Since energy is the integral of the instantaneous power over time and each iteration takes R_sT seconds, the total energy ($E_{total(s)}$) of this schedule for one iteration of the schedule is:

$$\begin{aligned} E_{total(s)} &= R_sTP_{total(s)} \\ &= R_sT[\tfrac{1}{2}\alpha CV_{dd}^2 f + V_{dd}I_{sub}] \end{aligned} \tag{8.1}$$

In the above equation, $P_{total(s)}$ is the total power consumption, the sum of dynamic (charging the capacitive load $C(M_s)$) and static (leakage) power, where I_{sub} is the subthreshold current commonly used for approximating leakage. In the optimizations done in this paper, for each schedule we utilize the average energy per clock cycle $\frac{E_{total(s)}}{R_s}$ versus the schedule speed R_s/T for one iteration.

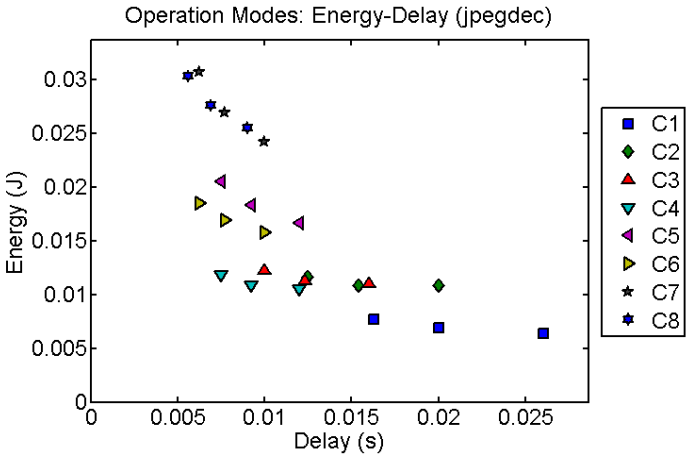


Figure 8.2: Energy-delay points for different configurations (8 allocations and 3 voltages) for the jpegdec benchmark.

The delay of CMOS based processing elements can be approximated as:

$$d = K \times \frac{V_{dd}}{(V_{dd} - V_t)^{\alpha}} \tag{8.2}$$

where K and α are technology-dependent parameters. Equation 8.2 in its general form is used to model delay and its changes with respect to operation frequency and supply voltage.

8.3.2 Multi-Allocation Architecture

Our objective is to minimize the energy consumption per task or collection of tasks, while satisfying the system- and application-imposed constraints, such as latency or throughput. Equation 8.1 suggests that there are potentially two main variables that can be controlled to alter system configurations. One is the hardware capacitance (C), and the other component is the supply voltage (V_{DD}), which leads to DVS techniques.

For the first component, we utilize processing elements called *hardware allocations*, where each allocation effectively has a different $C(M_s)$ (refer to Equation 8.1). Allocations can be individual processing units or can be constructed using

a subset of computational components from a larger computing platform. Note that with n power gating hardware positions, 2^n different hardware allocations can be created.

We are given (or designed) a set of hardware allocations, where each utilize different sets of hardware components. An allocation v_i is identified by a unique tuple, $v_i \equiv (e_i, d_i, s_i, R_i)$, where e_i is the energy per clock cycle of an allocation for a given schedule; d_i is the allocation's delay (latency) for a scheduling cycle; s_i is the allocation's *effective* speed; and R_i is the number of clock cycles to process a given schedule on the allocation.

The second effective variable is the supply voltage (V_{DD}). Energy optimization via dynamic or static voltage scheduling and scaling is a very well studied problem. What distinguishes this paper from previous work in DVS is that our methodology is a hybrid of hardware resource management and voltage scaling. The former corresponds to the fact that we potentially utilize different hardware allocations in the scheduling (via gating), while the latter indicates that each of these allocations can have multiple discrete supply voltage choices.

8.3.3 Configurations

We assume to have M hardware allocations where each can operate under K discrete supply voltages $V = \{V_{dd_1}, ..., V_{dd_K}\}$, which then leads to $N = M \times K$ *configurations*. Each configuration is represented as a hardware allocation-supply voltage pair, $m_i = (v_j, V_{dd_k})$. To each configuration we shall assign an effective speed (computed by Equations 8.3 and 8.2) as well as a value for energy per clock cycle (derived from Equation 8.4), where we represent a configuration as $m_i(v_j, V_{dd_k}, s_i, e_i)$. As can be observed, the configurations have a similar representation as the hardware allocations, which is natural since each hardware allocation is in fact a configuration under a given supply voltage.

Assume a configuration m_x is the fastest configuration in the sense that for a given schedule it requires R_x clock cycles, where $R_x = min(R_1, R_2, ..., R_N)$ and N is the number of configurations. This configuration is called the *base configuration* and its speed s_x is said to be the clock frequency f_{clk}. We normalize other configurations' specifications to the base configuration and define the *effective speed*, s_i, for configuration m_i as follows.

The latency d_i of m_i for a schedule which takes R_i clock cycles is $d_i = R_i/f_{clk}$. Consequently, if we assume that the schedule requires R_x clock cycles but the operational frequency (speed) of the configuration is s_i (not f_{clk}), the equivalent or effective speed of configuration m_i is:

$$s_i = f_{clk} \times R_x/R_i, \tag{8.3}$$

which results in the same delay, d_i. Under this normalization, switching across configurations is virtually equivalent to changing the operation frequency of the configuration under the assumption that the required number of clock cycles (R_i) remains the same. The energy per clock cycle of a configuration is also normalized with R_x, yielding:

$$e_i = \frac{R_i}{R_x} V_{dd} I_{sub} T + \frac{R_i}{R_x} \frac{1}{2} \alpha C V_{dd}^2. \tag{8.4}$$

Figure 8.2 shows energy consumption vs. latency for the jpeg-decoder benchmark. In this graph, we have used 8 hardware allocations, where for each allocation we show three supply voltages. We apply the normalization in Equation 8.3, which leads to energy-speed points shown in Figure 8.3. Details of the configurations (hardware allocation-supply voltage pairs) are described in Section 8.6.

8.4 Problem Formulation

8.4.1 Optimization Objective

Our key objective is to use multiple configurations to process a given task such that the total energy consumption of the system is minimized, while the processing is completed prior to a specified deadline. A scheduling output can be represented as an ordered series of configurations and the length of time each configuration is scheduled for operations: $\Psi =< (m_1, t_1), (m_2, t_2), (m_k, t_k) >$, where $M_r = \{m_1, v_2, ..., m_N\}$ are the N configurations as defined in Section 8.3.3, r is the scheduled task, and the duration for which each configuration m_i is active is t_i. Therefore, the total energy consumption for this schedule and optimization objective can be defined as:

$$E_\Psi = \int_0^T P(\xi(t))dt = \sum_0^R e(s_i)\Delta R \tag{8.5}$$

$$minimize(E_r), s.t. D_r = \sum_{i;v_i \in V_r} t_i \leq T \tag{8.6}$$

where D_r is the processing delay/latency for the schedule Ψ and T is the deadline.

8.4.2 Configuration Switching Overhead

Switching configurations has potentially two sources of overhead: (1) power gating overhead caused by switching across hardware allocations; and (2) voltage scaling overhead. The overhead presents itself in both energy and delay.

8.4.2.1 Power-Gating Overhead

Power gating is done by placing a suitably sized header or footer transistor for a circuit block. The amount of switching energy in the header device (E_{header}) and

the number of cycles needed to power gate a macro before reaching the break-even point ($N_{breakeven}$) are computed as [32]:

$$E_{header} = 2C_{header}V_{dd}^2 \approx 2W_H\frac{1}{2}C_{switching}V_{dd}^2. \tag{8.7}$$

$$N_{breakeven} = 2\frac{1}{2L\alpha}\sqrt{\frac{mV_tW_H}{V_{dd}DIBL}(1+2\frac{C_{supply}}{C_{switching}})}, \tag{8.8}$$

Parameter definitions are presented by Hu et al.; $N_{breakeven}$ is shown to be about 10 clock cycles [32].

Assuming that the ratio of the header device W_H is constant for all the modules, the timing overhead of power gating remains the same for the different schedules. However, the energy overhead is a monotonic function of the number of components in the schedule. Using Equations 8.7 and 8.8, this overhead can be modeled as:

$$\epsilon_{ij} = f(v_i, v_j) \propto |N_i - N_j|, \tag{8.9}$$

$$\delta_{ij} = g(v_i, v_j) = \delta. \tag{8.10}$$

Equation 8.9 indicates that the energy overhead as a result of power gating is a function of the difference in the resources in hardware allocations v_i and v_j, whereas the delay overhead could be assumed to be a constant value.

8.4.2.2 Voltage Scaling Overhead

Switching the supply voltage from V_{dd_i} to V_{dd_j} creates overheads in energy (ϵ_{ij}) and delay (δ_{ij}), which is represented by switching from configuration i to j for

each respective case. We use results from Andrei et al. [42] and Martin et al. [51] to model these overheads as:

$$\epsilon_{ij} = C_r|V_{dd_i} - V_{dd_j}|^2 + C_s|V_{bs_i} - V_{bs_i}|^2 \tag{8.11}$$

$$\delta_{ij} = max(p_{V_{dd}}|V_{dd_i} - V_{dd_j}|, p_{V_{bs}}|V_{bs_i} - V_{bs_i}|) \tag{8.12}$$

where C_r and C_s are constants for power rail capacitance and substrate-well capacitance, respectively. Note that when a configuration switch occurs, potentially both supply voltage V_{DD} and body-bias voltage V_{bs} change. The delay overhead caused by supply voltage and bias voltage changes are proportional to $p_{V_{dd}}$ and $p_{V_{bs}}$, respectively, and the larger delay of these two will be the overall delay overhead.

8.5 Optimal-Energy Scheduling

In this section, we derive a set of properties of our formulation and an optimal N-configuration schedule that leads us to the proposed methodology. This sections extends the work in [18] by incorporating both hardware allocation as well as operating supply voltage.

First, we illustrate methods for emulating any virtual operating speed using a mixture of configurations. Consider a simple example where there are two configurations m_1 and m_2. For the sake of simplicity, we have omitted the configuration switching overheads in this example. Recall that because the optimal solution uses at most two configurations, requiring a maximum of one configuration switch, this overhead is negligible. In order to run the system at speed s^* ($s_1 \leq s^* \leq s_2$) for a given interval $[a, b]$, we first use configuration m_1 in speed s_1 for t_1 seconds and m_2 with s_2 for t_2 seconds, where t_1 and t_2 are:

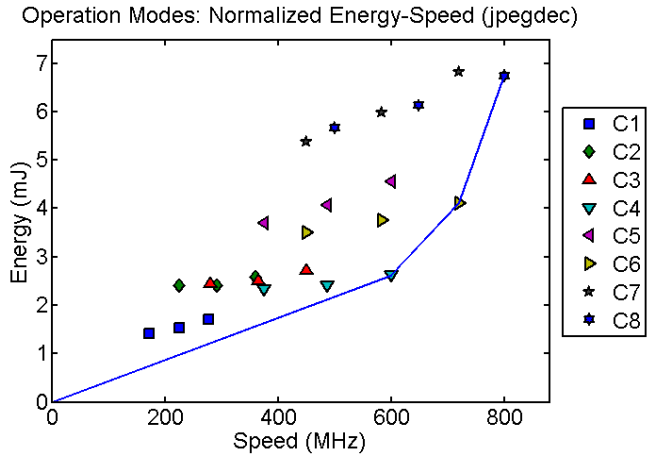


Figure 8.3: Energy-speed points for different configurations for the jpegdec benchmark. Note that the speed is *normalized* as described in Equation 8.3.

$$t_1 = \frac{s_2 - s^*}{s_2 - s_1} \times (b - a), t_2 = \frac{s^* - s_1}{s_2 - s_1} \times (b - a) \tag{8.13}$$

where s^* is the weighted average speed when the system is run at speeds s_1 and s_2, indicating that in the duration of $[a, b]$ the system was operating at the virtual speed of s^*. In the presence of switching overheads, the target speed would be $s^*_\delta = \frac{b-a}{b-a-\delta} s^*$. In this scenario, if $s^*_\delta < s_2$, then:

$$\mathrm{t}_1 = \frac{s_2 - s^*_\delta}{s_2 - s_1} \times (b - a), t_2 = \frac{s^*_\delta - s_1}{s_2 - s_1} \times (b - a) \tag{8.14}$$

Otherwise, $s^*_\delta \geq s_2$ indicates that in order to compensate for switching delay overhead, effective speed should be more than s_2. Therefore, there is no need to use two configurations and the processing can happen only with configuration m_2 for the duration of $b - a$:

$$\mathrm{t}_1 = 0, t_2 = \frac{s^*}{s_2} \times (b - a) \Rightarrow s^* = s_2, t_2 = b \tag{8.15}$$

8.5.1 Convex Energy-Speed Curve

Given two configurations m_i and m_j with effective speeds of s_i and s_j, $\mathcal{E}_{ij}$ is defined to be the the minimum energy per clock cycle for a given virtual speed s^*. Thus, when considering configuration switching overhead, $\mathcal{E'}_{ij}(s^*)$ is simply the minimum energy consumed between its representative virtual speed or by a single configuration, as described by:

$$\mathcal{E}_{ij}(s^*) = \frac{e_j - e_i}{s_j - s_i}(s^* - s_i) + e_i, \tag{8.16}$$

$$\mathcal{E'}_{ij}(s^*) = min(\mathcal{E}_{ij}(s^*) + \Delta e_{ij}, e_i). \tag{8.17}$$

Using this notion of a continuous energy-speed definition, we form a bounding curve on the energy-speed points of the configurations. This curve is convex when switching overhead is ignored and has a convexity property with switching overhead under some conditions which are presented below. Figure 8.3 shows this continuous energy-speed curve, which will be the target operation space. When considering switching overhead, the optimal bounding curve would have vertical shifts (proportional to its energy overhead) and horizontal segments. Horizontal segments are cases where running at a higher speed than s* (e.g. a single configuration) would be more energy efficient than switching between two surrounding configurations due to the overhead. Note that the optimality of the proposed algorithm still holds by just adding the energy (ϵ_{ik}) and speed (δ_{ij}) overheads to the proof of Theorem 8.5.1.

8.5.2 N-Configuration Scheduling Algorithm

Theorem 8.5.1. *There is an optimal N-configuration scheduling where only two or less configurations are used throughout the execution of a task*

Proof. Assume there is an optimal configuration scheduling Ψ which uses k dif-

Table 8.1: Hardware allocation parameters.

Allocation	IDC	LSU	ALU	MUL	L1	L2
1	2	2	2	1	16KB	*none*
2	2	4	2	1	16KB	32KB
3	4	4	4	2	16KB	32KB
4	4	8	4	2	16KB	32KB
5	8	8	8	2	32KB	64KB
6	8	16	8	4	32KB	64KB
7	16	16	8	4	32KB	64KB
8	16	32	16	8	64KB	128KB

Table 8.2: Hardware allocation power dissipation (W) at max V_{DD}.

Allocation	IFU	LSU	MMU	ALU	MUL	L2
1	0.33	0.17	0.05	0.19	0.89	*none*
2	0.33	0.17	0.09	0.19	0.89	0.37
3	0.38	0.22	0.10	0.25	1.19	0.37
4	0.38	0.22	0.10	0.25	1.19	0.37
5	0.63	0.37	0.37	0.82	1.19	0.64
6	0.63	0.37	0.37	0.82	3.85	0.64
7	0.88	0.55	0.91	0.82	3.85	0.64
8	1.02	0.64	0.91	1.48	6.92	1.10

ferent configurations where $k > 2$. Furthermore assume m_1, m_2, and m_3 are three consecutive configurations in terms of energy consumption in the schedule and each runs for a duration of t_1, t_2, and t_3 seconds respectively. We will show that there is another schedule Ψ^* that can be derived from Ψ which uses one less configuration and $E_{\Psi^*} \leq E_{\Psi}$ while $D_{\Psi^*} \leq D_{\Psi}$. We omit the details for brevity but there are two cases to consider in the proof:

- $\mathcal{E}_{13}(s_2) \leq e_2$: This shows that by removing m_2 and only using m_1 and m_3, total energy consumption will be reduced since s_2 can be virtually achieved using equation 9.1, which leads to energy consumption compared to $\mathcal{E}_2$;

- $\mathcal{E}_{13}(s_2) > e_2$: This scenario itself is divided into three cases: $s^* < s_2$, $s^* > s_2$, or $s^* = s_2$, where s^* is the effective speed of the schedule when switching between the three configurations. If $s^* < s_2$, it is trivial that s^* can be created using only m_1 and m_2 with less energy consumption. Similar arguments hold for $s^* > s_2$ and if $s^* = s_2$ and it is evident that only m_2 should have been used for the whole execution of the task.

Therefore, we can reduce the number of configurations in the schedule by one and still use less or equal amount of energy. The same method is applied recursively till at most two configurations remain in the schedule. □

An immediate observation follows that for a given feasible schedule, the average speed of the system, s^* has a lower bound of R_x/T where T is the schedule deadline and R is the required number of clock cycles when the fastest configuration is used. We call this speed the *critical speed*. The idea behind the methodology is to utilize the maximum slack available and run the system at the lowest speed possible, s^*, to minimize the energy consumption. Algorithm 4 summarizes the optimal scheduling for one task. For multiple tasks, we use the results from [3] and apply the same scheduling with the observation that for a critical speed, s^*, the schedule and configurations are found using Algorithm 4. Note that the scheduling is the implicit result of Algorithm 4.

Algorithm 4 N-Configuration Scheduling: Single Task.

1: Find the critical speed: $s^* = R_x/T$;

2: Find i and j such that $\mathcal{E'}_{ij}(s^*)$ is minimized (binary search);

3: Use Equation 8.15 to find the configuration and schedule times for m_i and m_j;

Theorem 8.5.2. *Algorithm 4 results in the minimum energy consumption per frame while meeting all hard deadlines.*

Proof. From 8.5.1 we conclude that at most two configurations are needed to find the minimum energy consumption. Also, from step 2 of the Algorithm 4, the minimality of energy consumption for 2-configuration schedules is guaranteed. □

8.6 Experimental Results

We used the SimpleScalar-ARM simulator [40] to generate single-threaded ARM7-ISA cycle accurate traces. Resources considered are: 1) instruction fetch units (IFU); 2) load-store units (LSU); 3) arithmetic logic units (ALU); 4) multipliers (MUL); and 5) level 1 and 2 instruction/data caches (L1/L2) (Table 9.1). Our power model uses 45 nm parameters included in McPAT [50]. We extract the power values for each modeled resource for each configuration (Table 9.1). Configuration energy (ϵ_{ij}) and delay (δ_{ij}) overheads are computed using representative functional unit load, rail, and substrate capacitances using equations in Section 8.4. We cover eight hardware allocations and enable five discrete supply voltages (0.7, 0.8, 0.9, 1.0, and 1.1V). Due to space constraints, Table 8.2 only lists the hardware allocations' combined dynamic and static powers at the maximum supply voltage. We update the Wattch [43] model with these values to generate total energy and runtime values for each benchmark at each configuration.

We performed our evaluations on 13 different benchmarks, considering 40 configurations comprised of 8 hardware allocations and 5 different supply voltages per allocation. We compare our optimal coordinated DVS and power gating approach against a) DVS alone and b) DVS and power gating. Recall that both of these approaches use only a single allocation and voltage, while our approach uses up to two allocations and two voltages (two configurations) that achieve the optimal

energy for a given delay constraint.

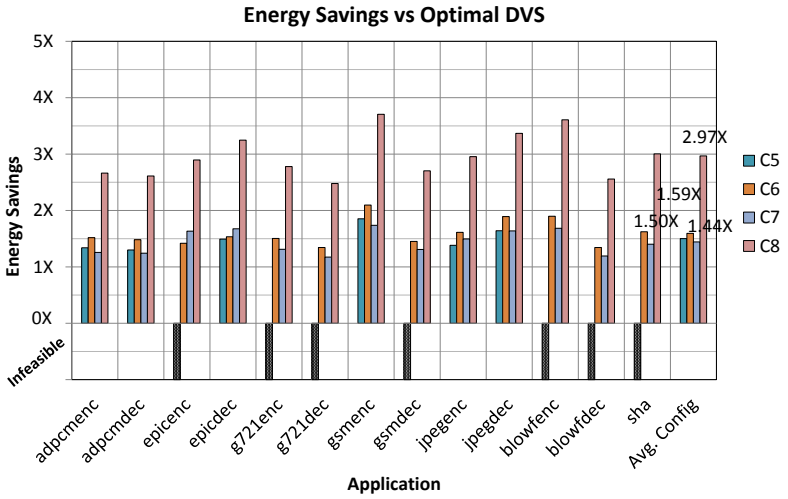
Figures 8.4a and 8.4b illustrate the energy savings achieved in comparison to these scenarios. The x-axis represents the application and the y-axis is the normalized energy savings. For each application, there are 8 columns corresponding to the different allocations. For applications epicenc, g721enc, g721dec, gsmdec, blowfenc, blowfdec, and sha, allocation C5 could not be used to execute the application by the given deadline. In this case, a black bar is shown below the axis to indicate that the base case (either DVS or DVS and power gating) does not have a feasible solution and therefore no energy comparison can be made. Allocations C1-C4 are omitted from the results because they could not be used by the DVS and DVS and power gating approaches to execute any applications within the allotted time. This is to be expected, since these allocations have very limited hardware resources.

Furthermore, Figure 8.5 shows the given deadline and result for our approach for the epicenc application. It is clear from the figure that although hardware allocations C1-C5 cannot meet the deadline independently, the optimal result uses a combination of allocations C6 and C3, whose highest-voltage configurations are the surrounding points of the target speed on the convex enclosure. For each allocation, the DVS-only approach would consume energy equivalent to the energy consumption of the lowest-energy configuration to the right of the target speed. In the DVS and power gating approach, the same configuration would be selected but powered off after completing the task.

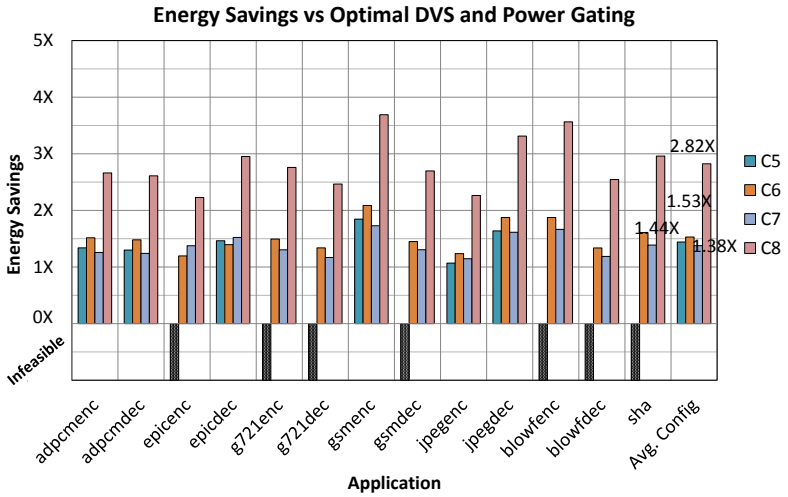
8.7 Summary

We have developed a new approach for energy minimization under timing constraints that combines the effectiveness of power gating and dynamic voltage scaling (DVS) in a coordinated manner. We use a convex programming procedure

to optimally solve the problem without placing any restrictions on the energy-speed of execution relationship or the voltage step values. The technique is highly practical; on standard benchmarks, our method results in an average savings of 1.44X–2.97X and 1.44X–2.82X with respect to the best DVS and DVS and power gating solutions, respectively.



(a)



(b)

Figure 8.4: Energy savings of our coordinated DVS and power gating approach vs. (a) DVS and (b) DVS and power gating. Note that for a subset of applications, hardware allocation C5 is not fast enough to meet the required deadline. Furthermore, allocations C1-C4 are omitted from the results because they could not meet the deadline for any application.

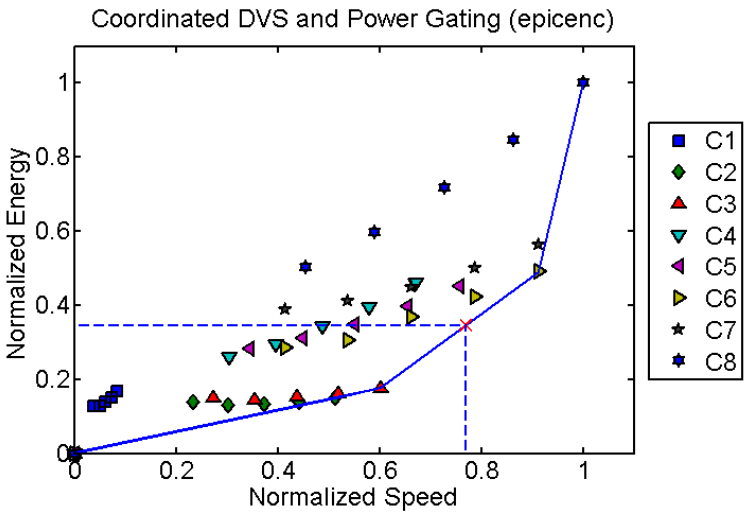


Figure 8.5: Normalized energy consumption for a given delay constraint for the epicenc benchmark using our coordinated DVS and power gating approach.

CHAPTER 9

System Customization and Fine-grained Hardware Adaptation

Customization and adaptation have emerged as the most effective paradigms for energy minimization [161][162][169][170][159][178]. In this chapter, we employ these paradigms to address coordinated power gating and dynamic voltage scaling for energy minimization of real-time tasks in both application-specific and programmable systems and is based from work in [190]. In the customization phase, we determine the optimal hardware configurations to minimize energy for expected applications and workloads in the presence of power gating circuitry overheads using dynamic programming. The first task in adaptation is to determine which hardware allocations and supply voltages should be used to execute a given set of tasks. In the case where tasks induce constant capacitance and are of sufficient length to render power gating and voltage scaling overheads negligible, we obtain the provably optimal solution using a combination of convex enclosure and convex optimization. Our next goal is to relax these two assumptions to consider overheads and non-uniformity in capacitance by subdividing each task into multiple subtasks at a fine granularity. In this case, we use another dynamic programming formulation to find a solution which is optimal in most practical cases. We have also developed a dynamic programming-based approach to minimize overheads of configuration switching.

9.1 Introduction

There is a wide consensus that energy efficiency is a design metric of paramount importance [171][25][26][29][180]. It is, for instance, important for computers in data centers in order to minimize cooling cost that is measured to be in the tens of millions of dollars. It is crucial for mobile wireless devices such as sensor networks, phones, and tablets in particular with a pending transition from voice to data traffic (e.g. IPTV and complex video games) [182][185]. Energy minimization is rarely an easy task but it is even more demanding in systems that are subject to real-time constraints [163][164][179][181].

Therefore, it is not surprising that a tremendous amount of industrial and academic effort has been dedicated to energy minimization, most often under the umbrella of low power research [164][165][172][176][177]. The initial energy models were simple from the optimization point of view and included only switching energy as impacted by supply voltage, frequency, and switching activity. The dependency between the speed of execution and the energy was convex and hence simple for optimization. Finally, initially only strict real-time constraints were considered.

Today, in addition to switching energy, several types of leakage energy are increasingly important. Systems are complex in the sense that the energy of different components scales differently with the supply voltage. Also, the most effective techniques such as power or clock gating result in highly non-linear speed of execution to energy trade-offs. Dynamic voltage scaling techniques for energy minimization now provide rather limited improvements. From one side currently used supply voltages are already greatly reduced as dictated by finer feature sizes. From another side process variation results in some gates having rather high voltage thresholds. Therefore, the gap between the maximal and minimal feasible voltage is small and continues to shrink. As a ramification, the need for emphasis

on different energy minimization mechanisms is well established.

The new most popular applications such as mobile phone calls, WWW surfing, and movie watching on smart phones are subject to new types of real-time constraints. They also have segments with sharply different amounts of parallelism and requirements for different types of resources. Therefore, the important problem of determining ideal hardware configurations and runtime allocations must be addressed in a synergistic fashion across several levels of system abstraction and design phases. For example, identifying and choosing hardware allocations that maximize energy efficiency on given a set of functional units with very different energy-speed trade-offs across a set of expected applications early in design is of crucial importance, since the choices made have significant impact on available runtime configurations later [159][160]. Furthermore, once a set of available system configurations is defined, it is important in the later design stages to identify the suitable time allocations for applications, since they can be fine-tuned to their respective runtime configurations [166][175]. The decisions made in these steps can have dramatic effect on power management schemes in achieving and maintaining quality of service levels [28][173], which are often managed at the operating system level, such as through the widely accepted Advanced Configuration and Power Interface (ACPI) utilizing dynamic voltage scaling and clock and/or power gating schemes [1].

In summary, for modern wireless mobile applications the standard energy minimization techniques have limited effectiveness, and conceptually new energy minimization techniques are required at both the architectural and operating system levels. These techniques are associated with much more complex optimization strategies that are highly non-linear, non-convex, and with a large number of discontinuities [18][181].

9.1.1 Problem Formulation

We focus on one real-time synthesis and one energy management problem that can be informally stated in the following highly simplified way. We start with the low energy synthesis problem. A task and a computational platform are given. The computational platform can be configured in a large number of ways and profiling of the task is conducted for each platform. The goal is to select a user-specified number of configurations such that the task (application) consumes minimal energy while being completed within a user-specified total execution time. No restriction on the speed-energy curve are imposed. Our basic operating system-level problem is to find the allocation of an available time for execution of a given task on each of the allocated hardware configurations, so that the total used energy is minimized.

These two generic problem formulations are the starting point for several more complex and practically important versions. For example, we can address formulations where multiple applications that may be subject to uncertainty are considered. Or we may consider the situation where each task is composed of a number of subtasks with unique speed-energy curves on different hardware configurations.

9.1.2 Objectives and Techniques: Summary

In summary, our goal is to create a hardware platform and accompanying energy minimization techniques that are:

1. *Flexible.* The approach should be such that it can be used for a large number of applications under a wide spectrum of real-time constraints. Hence, we require a very large number of highly diverse and energy efficient hardware configurations. We address this requirement by using two paradigms. The first is power gating of overlapping partitions of a single large hardware platform. The second is creation of virtual configurations by combining two

hardware platforms and assigning the pertinent application to each of them for x and $100 - x$ percent of the available time.

2. *Practical.* Overlapped power gating or other strategies may provide numerous and diverse configurations to the extent that there are too many of them. Hence, often there is a need to drastically reduce their number while preserving their capabilities to realize any of the targeted applications in an energy efficient way. For example, power and/or hardware gating overheads may be very large if there are too many hardware options. More importantly, profiling of a large (e.g. exponential) number of hardware configurations is not practically feasible. We solve this problem by identifying a user-specified number of configurations in such a way that for a specified frequency of a specified set of applications under a specified system of timing constraints we use provably minimum average energy. The technique employs dynamic programming in the speed of execution vs. energy space.

3. *Adaptive.* As we already stated, the effective capacitance and types of required resources in modern application often rapidly change. For example, in many wireless mobile video and audio applications we have periods of high parallelism, where multiplications are often used, interleaved with essentially sequential decision procedures. Hence, it is essential for low energy implementations that hardware allocation and supply voltage are well correlated with the needs of the pertinent application in a particular period of time; this has the potential to impact energy consumption when addressing security primitives [186][187]. Again, we solve this problem using a dynamic programming formulation.

4. *Generic.* It is important that the approach allows easy integration with other techniques for energy reduction. We demonstrate that the new approach satisfies this desideratum by combining optimization using power

gating with dynamic voltage scaling using a simple preprocessing step. It is also important that the approach can be applied at several levels of abstraction. We show this property by using the same dynamic programming-based optimization framework for addressing variable effective capacitance within a single application and for minimization of energy of a set of independent tasks.

5. *Robust.* In some scenarios the same application will be executed multiple times using identical inputs (e.g. a popular movie, song, or WWW page on a popular smart phone). In this situation the effort to profile the scenario accurately is easily amortized over a short period of time. However, in many other scenarios one must use statistical knowledge about the application and inputs (e.g. a sport event on TV or dynamic WWW page with stock information). Now, the detailed and accurate analysis is not an alternative due to high cost and latency constraints. Therefore, robust operation in the presence of uncertainty is very important. We use for this purpose convex programming, which provides the optimal solution to our piecewise linear programming formulation under the often realistic assumption of Gaussian distribution of run-time uncertainty.

6. *Scalable.* All trends indicate that the size of applications and the amount of available hardware resources will continue to increase. Hence, there is or will be a need for the ability to solve very large instances. We address this requirement by providing in addition to our provably optimal dynamic programming solution also a fast convex programming-based approximate solution that does not fully consider the power gating induced overheads.

9.2 Related Work

The emergence of battery-driven mobile communication systems created an impetus for the development of low energy techniques in the very early '90s. Many factors, including technological, integrated circuit, architectural, operating system, and properties of applications impact the selection and creation of the most effective energy optimization techniques and algorithms. Nevertheless, the central role is most often related to the energy vs. speed of execution trade-off.

In the early '90s the feature size of integrated circuits was 1 micron and the supply (V_{DD}) and threshold (V_{TH}) voltages were 5 V and around 1 V, respectively. The high gap between Vdd and Vth and the quadratic dependence between the energy and the supply voltage enabled improvements of more than an order of magnitude and suggested a paradigm where the goal is to transform the pertinent computation in such a way that first the maximal speed-up is created, and consequently it is used for the reduction of the supply voltage and therefore energy minimization [2]. The essential background information is that the switching power (P) is equal to $C_{eff} \cdot V_{dd}^2 \cdot s$, where s is the speed of execution and $s = k \cdot (V_{dd} - V_{th})^2 / V_{dd}$, C_{eff} is effective switching capacitance, and k is a technology and integrated circuit dependent constant. Note that only dynamic (switching) energy was considered because at that time the static (leakage) component was negligible. Therefore, to achieve energy efficiency for a given speed it was found to be beneficial to utilize the lowest supply voltage in periods where there exists high switching capacitance, and vice versa [33]. Algebriac techniques was presented by Potkonjak et al., for minimizing power [55].

Soon it was realized that the same paradigm could be used as a basis for the minimization of energy in a more complex computational model. Yao, Demers, and Shenker considered a set of aperiodic jobs where any point on a continuous convex trade-off energy-speed curve is available and where there is no overhead for

changing the speed [3]. Yao, Demers, and Shenker used a maximally-constrained minimally-constraining optimization paradigm to develop an off-line approximation algorithm that was improved consequently by Bansal et al. [4].

Ishihara and Yasuura considered a more realistic energy-speed model where only a discrete subset of trade-off points is available and under the assumption of a negligible speed change overhead [5]. They proved that in this case the use of two consequent options that bound the required optimal speed produces the optimal solution. Vigorous research efforts along this line eventually resulted in a fully polynomial approximation algorithm with arbitrarily tight approximation [6].

Just before the Ishihara and Yasuura results were published, Hong et al. provided an answer to an important related question of how to use dynamic variable voltage low energy techniques when there is a speed change timing overhead by proving using variational calculus that when there is a speed change overhead it is most beneficial to accomplish the transition between two speeds at the maximal rate if the discrete points belong to a convex curve [7].

At the turn of the century it was realized that the energy-speed relationship is no longer convex due to several reasons including different operational speeds of different components (e.g. memory vs. datapath) and rapidly increasing leakage current in deep submicron technologies. In the current 22 nm technology it forms almost one half of the overall energy consumption. The initial observation was that by powering down one can save a very significant amount of energy. Soon, in addition to active mode, idle, standby, and sleep were considered as shutdown states each with lower energy and longer activation time. Irani et al. have developed a 3-approximation off-line algorithm to minimize dynamic and static energy using as a starting point the algorithm by Yu, Demers, and Shenker [8]. Lee, Reddy, and Krishna introduced the procrastination concept, where the system enters or stays in a shutdown mode even when there are pending tasks in order to

reduce leakage energy [9]. The procrastination algorithms have attracted significant attention [10][11]. From another perspective, Chandrakasan et al. showed that by trading silicon area, power consumption can be reduced while maintaining throughput, for example by replicating hardware and reducing the operating voltage [12].

An in one sense similar yet in another sense opposite concept is slack reclamation, where slack left by completed tasks can be used either for slowdown of other tasks or for staying longer in a shutdown state [13][14]. Furthermore, one important class of energy minimization is one where the optimization has to be conducted under uncertainty. The most popular model is where a probability distribution function (PDF) of execution times is available. For example, Lorch et al. have developed an approach where a task starts at low speed and keeps increasing its speed of execution in accordance with the expected required time for a specific instance [15].

Another important class of energy minimization problems is a set of dual problems to already stated ones, where the goal is to minimize the allocation cost of the hardware platform under energy and/or timing constraints [16][17]. They proposed heuristics and approximation algorithms for this computationally difficult variant of our problems. Recently, Dabiri et al. presented surprising results that in a sense ultimately addressed optimization when the energy-speed trade-off has an arbitrary dependency [18]. Regardless of the shape and form of the dependency, they showed one can always provably optimally minimize the total energy of any task using at most two actual speeds by using computational geometry concepts. The use of N-versions was explored by Alkabani et al., in order to leverage high-level synthesis in the context of energy minimization [19] with temperature dependencies. A low-cost temperature self-sensing system was also presented by Vahdatpour et al. [20]. Multiple supply and/or threshold voltages are another popular set of energy minimization techniques [21]-[24][27][168].

However, both multiple V_{DD}/V_{TH} and dynamic voltage scaling techniques have decreasing efficiency as technology progresses. Two main reasons are that feature scaling drastically reduced not just supply and threshold voltages but also their gap. Also, process variation causes threshold voltage to become subject to a relatively wide distribution, and the threshold voltage must be higher than the highest threshold voltage of any transistor. Therefore, although dynamic voltage scaling is still important, power gating may now enable much better improvements.

In the last decade, power gating was recognized as an effective way to reduce leakage energy [32][34][35][36][37]. Our goal is to provide an impetus for further development of this line of optimization by showing how a simple but rich power gating structure can be adaptively used to facilitate energy minimization in modern and pending systems.

For the sake of brevity and due to space limitations we do not cover energy minimization in distributed systems such as wireless phones, sensor networks, and data centers [2][38][39] and simultaneous thermal management and energy minimization.

9.3 Preliminaries

We begin by describing a critical set of technical preliminaries, including creation of hardware allocations via power gating; detailed profiling of applications; a previously proposed approach for executing a single application at a specified speed with minimal energy; and our simulation platform and energy, delay and configuration switching overhead models. Note that the emphasis here is functional unit-level power modeling and, thus, is a higher-level abstraction than the gate-level models presented in chapters 3, 5, 6.

9.3.1 Multi-Allocation Architecture

In this section, we describe a method of employing power gating techniques to create processing elements called *hardware allocations*. A hardware allocation can be defined as a subset of computational components (e.g. caches, ALUs, registers, etc.) that together form a complete computing platform. Note that there are an exponential number of unique subsets that can be created from such a set of components, so in principle the number of hardware allocations that can be created from a single computing platform is exponential in the number of computational components or functional units. However, there are two repercussions. The first is that the subsets must themselves form completely functioning computing platforms in order to be candidate hardware allocations. Furthermore, the power gating circuitry required to power on and off specific hardware allocations imposes area and energy overheads and therefore the number and types of hardware allocations is limited.

Nevertheless, power gating can be used to construct M hardware allocations $\{m_1, ..., m_M\} \in \Phi$, which can in turn be operated at V discrete supply voltages $\{v1, ..., v_V\} \in \Psi$; each hardware allocation-supply voltage pair constitutes a single hardware configuration $c_i, 1 \leq i \leq M \cdot V$. Consequently, each configuration will execute a given task with a unique speed and energy.

An example power gating scheme for hardware allocation creation is shown Figure 9.1. The figure shows a complete computing platform consisting of 3 instruction fetch units (IFU), 8 execution units (EX), 3 memory management units (MMU), L1 and L2 caches, a phase-locked loop (PLL), voltage regulators (VREG), and a power management unit (PMU). We use power gating to create $M = 3$ hardware allocations, where each shading level corresponds to a single hardware allocation. The lightest shaded components are present in all 3 allocations ($m1$, $m2$, and $m3$) the medium shaded components are present in only the two larger

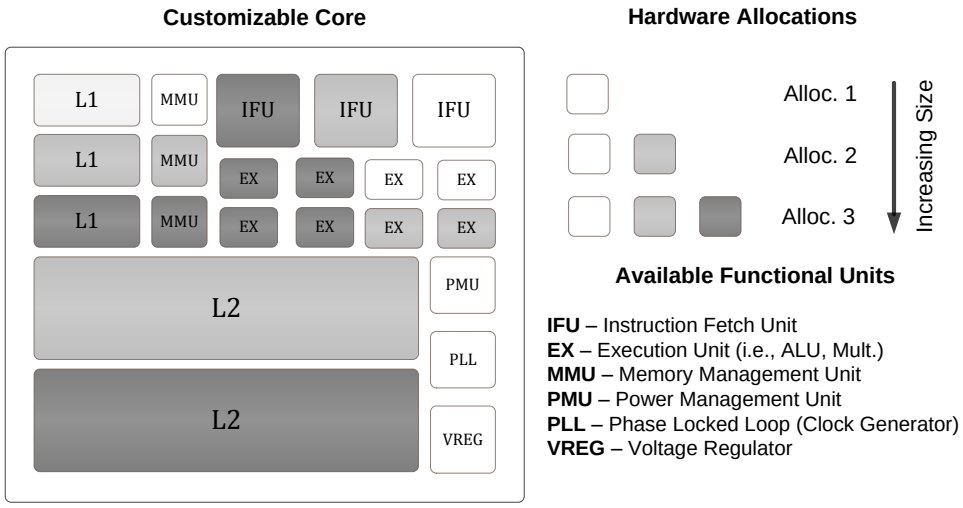


Figure 9.1: Hardware allocation example showing three hardware allocations: 1) *light shading* – smallest allocation; *intermediate shading* – intermediate allocation; *dark shading* – largest allocation.

allocations ($m2$ and $m3$), and the darkest shaded components are present in the largest allocation comprising the entire computing platform ($m3$). For example, $m1$ consists of only 4 EX units, $m2$ consists of 6, and $m3$ contains all 8.

Note that the PLL and VREG are used to achieve desired clock speeds and are managed by the PMU, which is also responsible for coordinating the power gating circuitry; therefore, these components must be present in all allocations. Note also that although only one L1 cache and one L2 cache are physically present, the multiple blocks in the figure represent different sizes of the caches, where a portion of the cache is power gated for smaller allocations.

9.3.2 Application Profiling

The goal of the application profiling step is to capture an application's execution behavior (e.g. instructions per cycle, cache misses, macro/micro operations, etc.) and quantify how it uses available hardware resources (e.g. ALUs, registers,

caches, etc.) as well as how it is affected by its data input parameters (e.g. jpeg, mpeg, audio, etc.). Application profiling provides essential information about an application's achievable performance (instruction-level parallelism) and power consumption (functional unit switching capacitance), enabling key energy and speed trade-offs to be analyzed on both a per-application and a per-hardware allocation basis.

We utilize the cycle-accurate *SimpleScalar* simulator for acquiring per application profiling statistics for each considered hardware allocation [40]. The recorded statistics include the number of instructions, execution cycles, and functional unit activity. As a result, a unique profiling result is generated for each hardware allocation and can be compared across different hardware allocations at identical time frames by their instruction count id. Note that instruction cycles cannot be used since cache sizes, number of functional units, and bus-width, for example, each potentially impacts the total required execution cycles of an application if the hardware allocation was altered during runtime. Thus, we use the instruction id as a synchronization mechanism to compare profiling statistics across various hardware allocations enabled by power gating.

We profile each considered benchmark for each hardware allocation. Profiling at different voltages is not required since voltage scaling primarily impacts the operational speed or clock frequency. Note that more sophisticated profiling must be conducted for an asynchronous processor where the clock rate may vary across functional blocks. However, for our experimental platform we consider a synchronous in-order processor platform that operates at a single global voltage, where all functional units operate at a unified clock rate.

Application profile results can also indicate periods of high application- and hardware-level parallelism, further improving allocation of tasks to available configurations. For example, periods of high instruction throughput can be assigned configurations set at lower voltages, while periods of low throughput can be as-

signed at higher voltages in order to speed up the execution of the bottleneck while minimally impacting energy consumption. A similar approach is found on conventional modern microprocessors by transitioning to *turbo-mode*, which operates by assigning periods of low thread-level parallelism to higher clock frequencies in order to maximize energy efficiency. Our approach differs since we alter both the hardware platform via power gating and speed via DVS techniques.

The functional unit activity for each profile can then be processed through an event-driven power simulator such as *Wattch* in order to generate energy results for each hardware allocation setting [43]. We extract the energy and timing values at a fine subtask granularity. The minimum task size (or subtask) should be sized in accordance to the minimum configuration switching overhead break-even point; for our experiments, we set the minimum subtask granularity to be 10 instructions. Profiling at smaller granularities would incur long simulation run times. However, in many cases fine granularity profiling can be performed since it must be done only once for each considered application, input set, and hardware allocation and is done off-line.

9.3.3 Single Task Allocation

Dabiri et al. described a technique for using at most two configurations to achieve any *virtual* operating speed [18]. Consider a simple example where there are two configurations c_1 and c_2. To run the system at virtual speed s^* ($s_1 \leq s^* \leq s_2$) for a given interval $[a, b]$, where s_1 and s_2 are the speeds of configurations c_1 and c_2, respectively, for the given task (obtained from profiling), we use configuration c_1 for t_1 seconds and c_2 for t_2 seconds. We calculate t_1 and t_2 as:

$$t_1 = \frac{s_2 - s^*}{s_2 - s_1} \times (b - a), t_2 = \frac{s^* - s_1}{s_2 - s_1} \times (b - a) \tag{9.1}$$

where s^* is the weighted average speed when the system is run at speeds s_1 and

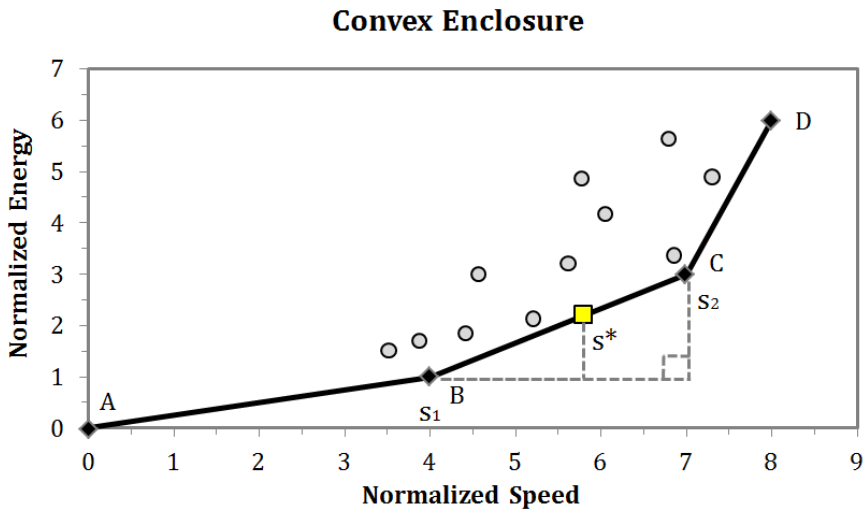


Figure 9.2: Convex enclosure comprising configurations A, B, C, and D over all configurations. The virtual speed s^* can be achieved by utilizing speeds s_1 and s_2 under configurations B and C, respectively.

s_2, indicating that in the duration of $[a, b]$ the system was operating at the virtual speed of s^*. The key result is that at most two configurations on the *convex enclosure* of the speed-energy trade-off curve of available configurations can be used to obtain any virtual speed of execution using provably minimal energy. The convex enclosure can be defined as the convex, piecewise linear curve connecting points such that all points are either on the enclosure or above and to the left of it.

Figure 9.2 provides an example of a convex enclosure for an example task with 15 candidate configurations. *Black diamond* points (A, B, C, and D) represent configurations comprising the convex enclosure. An optimal virtual speed s^* can be achieved by executing the task on at most two configurations on the convex enclosure that bound that virtual speed. Recall that the durations of execution on either configuration can be calculated using Equation 9.1. The convex enclosure property ensures that no other combination of configurations can execute the task at virtual speed s^* with lower energy. In this example, configuration B would be

Table 9.1: Hardware allocation parameters.

Allocation	IDC	LSU	ALU	MUL	L1	L2
1	2	2	2	1	16KB	*none*
2	2	4	2	1	16KB	32KB
3	4	4	4	2	16KB	32KB
4	4	8	4	2	16KB	32KB
5	8	8	8	2	32KB	64KB

used for time t_1 and configuration C for time t_2.

Note that because the optimal solution uses at most two configurations, it requires a maximum of only one configuration switch; therefore, the configuration switching overhead is negligible for reasonable task lengths. However, this approach optimizes for only a single task and is optimal only under the assumption that energy consumption is uniform within a task. However, often a computing platform must execute a variety of expected applications, each of which has different degrees of instruction-level parallelism and functional unit usage throughout execution. Our goal is to provide a technical approach to allocating both time and hardware to execute multiple tasks and subtasks with minimal energy while maintaining execution deadlines in the presence of both uncertainty and configuration-switching overheads.

9.3.4 Simulation Environment and Models

The *SimpleScalar*-ARM cycle-accurate simulator was used to profile application traces for up to 10 million instructions on the hardware allocations shown in Table 9.1 [40]. We used *McPAT* [50] to extract dynamic and leakage power values for each allocation. We construct our power model to support five hardware allocations, with each allocation capable of operating at five discrete supply voltages and frequencies (0.6V at 450 MHz, 0.7 at 600MHz, 0.8V at 850 MHz, 0.9V at 1.0

GHz, and 1.1V at 1.15 GHz). We utilize a *Wattch*-based power model to obtain the total energy consumption values for each application at each configuration by using the recorded application profiling results for each functional unit and applying its respective power cost. It is important to note that the total execution time is required for accounting for leakage energy.

9.3.5 Configuration Switching Overhead

Switching between configurations has potentially two sources of overhead: 1) power gating overhead caused by switching between hardware allocations (pg); and 2) voltage scaling overhead (dvs). Therefore, the total energy overhead $\varepsilon_{i\to j}$ is the sum of the two components $\varepsilon^{pg}_{i\to j}$ and $\varepsilon^{dvs}_{i\to j}$, while the total delay overhead is the maximum between $\delta^{pg}_{i\to j}$ and $\delta^{dvs}_{i\to j}$.

9.3.5.1 Power gating

Power gating has been used as an effective leakage energy saving technique and operates by disconnecting the supply voltage source to the circuit block of interest. We adopt the formulations presented by Hu et al. in accounting for energy and delay transition overheads [32]. We compute the total energy overhead $\varepsilon^{pg}_{i\to j}$ for switching from configuration c_i (with operating voltage v_i and switching capacitance C_i defined by the hardware allocation) to configuration c_j as the amount of energy required to drive the device headers to both power off the blocks present in c_i and power on those in c_j, shown below:

$$\varepsilon^{pg}_{i\to j} = W_H \cdot (C_i \cdot v_i^2 + C_j \cdot v_j^2). \tag{9.2}$$

where W_H represents the ratio of the total area of the header device to the total area of the power-gated component. As in the work by Hu et al., we use the typically quoted ratio $W_H = 0.1$ and assume a constant delay overhead of $\delta^{pg}_{i\to j} = 10$ clock cycles for powering up or down functional units.

9.3.5.2 Dynamic voltage scaling

We adopt energy and time overheads when scaling voltages provided by Burd et al. [41], reproduced below,

$$\varepsilon_{i\to j}^{dvs} = (1-\eta)\cdot C_{reg}\cdot \|v_j^2 - v_i^2\| \quad (9.3)$$

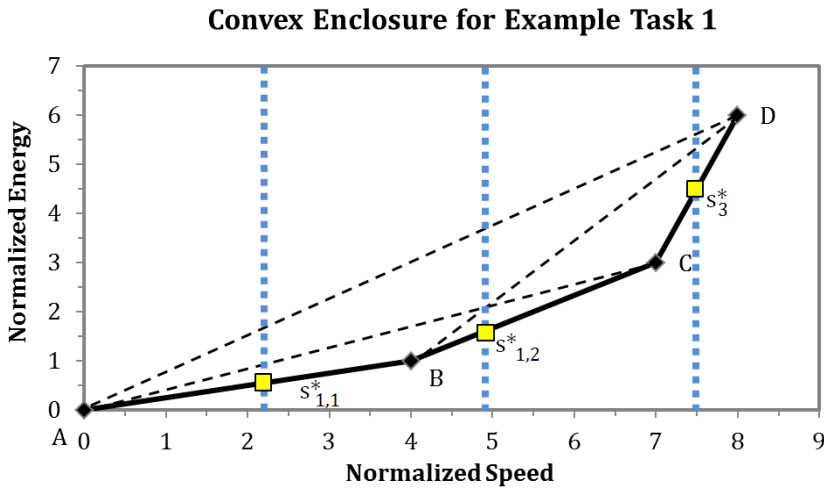
$$\delta_{i\to j}^{dvs} = \frac{2\cdot C_{reg}}{I_{max}}\cdot \|v_j - v_i\| \quad (9.4)$$

where η and C_{reg} are the voltage regular efficiency and load capacitance, respectively, and I_{max} is the circuit's maximum drive current. We adopt the standard values of $C_{reg} = 10$ μF and $I_{max} = 1$ A to estimate delay and energy overheads roughly on the order of tens of μs and μJ, respectively.

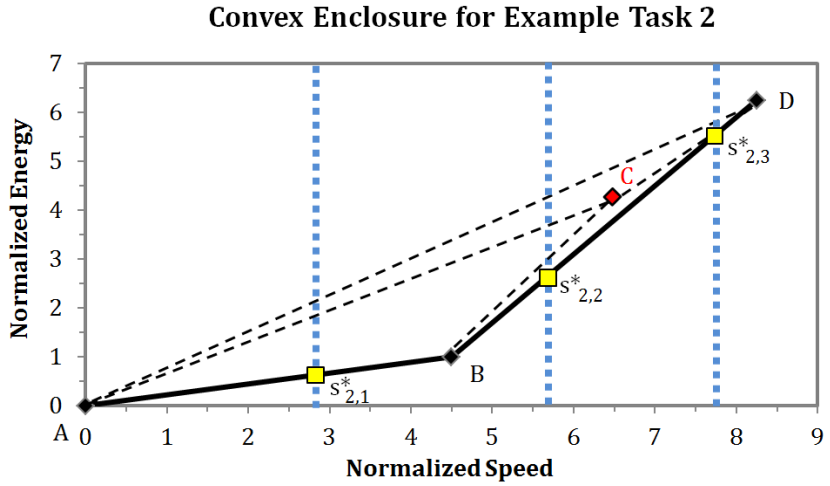
9.4 Customization: Hardware Configuration Selection

The goal of this step is to define a subset of candidate hardware configurations that will be implemented in the final system architecture. Recall that a single configuration consists of a single hardware allocation and supply voltage pair; each hardware allocation is obtained by power gating subsets of microarchitectural components (e.g., functional units), and we can support discrete supply voltages at any arbitrary granularity in our optimization formulation. The key motivation for this problem is that area and energy overheads for power gating circuitry may be too high to support a very large number of configurations. Therefore, limiting the total number of supported configurations while minimizing total expected energy consumption is a crucial goal.

Assumptions. N tasks have been profiled for each of $V \cdot M$ hardware configurations; resulting speed-energy points for each task on each configuration have no uncertainty; the tasks are of sufficient length that configuration switching overheads are negligible; a distribution of runtime deadlines and frequencies for each task are expected; the origin (all hardware powered off) is always a supported



(a) Task type 1



(b) Task type 2

Figure 9.3: Example speed-energy trade-offs for two tasks $t_i = 1$ (a) and $t_i = 2$ (b) with 4 configurations A, B, C, and D. Shown are each task's distinct convex enclosures and expected virtual speed requirements of $s^*_{i,1}$, $s^*_{i,2}$, and $s^*_{i,3}$.

configuration.

Problem Formulation. Our goal is to determine the best K configurations ($2 \leq K \leq V \cdot M$) in terms of minimizing total expected energy consumption while satisfying all expected task deadlines.

For example, consider the case where there is only a single task. In this case, only the configurations that provide points on the convex enclosure should be considered, as any other (virtual) speed can be obtained with lower energy using a combination of at most the 2 surrounding points (actual speeds) on the convex enclosure. For $K = 2$, where only a single configuration (in addition to the origin) will be supported by the hardware, this configuration must provide a speed that will meet any expected deadline for the task. To meet the deadlines while consuming minimal energy, the optimal point is the lowest speed configuration (leftmost point) on the convex enclosure that is faster than (to the right of) the fastest (rightmost) expected required speed. This result is intuitive from the definition of the convex enclosure; this point on the convex enclosure can produce the fastest required speed with the lowest possible energy. The energy consumed will be the energy value for the line connecting the selected configuration and the origin, as the hardware can be power gated once the task is complete.

For $K > 2$, we develop a dynamic programming based approach to find the K configurations that can meet expected deadlines with optimal energy. Again, we assume that the origin and lowest speed configuration that is faster than any expected deadline are 2 included configurations. The approach is described in Algorithm 5. The essence of the algorithm is that if we consider points from left to right, optimal substructure is maintained in the sense that the best K configurations up to configuration c_i will include the solution with minimum *cost* among the solutions for the best $K - 1$ configurations up to all configurations $c_j, j < i$. The cost in this case is the weighted integral between the selected points and the complete convex enclosure up to the fastest included configuration,

Algorithm 5 Configuration Selection

1: **Inputs:**

2: speed-energy points (s_{ij}, e_{ij}), $1 \leq i \leq M$, $1 \leq j \leq N$ for running task t_i at configuration c_j, sorted for each task by increasing speed

3: expected frequencies of occurrence f_i for each task t_i

4: expected probability distribution function $P_i(s^*)$ of expected speed deadlines s^* for each task t_i

5: K, the number of configurations supported by the hardware, $K > 2$ (since the origin and fastest configuration are always included)

6: **Output:**

7: a table O, where a cell o_{ij} represents the cost corresponding to the best j configurations up to the ith configuration (note that the best K configurations can be extracted by bookkeeping while building the table, which is omitted from this description)

weighted by the expected probability of the particular speed requirements. This is again because selecting all points on the convex enclosure represents the provably optimal solution for all deadlines.

The problem becomes more challenging in the general case where there is more than one expected task, but it can still be solved optimally for many practical examples. In this case, attention cannot be restricted to the convex enclosures, because each task may in general have different configurations on their enclosures. Therefore, there are two required modifications to support multiple tasks: i) all points must be considered, not just those on the convex enclosure; and ii) the integral is calculated over all tasks, weighted both by probability of the particular speed requirement as well as expected frequency of the particular task.

Note that configurations may produce points in different orders in terms of speed for different tasks. In other words, if configuration c_i is faster than configu-

ration c_j for task t_a, it may be slower for task t_b, due for example to differences in locality of memory accesses that cause caches of different sizes to produce different trade-offs in speed and energy. This phenomenon results in sub-optimality of our dynamic programming formulation when multiple tasks are considered. However, because it occurs under relatively rare conditions, and mainly for configurations that produce relatively similar speeds of execution, the approach is still highly effective and indeed optimal for many practical examples.

A small example of the configuration selection problem is shown in Figures 9.3a and 9.3b, where two tasks are shown. Each task is expected to have one of three deadlines, resulting in three expected minimum speed requirements ($s^*_{1,1}$, $s^*_{1,2}$, and $s^*_{1,3}$ for task 1, and $s^*_{2,1}$, $s^*_{2,2}$, and $s^*_{3,2}$ for task 2), with equal likelihood. Note that the expected deadlines can follow any arbitrary distribution. Also assume that the two tasks are expected with equal frequency, though this can also be arbitrary. The 4 possible hardware configurations are labeled as A, B, C, and D. Recall that configuration A is the origin and represents the state when the entire system is completely power gated (except for any power management logic). Note that the convex enclosure is different for each task (e.g., all 4 configurations are on the convex enclosure for task 1, but C is *not* on the enclosure for task 2).

Also shown are grey and blue dashed lines that represent the cost of selecting various subsets of configurations. For example, if only configurations A and D are selected, then the mean average of the lengths of the six blue dashed line segments between the convex enclosures (solid) and the grey dashed line connecting A and D is the expected energy cost of selecting this particular subset. If each deadline was expected with different likelihoods, or each task was expected at different frequencies, then the weights of the corresponding line segments must be adjusted accordingly.

9.5 Adaptation: Time Allocation

Once the hardware configurations have been finalized, the next phase is to develop an adaptive time allocation algorithm that allocates tasks or subtasks to particular hardware configurations for particular amounts of time. In this phase, the goal is to minimize total energy by assigning an execution time and hardware configuration to each task or subtask given a global execution deadline. We address time allocation under the following two scenarios: i) detailed task analysis is not practical, in which case tasks can only be subdivided into coarse-grained subtasks or not at all; and ii) accurate profiling of a task is practical, and therefore the task can be split into multiple subtasks at a very fine granularity. In the former case, configuration switching overheads (for power gating and voltage scaling) are negligible in the case where task intervals are significantly longer than the switching overheads. However, in the latter, switching overheads impact the final result due to the finer task scheduling granularity and thus optimality can no longer be guaranteed.

9.5.1 Coarse-Grained Profiling

Assumptions. N tasks (or subtasks) have been profiled for each supported hardware configuration; resulting speed-energy points for each task on each configuration have no uncertainty; each task t_i can be run on C_i configurations $c_{i,1}$, $c_{i,2}$, ..., c_{i,C_i} on its convex enclosure with speeds $s_{i,1}$, $s_{i,2}$, ..., s_{i,C_i}, respectively; the tasks are of sufficient length that configuration switching overheads are negligible; and an overall speed requirement s_{global} is expected.

Problem Formulation. Our key objective is to determine the set of configurations with which to process a given set of tasks such that the total energy consumption of the system is minimized, while completing within a specified deadline. Recall that for a single task t_i, a target speed s_i^* can be achieved optimally

in terms of energy using the method described in Section 9.3.3. Therefore, this problem can be reduced to finding the set of speed requirements s_i^* for each task t_i such that the global speed requirement s_{global} is met with minimal overall energy consumption.

We solve this problem using convex or piecewise linear programming. Each piecewise-linear segment corresponds to the line segment connecting two consecutive configurations on the convex enclosure of a task. The energy consumed by each task can be represented by the equations below:

$$e_i^*(s_i^*) = \begin{cases} f_{i,1}(s_i^*) & s_{i,1} < s_i^* \leq s_{i,2} \\ f_{i,2}(s_i^*) & s_{i,2} < s_i^* \leq s_{i,3} \\ \quad ... & \\ f_{i,C_i-1}(s_i^*) & s_{i,C_i-1} < s_i^* \leq s_{i,C_i} \end{cases} \tag{9.5}$$

The energy consumed per task $e_i^*(s_i^*)$ is a function of the energy-speed slope $f_{i,j}(s_i^*)$ between the speed interval $(s_{i,j},\ s_{i,j+1}]$ that includes the virtual speed s_i^*. Since the energy-speed slope for each task is a linear function of speed, we can determine the ratio of time spent, $x_{i,j}$ for task t_i under configuration c_j. The objective is to minimize the total energy for all tasks:

$$\min \sum_{i=1}^{N} e_i^*(s_i^*) \tag{9.6}$$

subject to the constraints:

$$\sum_{j=1}^{C_i} s_{i,j} \cdot x_{i,j} = s_i^* \quad \forall i \tag{9.7}$$

$$\sum_{j=1}^{C_i} x_{i,j} = 1 \quad \forall i \tag{9.8}$$

$$0 \leq x_{i,j} \leq 1 \quad \forall i, j \tag{9.9}$$

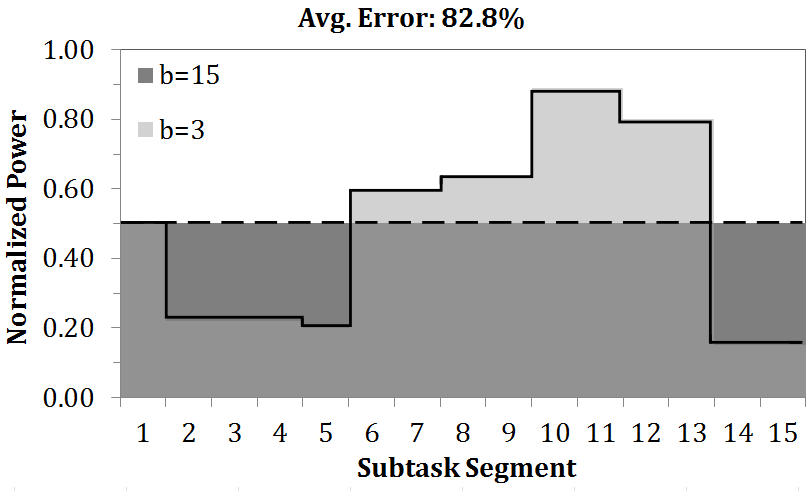
$$\sum_{i=1}^{N} \frac{1}{s_i^*} \leq \frac{1}{s_{global}^*} \quad \tilde{} \tag{9.10}$$

Equations 9.7-9.9 specify that the achieved virtual speed s_i^* for each task may only be achieved by utilizing real speeds from actual configurations for a total of 100% of the allotted time. Equation 9.10 provides the guarantee that the time allotted for each task must satisfy the global execution time deadline. Therefore, the convex piecewise-linear formulation determines the optimal virtual speed for each task (s_i^*) such that the global speed deadline is met and the total energy for all tasks is minimized. It is important to note that because the formulation is convex, the optimal solution will include at most 2 consecutive configurations for each task.

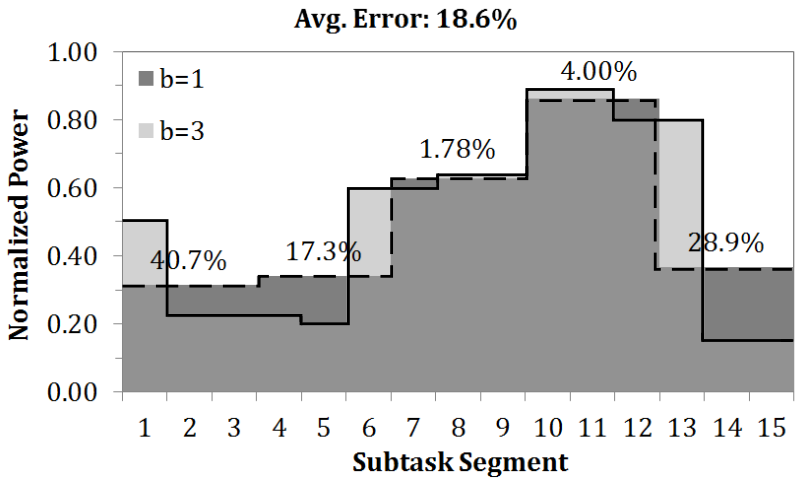
Furthermore, the solution is globally optimal under the following assumptions:

1. *Tasks are long enough that energy and delay for switching configurations is negligible.* The formulation does not consider energy and delay overheads for configuration switching. The key observation is that because there is only at most one configuration switch for each task, and at most one configuration switch between tasks, the time and energy spent in context switching is negligible compared to the overall time and energy expenditure.

2. *Energy consumption is constant throughout the execution of a task.* The formulation does not differentiate between which portions of a task are executed under which configuration. It simply determines the ratios of time, $x_{i,j1}$ and $x_{i,j2}$, for which to execute task i under configurations $c_{i,j1}$ and $c_{i,j2}$, respectively; the actual sections of the task that are run under each configuration are arbitrary. However, energy consumption within a task is highly variable, as we discuss in the next subsection. Therefore, we use this formulation only when application profiling is coarse-grained (i.e. tasks are split into relatively long sub-tasks) and thus the energy consumption distribution within a subtask is unknown.



(a)



(b)

Figure 9.4: Non-uniform task power consumption (solid lines) and average power consumption (dashed lines) for *jpegdec* application under two subtask block sizes: b=15 (a) and b=3 (b). Displayed percentages denote the error from assuming a uniform power consumption within a block. Subdividing a task into smaller subtasks enables greater accuracy as shown by reduced error achieved by block size $b = 3$ (18.6% overall error) vs. the error achieved with $b = 15$ (82.8% overall error).

9.5.2 Fine-Grained Profiling

Assumptions. N tasks (or subtasks) have been profiled for each supported hardware configuration; resulting speed-energy points for each task on each configuration have no uncertainty; each task t_i can be run on C_i configurations $c_{i,1}$, $c_{i,2}$, ..., c_{i,C_i} on its convex enclosure with speeds $s_{i,1}$, $s_{i,2}$, ..., s_{i,C_i}, respectively; configuration switching delay and energy overheads are $\varepsilon_{j\to k}$ and $\delta_{j\to k}$, respectively, for switching from configuration $c_{i,j}$ to $c_{i,k}$ for any task t_i; and an overall speed requirement s_{global} is expected.

Problem Formulation. Our objective is to allocate a single configuration to each task such that total energy consumption of the system is minimized while a specified global execution deadline is satisfied. In this case, because configuration switching overheads are potentially significant (i.e. tasks are of relatively small length), each task is run on only a single configuration and thus the virtual speed of a task is equivalent to the actual speed of the configuration.

Figures 9.4a and 9.4b show the actual non-uniform power consumption of a task, *jpegdec* (solid lines). In Figure 9.4a, the task is profiled at a block size of $b = 15$ instructions, with uniform power consumption assumed within the block, leading to an average error of 82.8%. In Figure 9.4b, a much smaller block size $b = 3$ is used for fine-grained profiling, leading to a much smaller average energy of only 18.6%. This error ultimately translates to suboptimal configuration assignment to subtasks, and therefore minimizing it should be a crucial goal. Therefore, tasks should be profiled at as small a granularity as is practical to minimize energy.

However, recall that at this granularity, configuration switching overheads become significant. To solve the configuration allocation problem under these conditions, we propose another dynamic programming approach, summarized in Figure 9.5. We construct a directed acyclic graph (DAG), where each node represents

the energy and time expenditure for running a block of instructions (or subtask) $b_i, 1 \leq i \leq M$ on configuration $c_j, 1 \leq j \leq C$. A directed edge between two nodes represents the energy and time overheads for switching from the first configuration to the next. In the figure, each vertical stage corresponds to a single subtask or block, and each horizontal set of nodes corresponds to a single configuration, with edges from all configurations at one block to all configurations at the next block. There is also a single start node and a single end node.

The minimal energy set of configurations on which to execute each subtask with global speed requirement s_{global}, then, corresponds to the shortest path from the start node to the end node in terms of energy while not exceeding the global time constraint. This can be solved using the standard dynamic programming approach, with the following caveat. In order to prevent combinatorial explosion of paths that need to be maintained (in order to meet the speed requirement), a minimal amount of bookkeeping must be performed. This bookkeeping maintains at each node only a maximum number of representative Pareto optimal points in terms of speed and energy. In other words, we conduct uniform sampling of Pareto optimal speed-energy points at each node.

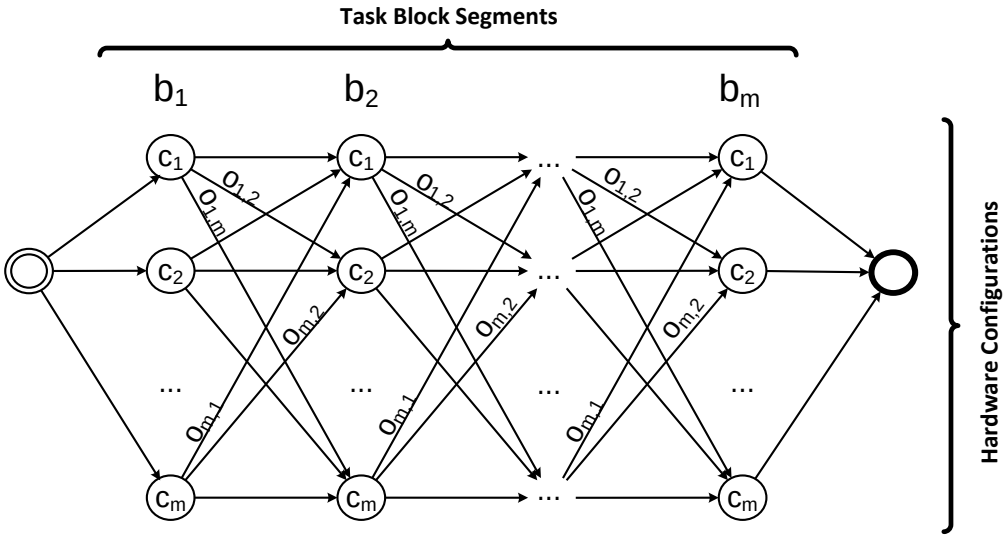


Figure 9.5: Directed acyclic graph (DAG) for obtaining the shortest path when considering switching overheads, as formulated by our dynamic programming approach. Each node ($c1$, $c2$, ..., c_m) per block column represents a given configuration setting (m possible configurations). Energy and delay overheads are modeled with $O_{i,j}$ where i, where $i \neq j$.

Table 9.2: Expected norm-energy of supporting only K out of N configurations, where $N = 26$ (5 allocations at 5 voltages, plus the configuration where all components are powered off). Values are normalized between 0 and 100, where 0 is the optimum case ($K = N$) and 100 is the base case ($K = 1$). Results for $K > 17$ are optimal for all benchmarks and are therefore not shown.

bench	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
adpcmdec	40.8	6.7	1.3	0.3	0.1	0	0	0	0	0	0	0	0	0	0	0
adpcmenc	41.8	8.2	2.1	0.7	0.3	0.1	0	0	0	0	0	0	0	0	0	0
unepic	39.5	7.6	2.1	0.7	0.3	0.1	0	0	0	0	0	0	0	0	0	0
epic	39.4	7.6	2.2	0.8	0.3	0.1	0	0	0	0	0	0	0	0	0	0
g721dec	43.7	6.0	2.0	0.7	0.2	0.1	0	0	0	0	0	0	0	0	0	0
g721enc	40.5	8.4	2.1	0.7	0.3	0.1	0	0	0	0	0	0	0	0	0	0
gsmdec	41.8	7.5	1.9	0.9	0.3	0.1	0	0	0	0	0	0	0	0	0	0
gsmenc	24.7	4.6	1.0	0.5	0	0	0	0	0	0	0	0	0	0	0	0
jpegdec	24.8	4.6	1.0	0.5	0	0	0	0	0	0	0	0	0	0	0	0
jpegenc	42.8	7.0	1.3	0.3	0.1	0	0	0	0	0	0	0	0	0	0	0
mpeg2dec	26.2	5.4	1.8	0.7	0.4	0	0	0	0	0	0	0	0	0	0	0
mpeg2enc	41.6	7.2	2.0	0.9	0.3	0.1	0	0	0	0	0	0	0	0	0	0
**dec*	35.5	14.6	8.7	4.3	3.9	3.8	3.5	3.0	2.9	2.7	2.4	2.3	1.8	1.2	1.1	0
**enc*	38.3	13.3	6.7	3.2	1.7	1.5	1.2	0	0	0	0	0	0	0	0	0
all	37.3	13.7	6.0	3.4	2.5	2.3	2.0	1.9	1.6	1.5	1.4	1.4	1.3	.9	.9	0

9.6 Experimental Results

In this section, we present the experimental results for our approaches using 12 standard benchmarks and 26 hardware configurations, composed of 5 base hardware allocations, 5 possible operating voltage, and one configuration representing the situation where all hardware is power gated off.

9.6.1 Hardware Configuration Selection

Table 9.2 presents experimental results for the procedure described in Section 9.4, where an optimal set of K out of a total N configurations is selected to support expected tasks and their speed requirements with minimal expected energy. We conducted configuration selection for 15 different scenarios: 12 in which only 1 task is expected (each individual benchmark) and 3 in which multiple tasks are expected with equal frequency. The latter set of scenarios consists of all decoder benchmarks, all encoder benchmarks, and all benchmarks. In all scenarios, we used a uniform distribution of speed requirements ranging from no deadline to the fastest possible execution speed.

Each row in the table represents one of these scenarios. There is one column for each value of K, representing the number of configurations that can be supported by the hardware. The values in the table correspond to the energy cost for picking the best K configurations, normalized between 0 (optimal) and 100 ($K = 1$). Note that the configuration in which all hardware is gated off is always supported, and that $K > 17$ is optimal for every scenario, so we omit these results in the table.

We can see from the results that the cost of having a very low K is high, but drops significantly quickly as K is increased. In fact, for the single-task scenarios, $K > 7$ is always optimal. This is because there are at most 7 configurations on the convex enclosure of each task (6 for *adpcmdec* and *mpeg2*$*$, and 5 for *gsmenc* and *jpegdec*. For the latter 3 scenarios, a much higher K is needed for

the optimal solution, but again the energy cost drops quickly at first as K is increased. The ramification is that the hardware (e.g. power gating circuitry) need only support a small number of hardware configuration to approach the optimal energy consumption.

9.6.2 Time Allocation

In this subsection, we evaluate the approaches proposed in Section 9.5, where a single task or set of tasks (or subtasks) are assigned hardware configurations on which to execute to minimize energy in the presence of a global speed requirement. Specifically, we evaluate the energy consumption using the following four methods: a) DVS alone, using the best single configuration that satisfies the deadline; b) DVS and power gating under the assumption of uniform energy consumption across the entire task; c) DVS and power gating after coarse-grained task profiling to reduce the error in energy estimation; and d) DVS and power gating after fine-grained task profiling to further minimize energy.

We use the first approach as a baseline for comparison, where only a single hardware allocation under multiple voltage settings is supported by the hardware platform. The second approach is the one proposed by Dabiri et al., where at most two configurations are used throughout the execution of the task. Finally, the third and fourth are evaluations of our approaches presented in Sections 9.5.1 and 9.5.2, respectively. For coarse-grained and fine-grained profiling, we profiled the benchmarks using block sizes of 100,000 and 10,000 instructions, respectively.

Note that in order to do a fair energy comparison, we evaluate all approaches under the same hard performance constraint, which was set to mid-range achievable clock frequency of 800MHz. Figure 9.6 presents the energy consumption for each of the four considered configuration selection methods. The results are normalized to the energy consumed by the best single configuration (*1-conf.*) that sat-

isfies the desired deadline or performance constraint. We compared the *1-conf.* result against the best 2-configuration result (*2-conf*), our convex programming approach under coarse-grained profiling (*cv*), and our dynamic programming-based approach under fine-grained profiling (*dp*). We again evaluated the approaches were evaluated under three task set assumptions: 1) per application (single task); 2) separate encode ($*enc$ - 6 tasks) and decode ($*dec$ – 6 tasks) classes; and 3) an aggregate task set comprised of all benchmarks ($*all$ - 12 tasks).

The results in Figure 9.6 demonstrate the additional benefits of assigning configurations at smaller subtask granularities. Our results indicate that increasing the number of configuration selection opportunities, achieved by decreasing the subtask size, is shown to achieve greater savings. The number of configuration selection opportunities is 1, 2, 10, and 100 for *1-conf*, *2-conf*, *cv*, and *dp*, respectively. The best configuration that can be selected for the *1-conf* is for the case where the specified timing constraint falls exactly on the same energy and speed point of a base configuration. A *1-conf.* solution would yield the same result as *2-conf* since the virtual speed derived using only one configuration for the entire task would be used. As a result, the additional savings achieved by a *2-conf.* method over the *1-conf.* is limited by the difference between the two surrounding configurations on the convex enclosure. Our results show that utilizing *2-conf.* achieves a *3.39X* max (*1.46X* avg.) savings over the best individual configuration *1-conf.*

Additional energy savings is achieved by subdividing the entire task into smaller subtask sizes, and is shown by the convex programming solution *cv*. For the *cv* results, the entire task (10M instructions) was subdivided into equal 100K instructions, grouped into subtasks. The convex programming solution determines the optimal time allocation for each subtask. As a result, the *cv* method is guaranteed to obtain a solution that achieves greater savings than *2-conf.* by leveraging the non-uniform energy and speed behavior across the subdivided subtasks,

achieving up to *4.64X* max (*2.44X* avg.) energy savings over *1-conf.* Therefore, a convex programming solution capable of assigning configurations at smaller subtask granularities will always yield a superior solution. Recall that in this case, configuration switching overheads are negligible due to long tasks size with respect to the switching granularity (assuming the convex enclosure for each subtask is different).

Eventually the overhead transitions incurred by configuration switching at smaller subtasks reach a limit where the energy and speed costs become comparable to costs associated in running the task under a constant configuration. Therefore, in order to maximize energy savings for these scenarios, it becomes even more critical that configurations are selected in such a way that energy is minimized without compromising performance targets. For our experiments, subdividing tasks into equal 100 subtasks (e.g., 1K instructions = $\frac{1M}{100}$) results in comparable consumed energy and latency requirements to the associated energy (6-20 μJ) and timing transition costs (μs or approx. 600-3000 clock cycles) when performing DVS. Power gating overheads were negligible at these subtask granularities. Therefore, a convex programming solution capable of assigning configurations at a subtask granularity of 100 subtasks would produce a solution that will potentially violate timing constraints. The convex programming solutions are limited since there is no notion in the formulation of the configuration of the previous or subsequent tasks. Although this solution is provably optimal, it is so only in the case where overheads are negligible.

A dynamic programming *dp* solution becomes a natural choice for configuration selection at ultra-fine subtask granularities. Our results show that it consistently generated the minimal energy configuration among the considered methods, achieving up to 8.27X max (4.11X avg.) energy savings while satisfying timing constraints. The dynamic programming solution achieves the best solution due to its advantage of assigning configurations at smaller granularities, effectively

leveraging energy and speed trade-offs across all subtasks of a given application or application set, while accounting for transition overheads.

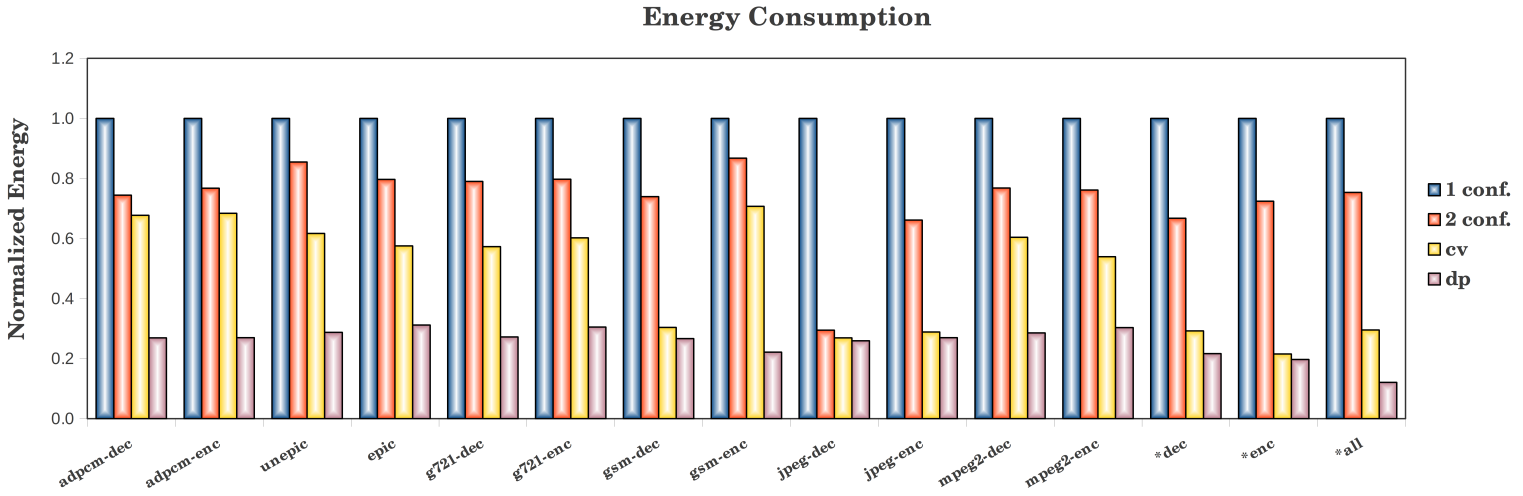


Figure 9.6: Energy consumption for each task and task set *{*enc, dec, all*} under various energy reduction methods: 1) best single configuration (*1 conf.*); 2) best two configurations employing DVS and power gating (*2 conf.*); 3) convex optimization with coarse-grained profiling (*cv*); and 4) dynamic programming with fine-grained profiling (*dp*).

9.7 Summary

We have presented a framework that addresses energy minimization while accounting for performance constraints in the context of determining hardware configurations and task time allocation. First, we have presented a dynamic programming formulation for determining the energy optimal hardware configurations for running a set of expected tasks or applications, while meeting target performance requirements. Furthermore, given a set of operating configurations and a set application behavior, we have presented algorithms for energy optimization under the following two scenarios: 1) application profiling is done at a coarse granularity and therefore tasks are long enough to render configuration switching overheads negligible (convex enclosure and convex programming); and 2) application profiling is done at a fine granularity, where configuration switching overheads are significant (dynamic programming). Finally, we have evaluated our algorithms and conducted quantitative comparison with previous approaches across 15 sets of 12 real applications, achieving up to 4.64X (2.44X average) energy improvements.

CHAPTER 10

Concluding Remarks

We have developed a set of multi-phase energy minimization synthesis techniques that have addressed low to high-level synthesis objectives spanning pre- and-post-silicon domains. Each presented technique conducts energy minimization to satisfy a specified performance target, such as delay or execution time. Additional energy reductions over conventional approaches were achieved by conducting our coordinated multi-phase optimization techniques that address the three major uncertainty factors: 1) technological; 2) application; and 3) optimization. We show that accounting for uncertainty factors in new ways in deep submicron regimes can enable significant energy reductions and yield improvements.

We began in Chapter 3 by presenting a new and efficient cut-based gate-sizing technique to account for accurate gate switching activity, input vector leakage state, and application duty-cycle uncertainty factors in the objective function. Significant energy reductions between 1 to 62% (29% avg.) were achieved over an equivalent gate sizing approach that did not. Improvements were achieved by enabling the optimization tool to perform more efficient energy and speed trade-offs by leveraging the variability in gate-level switching and leakage power across a given circuit design. The impact of duty-cycle requirements were also shown to influence the obtained minimum energy configurations, since the impact of switching and leakage energy contributions can also vary greatly depending on the relative weights of active and idle period requirements of a given design.

Chapter 4 builds upon the previous chapter by leveraging accurate tempera-

ture heat maps when conducting gate sizing/V_{TH} techniques. Only leakage power was addressed, however, since the objective here was to minimize leakage energy in standby-mode and to leverage its exponential dependency on temperature in order to achieve greater energy and speed trade-offs. The advantage of utilizing accurate temperature knowledge over techniques that do not was shown through the achieved incremental energy improvements when applying a set of coordinated multi-phase leakage minimization, such as input vector control, gate replacement, and input-pin reordering techniques. The application of these standard techniques in new coordinated ways motivates the use of iterative refinement for achieving additional energy reductions, as each step results with a new optimization search space. Leakage energy reductions of up to 5.6X (2.3X avg.) were achieved by accounting for temperature at the gate-level through specified temperature corners, which enabled more efficient sizing and V_{TH} selection trade-offs to be made when conducting the aforementioned popular leakage minimization techniques.

Next, the impact of process variation (PV) was investigated when conducting gate-sizing and V_{TH} selection in the near-threshold computing (NTC) regime. An NTC optimization design platform is selected, since delay variation is one of the major challenges to overcome [61]. In this chapter, we presented a novel scenario-based approach for conducting efficient yield optimization by incorporating an efficient circuit partitioning scheme for conducting efficient gate-sizing and V_{TH} selecting under PV uncertainty. The objective was to simultaneously maximize the number of circuit instances that meet specified timing requirements and minimize energy consumption among valid instances. Promising results in terms of power reductions of up to 4.4X (3.3X avg.) with respect to identical timing constraints and delay yield targets, when comparing our PV-aware yield optimization approach over a state-of-the-art gate-sizing and V_{TH} technique that did not take PV uncertainty factors into account [60].

The next two chapters focused on applying gate-level transformations for *en-*

abling additional energy reductions. Starting in Chapter 6, the impact of PV on yield is addressed by applying a gate-level sequential circuit unfolding transformation to enable additional improvements with respect to target delay and energy yield objectives. Unfolding was shown to be effective in minimizing delay variations, since the delay variations were cancelled or average out across a logic depth. Furthermore, conducting unfolding as a pre-processing step enabled more efficient gate-sizing/V_{TH} achieved delay improvements (per result), thus, reducing the amount of additional speed (delay reduction) to be achieved via gate-sizing. Therefore, applying gate sizing/V_{TH} on a post-unfolded circuit required less cells to be placed at high power configurations, thus, minimizing the area overhead incurred by replication and achieve greater energy efficiency. Significant improvements of up to 15.4X in energy and 1.8X in throughput with respect identical delay yield targets were achieved when applying our unfolding technique.

Chapter 7 applies a series of retiming techniques for enabling additional energy reductions using dual-V_{DD} optimization. The first step applies minimum delay retiming to reduce the requirement of placing cells at high V_{DD} to achieve a specified speed target. Next, in order to further achieve energy savings through dual-V_{DD} application, the number of flip-flops is reduced through a min-cut procedure, without impacting the current delay target, in order to reduce the number of cells to be placed at high-V_{DD}. Energy reductions between 8 to 57% (33% avg.) were achieved by conducting these steps in concert.

We then transitioned architectural domain and presented two high-level synthesis techniques for conducting energy minimization techniques at pre- and -post-silicon domains were presented. In Chapter 8, we present a provably minimal energy offline hardware adaptation scheme, where hardware resources are adapted by coordinated power gating and dynamic voltage scaling (DVS) procedure. To identify the best hardware allocations to use for a given set of configurations, a convex programming formulation was used and we proved that the minimum

energy point can be achieved by using at most two configurations that lie on the convex bounding curve. Optimality, however, is only preserved under the assumption that the power profile for an assigned task is uniform. Energy reductions between 1.4 to 1.9X were achieved under the specified assumptions.

Chapter 9 presented two a fine-grained hardware adaptation techniques which applies an identical procedure as in Chapter 8 by sub-dividing a given task into smaller subtasks. Also presented is a path-finding procedure using dynamic programming formulation where, the task is to construct a given hardware architecture platform using a set of functional units (e.g., caches, ALUs, and etc.) and set of predicted tasks (or subtasks) to run. It was shown that a relatively small set of available hardware allocations (e.g., 7 out of 26) were needed to achieve minimum energy. Next, once the given architecture has been specified, the next step is similar to previous chapter. The methods we propose in this chapter, however, differs from Chapter 9 since minimum energy is solved under two assumptions. The first assumption assumes that the hardware transition overheads (e.g., DVS, power gating) are considered to be negligible and is the case when tasks lengths are long enough to mask the effects of transition overheads. A convex piecewise linear formulation is presented for achieving minimal energy for a given timing constraint. Additional energy reductions were achieved over the methods presented in Chapter 9, showcasing the benefits by sub-dividing long tasks blocks into smaller segments to enable additional energy and speed trade-offs. The second assumption assumes that the tasks lengths are such that their power, energy and timing costs are comparable to pertaining DVS and power gating transition costs, thus, requiring transition overheads must be taken into account. A dynamic programming formulation was presented to address this case. The obtained schedule is the minimal energy allocation for the given delay target. The drawback of this procedure, however, is that the timing constraint relaxed to solvable. Therefore, if a given schedule cannot satisfy a target delay, the dynamic procedure can

be re-applied with slight modifications in available hardware allocations until the target delay is met. Energy reductions of up to 4.6X (2.4X avg.) were achieved using our approach.

References

[1] Advanced Configuration and Power Interface (ACPI), http://www.acpi.info, 2012.

[2] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *ACM Operating Systems Principles*, pp. 103-116, 2001.

[3] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Foundations of Computer Science*, pp. 374-382, 1995.

[4] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," *IEEE Foundations of Computer Science*, pp. 520-529, 2004.

[5] T. Ishihara and H. Yasuura, "Voltage scheduling problems for dynamically variable voltage processors," *Low Power Electronics and Design*, pp. 197-202, 1998.

[6] J.-J. Chen, T.-W. Kuo, and C.-S. Shih, "1+ϵ approximation clock rate assignment for periodic real-time tasks on a voltage-scaling processor," *ACM Embedded Software*, pp. 247-250, 2005.

[7] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable voltage core-based systems," *ACM/IEEE Design Automation*, pp. 176-181, 1998.

[8] S. Irani, S. Shukla and R. Gupta, "Algorithms for power savings," *ACM Discrete Algorithms*, pp. 37-46, 2003.

[9] Y. H. Lee, "Scheduling techniques for reducing leakage power in hard real-time systems," *EMRTS*, pp. 105–112, 2003.

[10] R. Jejurikar and R. K. Gupta, "Procrastination scheduling in fixed priority real-time systems," *ACM Languages, Compilers, and Tools for Embedded Systems*, pp. 57-66, 2004.

[11] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," *Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 140-148, 2004.

[12] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Solid-State Circuits*, pp. 473-484, 1992.

[13] R. Jejurikar and R. K. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," *ACM/IEEE Design Automation*, pp. 111-116, 2005.

[14] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low power embedded operating systems," *ACM Operating Systems Principles*, pp. 21-24, 2001.

[15] J. R. Lorch and A. J. Smith, "Pace: a new approach to dynamic voltage scaling," *IEEE Computers*, pp. 856-869, 2004.

[16] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," *ACM/IEEE Design Automation*, pp. 697-702, 1997.

[17] H.-R. Hsu, J.-J. Chen, and T.-W. Kuo, "Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint," *ACM/IEEE Design, Automation, and Test in Europe*, 2006.

[18] F. Dabiri, A. Vahdatpour, M. Potkonjak, and M. Sarrafzadeh, "Energy minimization for real-time systems with non-convex and discrete operation modes," *ACM/IEEE Design, Automation, and Test in Europe*, pp. 1416-1421, 2009.

[19] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "N-version temperature-aware scheduling and binding," *ISLPED*, pp. 331-334, 2009.

[20] A. Vahdatpour, M. Potkonjak, "Leakage Minimization Using Self Sensing and Thermal Management," *ISLPED*, pp. 265-270, 2010.

[21] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki;S. Shigematsu and J. Yamada,"I-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Solid-State Circuits*, pp. 847-853, 1993.

[22] M. C. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *ACM Des. Autom. Electron. Syst.*, pp. 227-248, 1997.

[23] J. Kao and A. Chandrakasan, "Dual-threshold voltage techniques for low-power digital circuits," *IEEE Solid-State Circuits*, pp. 1009-1018, 2000.

[24] I. Hong, M. B. Srivastava, and M. Potkonjak, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," *ICCAD*, pp. 653-656, 1998.

[25] G. Qu and M. Potkonjak, "System Synthesis of Synchronous Multimedia Applications," *IEEE Transactions on Embedded Computing Systems*, pp. 74-97, 2002.

[26] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, "Code Coverage-based Power Estimation Techniques for Microprocessors," *Journal of Circuits, Systems, and Computers*, pp. 1-18, 2002.

[27] G. Qu and M. Potkonjak, "Techniques for Energy-Efficient Communication Pipeline Design," *IEEE Transactions on VLSI*, pp. 542-549, 2002.

[28] J. L. Wong, G. Qu, M. Potkonjak, "Power minimization in QoS sensitive systems," *IEEE Transactions on VLSI*, pp. 553-561, 2004.

[29] M. Drinic, D. Kirovski, S. Megerian, and M. Potkonjak, "Latency-Guided On-Chip Bus Network Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 2663-2673, 2006.

[30] V. Goudar, Z. Ren, P. Brochu, M. Potkonjak, and Q. Pei, "Optimizing the Output of a Human-Powered Energy Harvesting System with Miniaturization and Integrated Control," *IEEE Sensors Journal*, Accepted for publication, 2014.

[31] M. Rofouei, M. Potkonjak, and M. Sarrafzadeh, "Energy Efficient Collaborative Sensing-based Design: Soft Keyboard Case Study," *IEEE Transactions on Human-Machine Systems*, Accepted for publication, 2014.

[32] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," *Low Power Electronics and Design*, pp. 32-37, 2004.

[33] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing Power Using Transformations," *IEEE Transactions on CAD*, pp. 12-31, 1995.

[34] S. Rele, S. Pande, S. nder, and R. Gupta, "Optimizing static power dissipation by functional units in superscalar processors," *Compiler Construction*, pp. 261-275, 2002.

[35] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "Guarded power gating in a multi-core setting" *Computer Architecture*, pp. 198-210, 2010.

[36] H. Xu, W.-B. Jone, and R. Vemuri, "Stretching the limit of microarchitectural level leakage control with adaptive light-weight Vth hopping," *Computer-Aided Design*, pp. 632-636, 2010.

[37] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on GPUs," *ACM Architecture and Code Optimization*, pp. 1-25, 2011.

[38] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," *Computer Architecture*, pp. 66-77, 2006.

[39] J. Flinn and M. Satyanarayanan, "Energy-aware adaptation for mobile applications," *ACM Operating System Principles*, pp. 48-63, 1999.

[40] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH*, pp. 13–25, 1997.

[41] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," *ISLPED*, pp. 9–15, 2000.

[42] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," *DATE*, pp. 518–523, 2004.

[43] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *SIGARCH*, pp. 83–94, 2000.

[44] N. A. Conos, S. Meguerdichian, S. Wei, and M. Potkonjak, "Maximizing yield in Near-Threshold Computing under the presence of process variation," *PATMOS*, pp. 1–8, 2013.

[45] C.-M. Hung, J.-J. Chen, and T.-W Kuo, "Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element," *RTSS*, pp. 303–312, 2006.

[46] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," *CASES*, pp. 164–169, 2002.

[47] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold leakage modeling and reduction techniques," *ICCAD*, pp. 141–148, 2002.

[48] H.-S. Kim, "Impact of scaling on the effectiveness of dynamic power reduction schemes," *ICCD*, pp. 382–387, 2002.

[49] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, pp. 32–38, 2005.

[50] 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," *MICRO*, pp. 469–480, 2009.

[51] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," *ICCAD*, pp. 721–725, 2002.

[52] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," *ASP-DAC*, pp. 473–378, 2006.

[53] T. Wei, P. Mishra, W. Kaijie, and L. Han, "Online task-scheduling for fault-tolerant low-energy real-time systems," *ICCAD*, pp. 522–527, 2006.

[54] M. R. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, and J. Rabaey, "Performance Optimization using Template Mapping for Datapath-Intensive High-Level Synthesis," *IEEE Transaction on CAD*, pp. 877-888, 1996.

[55] M. Potkonjak and M.B. Srivastava, "Behavioral Optimization Using the Manipulation of Timing Constraints," *IEEE Transaction on CAD*, pp. 936-947, 1998.

[56] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans.*, pp. 178–195, 1991.

[57] L. Song and K. K. Parhi "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *JVLSI*, pp. 149–166, 1998.

[58] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *VLSI*, pp. 465–472, 1997.

[59] J. Hu, A. B. Kahng, S. H Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-guided metaheuristics for accurate discrete gate sizing." *ICCAD*, pp. 233–239, 2012.

[60] L. Li, P. Kang, Y. Lu, and H. Zhou, "An efficient algorithm for library-based cell-type selection in high-performance low-power designs," *ICCAD*, pp. 226–232, 2012.

[61] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: reclaiming Moore's Law through energy efficient integrated circuits," *IEEE*, pp. 253–266, 2010.

[62] Q. Xie, Y. Wang, and M. Pedram, "Variability-aware design of energy-delay optimal linear pipelines operating in the near-threshold regime and above," *GLSVLSI*, pp. 61–66, 2013.

[63] S. Seo; R. G. Dreslinski, M. Woh, Y. Yongjunm C. Charkrabari, S. Mahlke, D. Blaauw, and T. Mudge, "Process variation in near-threshold wide SIMD architectures,," *DAC*, pp. 980–987, 2012.

[64] B. Cline, K. Chopra, D. Blaauw, and Y. Cao, "Analysis and modeling of CD variation for statistical static timing," *ICCAD*, pp. 60–66, 2006.

[65] A. Asenov, "Random dopant induced threshold voltage lowering and fluctuations in sub-0.1 um MOSFET's: a 3-D atomistic simulation study," *IEEE T-ED*, pp. 2505–2513, 1998.

[66] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," *DAC*, pp. 338-342, 2003.

[67] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: a model of process variation and resulting timing errors for microarchitects," *IEEE T-SM*, pp. 3-13, 2008.

[68] B. E. Stine, D. S. Boning, and J. E. Chung, "Analysis and decomposition of spatial variation in integrated circuit processes and devices," *IEEE T-SM*, pp. 24-41, 1997.

[69] J. Fishburn and A. E. Dunlop, "TILOS: a posynomial approach to transistor sizing," *ICCAD*, pp. 326-328, 1985.

[70] S. Duvall, "Statistical circuit modeling and optimization," *IEEE International Workshop on Statistical Metrology*, pp. 56-63, 2000.

[71] R. Kendall, "Worst case analysis methods for electronic circuits and systems to reduce technical risk and improve system reliability," *Intuitive Research and Technology Corporation*, http://www.irtc-hq.com/WCA_white_paper_gen.pdf

[72] H. Chang and S. S. Sapatnekar, "Full-chip analysis of leakage power Under process variations, including spatial correlations," *DAC*, pp. 523-528, 2005.

[73] Y. Alkabani, T. Massey, F. Koushanfar, and M. Potkonjak, "Input vector control for post-silicon leakage current minimization in the presence of manufacturing variability," *DAC*, pp. 606-609, 2008.

[74] G. Calafiore and M. C. Campi, "The scenario approach to robust control design," *IEEE T-AC*, pp. 742-753, 2006.

[75] A. Nemirovski and A. Shapiro, "Convex approximations of change constrained programs," *SIAM*, pp. 969-996, 2006.

[76] J. Luedtke and S. Ahmed, "A sample approximation approach for optimization with probabilistic constraints," *SIOPT*, pp. 674-699, 2008.

[77] A. E. Ruehli P. K. Wolff Sr., G. Goertzel., "Analytical power/timing optimization technique for digital system," *DAC*, pp. 142-146, 1977.

[78] C. Zhuo, D, Blaauw, and D. Sylvester, "Variation-aware gate sizing and clustering for post-silicon optimized circuits," *ISLPED*, pp. 105-110, 2008.

[79] A. Davoodi and A. Srivastava,"Variability driven gate sizing for binning yield optimization," *VLSI*, pp. 683-692, 2008.

[80] D. Patil, S. Yun, S.-J. Kim, A. Cheung, M. Horowitz, and S. Boyd, "A new method for design of robust digital circuits," *ISQED*, pp. 676-681, 2005.

[81] M. R. Guthaus, N. Venkateswarant, C. Visweswariah, V. Zolotov, "Gate sizing using incremental parametrized statistical timing analysis," *ICCAD*, pp. 1026-1033, 2005.

[82] E. T. A. F. Jacobs and M. R. C. M. Berkelaar, "Gate sizing using a statistical delay model," *DATE*, pp. 283-290, 2002.

[83] J. Cong, J. Lee, and L. Vandenberghe, "Robust gate sizing via mean excess delay minimization," *ISPD*, pp. 10-14, 2008.

[84] B. Cheng, "Evaluation of statistical technology generation LSTP MOSFETs," *Solid-State Electronics*, pp. 767-772, 2009.

[85] S. Roy and A. Asenov, "Where do the dopants go?" *Science*, pp. 388–390, 2005.

[86] Z. Zhang, Y. Fan, M. Potkonjak, and J. Cong, "Gradual relaxation techniques with applications to behavioral synthesis," *ICCAD*, pp. 529-536, 2003.

[87] N. A. Conos, S. Meguerdichian, and M. Potkonjak, "Gate sizing in the presence of gate switching activity and input vector control," *VLSI-SOC*, pp. 138–143, 2013.

[88] N. A. Conos and M. Potkonjak, "A temperature-aware synthesis technique for simultaneous delay and leakage optimization," *ICCD*, pp. 316–321, 2013.

[89] J. B. Wendt and M. Potkonjak, "Improving energy efficiency in sensing subsystems via near-threshold computing and device aging," *IEEE Sensors*, 2013.

[90] S. Wei, J. X. Zheng, and M. Potkonjak, "Low power FPGA design using post-silicon device aging," *FPGA*, 2013.

[91] S. Wei, S. Meguerdichian, and M. Potkonjak, "Gate-level characterization: foundations and hardware security applications," *DAC*, 222-227, 2010.

[92] S. Wei, F. Koushanfar, and M. Potkonjak, "Integrated circuit digital rights management techniques using physical level characterization," *DRM*, 3-14, 2011.

[93] S. Wei, S. Meguerdichian, and M. Potkonjak, "Malicious circuitry detection using thermal conditioning," *TIFS*, 1136-1145, 2011.

[94] S. Wei and M. Potkonjak,"Scalable hardware trojan diagnosis," *VLSI*, pp. 1049-1057, 2012.

[95] S. Wei, A. Nahapetian, M. Nelson, F. Koushanfar, and M. Potkonjak., "Gate characterization using singular value decomposition: foundations and applications," *TIFS*, pp. 765-773, 2012.

[96] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, "The ISPD -2012 Discrete Cell Sizing Contest and Benchmark Suite," *In Proceedings of ISPD*, pp. 161–164, 2012.

[97] W. N. Li, "Strongly NP-hard discrete gate-sizing problems," *ICCD*, pp. 1045-1051, 1993.

[98] Y. Liu, "A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment," *TCAD*, pp. 223-234, 2010.

[99] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage current reduction in CMOS VLSI circuits by input vector control," *VLSI*, pp. 140-154, 2004.

[100] K. Roy, S. Mukhopadhyay, H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *IEEE*, pp. 305-327, 2003.

[101] A. Calimera, R. I. Bahar, E. Macii, and M. Poncino, "Temperature-insensitive dual-Vth synthesis for nanometer CMOS technologies under inverse temperature dependence," *TVLSI*, 1608-1620, 2009.

[102] J. P. Halter and F. N. Najm, "A gate-level leakage power reduction method for ultra-low-power CMOS circuits," *CICC*, pp. 475-478, 1997.

[103] Y. Ye S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," *VLSIC*, pp. 40-41, 1998.

[104] J. Gu, G. Qu, and L. Yuan, "Enhancing dual-Vt design with consideration of on-chip temperature variation," *ICCD*, pp. 542-547, 2010.

[105] N. Jayakumar and S. P. Khatri"An algorithm to minimize leakage through simultaneous input vector control and circuit modification," *DATE*, pp. 1-6, 2007.

[106] L. Yuan and G. Qu, "Simultaneous input vector selection and dual threshold voltage assignment for static leakage minimization," *ICCAD*, pp. 4-8, 2007.

[107] N. Jayakumar and S. P. Khatri, "A simultaneous input vector control and circuit modification technique to reduce leakage with zero delay penalty," *TODAES*, pp. 1-20, 2010.

[108] A. K. Sultania, D. Sylvester, and S. S. Sapatnekar, "Transistor and pin reordering for gate oxide leakage reduction in dual t_{ox} circuits," *ICCD*, pp. 228-233, 2004.

[109] H. Hassan M. ANis, and M. Elmasry, "Input vector reordering for leakage power reduction in FPGAs," *TCAD*, pp. 1555-1564, 2008.

[110] L. Wei, "Design and Optimization of dual-threshold circuits for low-voltage low-power applications," *TVLSI*, pp. 16-24, 1999.

[111] R. Kumar and V. Kursun"Temperature variation insensitive energy efficient CMOS circuits in a 65nm CMOS technology," *MWSCAS*, pp. 226-230, 2006.

[112] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie, "Temperature-aware NBTI modeling and the impact of standby leakage reduction techniques on circuit performance degradation," *TDSC*, pp. 756-769, 2011.

[113] W. Huang, K. Sankaranarayanan, K. Skadron, R. J. Ribando, and M. R. Stan, "Accurate, Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model," *IEEE Trans.*, pp. 1277-1288, 2008

[114] H. Shiyan, M. Ketkar, and J. Hu, "Gate Sizing For Cell Library-Based Designs," *DAC* pp. 847–852, 2007.

[115] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," *ICCAD*, pp. 724–731, 2011.

[116] S. Joshi, "An Efficient Method for Large-Scale Gate Sizing," *TCSI*, pp. 2760–2773, 2008.

[117] A. Agarwal, K. Chopra, and D. Blaauw, "Statistical timing based optimization using gate sizing," *DAC*, pp. 400–405, 2005.

[118] Nangate FreePDK45 nm Library, *http://www.si2.org/*, 2011.

[119] A. Srivastava, D. Sylvester, and D. Blaauw, D. "Power minimization using simultaneous gate sizing, dual-Vdd and dual-Vth assignment," *DAC*, pp. 783–787, 2004.

[120] Y.-H Huang, P.-Y. Chen, and T. Hwang "Switching-activity driven gate sizing and Vth assignment for low power design," *ASPDAC*, pp. 24–27, 2006.

[121] L. Yuan and G. Qu, "A combined gate replacement and input vector control approach for leakage current reduction," *VLSI*, pp. 173–182, 2006.

[122] Y. Wang, X. Chen, W. Wang, Y. Cao, Y. Xie, and H. Yang, "Leakage power and circuit aging cooptimization by gate replacement techniques," *VLSI*, pp. 615–628, 2011.

[123] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems," *MICRO*, pp. 330–335, 1997.

[124] T. Mudge, "The SimpleScalar-Arm power modeling project," *http://eecs.umich.edu/panalyzer/.*

[125] C. Tsui, R. Y. Au, and R. Y. Choi, "Minimizing the dynamic and subthreshold leakage power consumption using least leakage vector assisted technology mapping," *VLSI Journal*, pp. 76–86, 2008.

[126] S. Wei J. X. Zheng, and M. Potkonjak, "Aging-based Leakage Energy Reduction in FPGAs," *FPL*, pp. 1–4, 2013.

[127] L. Li, J. Sun, Y. Lu, H. Zhou, and X. Zeng, "Low power discrete voltage assignment under clock skew scheduling," ASP-DAC, pp. 515-520, 2011.

[128] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, pp. 5–35, 1991.

[129] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," *Computer-Aided Design of Integrated Circuits and Systems*, pp. 74-84, 1991.

[130] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical path minimization using retiming and algebraic speed-up," *DAC*, pp. 573-577. IEEE, 1993.

[131] S. Dey, M. Potkonjak, and S. G. Rothweiler, "Performance optimization of sequential circuits by eliminating retiming bottlenecks," *ICCAD*, pp. 504-509, 1992.

[132] J. Cong, and S. Lim, "Physical planning with retiming," *ICCAD*, pp. 2-7. IEEE Press, 2000.

[133] A. Mishchenko S. Chatterjee, and R. Brayton, "Integrating logic synthesis, technology mapping, and retiming," *IWLS*, 2005.

[134] S. Hassoun and C. Ebeling, "Architectural retiming: pipelining latency-constrained circuits," *DAC*, pp. 708-713, 1996.

[135] S. Dey and S. T. Chakradhar, "Retiming sequential circuits to enhance testability," *In VLSI Test Symposium*, pp. 28-33, 1994.

[136] A. Kuehlmann and J. Baumgartner, "Transformation-based verification using generalized retiming," *In Computer Aided Verification*, pp. 104-117, 2001.

[137] N. Shenoy and R. Rudell, "Efficient implementation of retiming," *ICCAD*, pp. 226-233, 1994.

[138] S. S. Sapatnekar and R. B. Deokar, "Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1237-1248, 1996.

[139] K. N. Lalgudi and M. C. Papaefthymiou, "DELAY: an efficient tool for retiming with realistic delay modeling," *In Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pp. 304-309, 1995.

[140] J. Monteiro, S. Devadas and A. Ghosh, "Retiming sequential circuits for low power," *International journal of high speed electronics and systems*, pp. 323-340, 1996.

[141] N. Chabini and W. Wolf, "Reducing dynamic power consumption in synchronous sequential digital designs using retiming and supply voltage scaling," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pp. 573-589, 2004.

[142] N. Maheshwari and S. S. Sapatnekar, "An improved algorithm for minimum-area retiming," *DAC*, pp. 2-7, 1997.

[143] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi, "Marsh: min-area retiming with setup and hold constraints," *ICCAD*, pp. 2-6, 1999.

[144] K. Usami, and M. Horowitz, "Clustered voltage scaling technique for low-power design," *ISLPED*, pp. 3-8. ACM, 1995.

[145] M. Igarashi, K. Usami, K. Nogami, F. Minami, Y. Kawasaki, T. Aoki, M. Takano, "A low-power design method using multiple supply voltages," *ISLPED*, pp. 36-41, 1997.

[146] S. Raje, and M. Sarrafzadeh, "Variable voltage scheduling," *In Proceedings of the 1995 international symposium on Low power design*, pp. 9-14, 1995.

[147] J. Chang, and M. Pedram, "Energy minimization using multiple supply voltages," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pp. 436-443, 1997.

[148] F. Li, Y. Lin, L. He, and J. Cong, "Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics," *In Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pp. 42-50. ACM, 2004.

[149] F. Ishihara, F. Sheikh, and B. Nikolic, "Level conversion for dual-supply systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, pp. 185-195, 2004.

[150] A. Gayasen, K. Lee, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and T. Tuan, "A Dual-Vdd Low Power FPGA Architecture," *In Field Programmable Logic and Application*, pp. 145-157, 2004.

[151] A. Srivastava, and D. Sylvester, "Minimizing total power by simultaneous Vdd/Vth assignment," *ASP-DAC*, pp. 665-677, 2003.

[152] W. Lee, H. Liu, and Y. Chang, "An ILP algorithm for post-floorplanning voltage-island generation considering power-network planning," *ICCAD*, pp. 650-655, 2007.

[153] K. Kim and V. D. Agrawal, "Minimum energy cmos design with dual subthreshold supply and multiple logic-level gates," *ISQED*, pp. 1-6, 2011.

[154] D. Markovic, C. C. Wang, L. P Alarcon, L. Tsung-Te, and J. M. Rabaey, "Ultralow-Power Design in Near-Threshold Region," *IEEE*, pp. 237-252, 2010.

[155] M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest," *ISPD*, pp. 168–170, 2013.

[156] J. C. Chi, H. Lee, S. Han, and M. C. Chi, "Gate Level Multiple Supply Voltage Assignment Algorithm for Power Optimization Under Timing Constraint," *VLSI*, pp. 637–648, 2007.

[157] M. Potkonjak and J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations," *ICCAD*, pp. 304-308, 1992.

[158] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen, "HyperLP: A Design System for Power Minimization using Architectural Transformations," *ICCAD*, pp. 300-303, 1992.

[159] L. Guerra, M. Potkonjak, and J. Rabaey, "System-Level Design Guidance Using Algorithm Properties," *VLSI Signal Processing Workshop*, pp. 73-82, 1994.

[160] M. Potkonjak and J. Rabaey, "Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach," *ICCAD*, pp. 90-95, 1994.

[161] M. B. Srivastava and M. Potkonjak, "Power Optimization in Programmable Processors and ASIC Implementation of Linear Systems: Transformation-based Approach," DAC, pp. 343-348, 1996.

[162] M. Potkonjak and A.P. Chandrakasan, "Synthesis and Selection of DCT Algorithms using Behavioral Synthesis-Based Algorithm Space Exploration," *ICIP*, pp. 65-68, October 1995.

[163] I. Hong and M. Potkonjak, "Power Optimization in Disk-Based Real-Time Application Specific Systems," *ICCAD*, pp. 634-637, 1996.

[164] D. Kirovski and M. Potkonjak, "System-Level Synthesis of Low-Power Hard Real-Time Systems," *DAC*, pp. 697-702, 1997.

[165] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Synthesis of Power-Efficient Systems-on-Silicon," *ASP-DAC*, pp. 557-562, 1998.

[166] L. Guerra, M. Potkonjak, and J. Rabaey, "A Methodology for Guided Behavioral-level Optimization," *DAC*, pp. 309-314, 1998.

[167] G. Qu and M. Potkonjak, "Techniques for Energy Minimization of Communication Pipelines," *ICCAD*, pp. 597-600, 1998.

[168] G. Qu, D. Kirovski, and M. Potkonjak, "Energy Minimization of Systems Pipelines Using Multiple Voltages," *International Symposium on Circuits and Systems*, pp.362-365, 1999.

[169] J. Kin, C. Lee, W. Mangione-Smith, and M. Potkonjak, "Power Efficient Media Processors: Design Space Exploration," *DAC*, pp. 321-326, 1999.

[170] C. Lee, J. Kin, W. Mangione-Smith, and M. Potkonjak, "Designing Power Efficient Hypermedia Processors," *ISLPED*, pp. 276-278, 1999.

[171] M. Ercegovac, D. Kirovski, and M. Potkonjak, "Low Power Behavioral Synthesis Optimization Using Multiple Precision Arithmetic," *DAC*, pp. 568-573, 1999.

[172] G. Qu and M. Potkonjak, "Power Minimization using System-level Partitioning of Applications with QoS," *ICCAD*, pp. 343-346, 1999.

[173] G. Qu and M. Potkonjak, "Energy Minimization with Guaranteed Quality of Service," *ISLPED*, pp. 43-48, 2000.

[174] G. Qu and M. Potkonjak, "Achieving Utility Arbitrarily Close to Optimal with Limited Energy," *ISLPED*, pp. 125-130, 2000.

[175] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, "Function-level power estimation methodology for microprocessors," *DAC*, pp. 810-813, 2000.

[176] F. Koushanfar, V. Prabhu, M. Potkonjak, and J.M. Rabaey, "Processors for mobile applications," *ICCD*, pp. 603-608, 2000.

[177] J. M. Rabaey, M. Potkonjak, F. Koushanfar, S. Li, and T. Truong, "Challenges and opportunities in broadband and wireless communication designs," *ICCAD*, pp. 76-82, 2000.

[178] S. Meguerdichian, F. Koushanfar, A. Mogre, D. Petranovic, and M. Potkonjak, "MetaCores: Design and Optimization Techniques," *DAC*, pp. 585-590, 2001.

[179] J. L. Wong, W. Liao, F. Li, L. He, and M. Potkonjak, "Scheduling of Soft Real-Time Systems For Context-Aware Applications," *DATE*, pp. 318-323, 2005.

[180] V. G. Moshnyaga, H. Vo, G. Reinman, and M. Potkonjak, "Reducing Energy of DRAM/Flash Memory System by OS-controlled Data Refresh," *ISCAS*, pp. 2108-2111, 2007.

[181] A. Nahapetian, F. Dabiri, M. Potkonjak, and M. Sarrafzadeh, "Optimization for Real-Time Systems with Non-convex Power Versus Speed Models," *PATMOS*, pp. 443-452, 2007.

[182] J. B. Wendt, S. Meguerdichian, H. Noshadi, and M. Potkonjak, "Semantics-driven sensor configuration for energy reduction in medical sensor networks," *ISLPED*, pp. 303-308, 2012.

[183] V. Goudar and M. Potkonjak, "Energy-efficient sampling schedules for body area networks," *IEEE Sensors*, pp. 1-4, 2012.

[184] V. Goudar and M. Potkonjak, "Dielectric Elastomer Generators for foot plantar pressure based energy scavenging," *IEEE Sensors*, pp. 1-4, 2012.

[185] J. B. Wendt, V. Goudar, H. Noshadi, and M. Potkonjak, "Spatiotemporal assignment of energy harvesters on a self-sustaining medical shoe," *IEEE Sensors*, pp. 1-4, 2012.

[186] S. Meguerdichian and M. Potkonjak, "Low Energy Trusted Private Sensing Using Shared Hardware Random Number Generators," *IEEE Sensors*, pp. 1-4, 2012.

[187] T. Xu, J. B. Wendt, and M. Potkonjak, "Digital Bimodal Function: An Ultra-Low Energy Security Primitive," *ISLPED*, pp. 292-297, 2013.

[188] V. Goudar, Z. Ren, P. Brochu, Q. Pei, and M. Potkonjak, "Optimizing the Configuration and Control of a Novel Human-Powered Energy Harvesting System," *PATMOS*, pp. 75-82, 2013.

[189] V. Goudar, Z. Ren, P. Brochu, M. Potkonjak, and Q. Pei, "Driving Low-Power Wearable Systems with an Adaptively-Controlled Foot-Strike Scavenging Platform," *International Symposium on Wearable Computers*, pp. 135-136, 2013.

[190] N. A. Conos, S. Meguerdichian, and M. Potkonjak, "Coordinated and Adaptive Power Gating and Dynamic Voltage Scaling for Energy Minimization," Accepted for publication, *ASAP*, 2014.