Power-Performance Tradeoffs in ASICs for Next Generation Wireless Communication Datapaths

Farhana Sheikh

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-93
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-93.html

August 14, 2008

Power-Performance Tradeoffs In ASICs for Next Generation Wireless Communication Datapaths

by

Farhana Sheikh

B.Eng. (Carleton University, Canada) 1993
M.S. (University of California, Berkeley) 1996

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

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Borivoje Nikolić, Chair
Professor Andreas Kuehlmann
Professor Paul Wright

Fall 2008

The dissertation of Farhana Sheikh is approved.

Chair Date

Date

Date

University of California, Berkeley

Fall 2008

Power-Performance Tradeoffs In ASICs for Next Generation Wireless Communication Datapaths

by

Farhana Sheikh

Abstract

Power-Performance Tradeoffs In ASICs for Next Generation Wireless Communication Datapaths

by

Farhana Sheikh

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Borivoje Nikolić, Chair

New design methodologies that quickly and systematically explore power-performance tradeoffs between architectures and design variables at each level of design abstraction can enable design innovation and reduce design cost and design time. This dissertation proposes a novel digital design methodology that systematically evaluates power-performance tradeoffs at each level of design hierarchy in the context of constraints from lower levels of design abstraction. It is a holistic approach that uses sensitivity information, which allows designers to systematically and rapidly traverse a vast design tradeoff space, leading to power-performance optimal architectures and enabling short design times. Little formalism has been built around some of the earlier published works that have proposed sensitivity-based design methodologies. This dissertation formalizes the methodology in an optimization framework and algorithm. The framework is conceived using a previously published custom circuit optimizer for power-performance optimization at the leaf cell. The viability of using physical circuit parameters to estimate sensitivity is investigated and shown to be instrumental in reducing design time required to uncover power-performance optimal architectures. A linear relationship between C_{gate}/C_{wire} and sensitivity to gate sizing is uncov-

ered. This first-order linear estimator mitigates the need to calculate derivatives or run large circuit simulations. The use of composition rules is investigated to enable fast generation of energy-delay curves for larger circuit blocks comprised of smaller leaf cells. Energy-efficiency curves are generated for multiple architectures within short periods of time, allowing rapid evaluation of architectures in the context of lower level design constraints and tuning variables such as circuit sizing. The composition process is formalized into an algorithm that can be implemented as a convex optimization program. This provides an automated mechanism for fast design space exploration at architecture, micro-architecture and circuit levels. A digital FIR kernel for use in multi-mode, multi-standard radio transceiver is optimized using the design methodology.

_____

Professor Borivoje Nikolić
Dissertation Committee Chair

To my parents – Asrar and Parveen; my siblings – Fahim and Samia; and my husband – Iftikhar

Contents

List of Figures

List of Tables

Acknowledgements

My adviser, Borivoje Nikolić, has been instrumental in motivating me to complete this research and this thesis. If I had not met him in January 2002, then I do not think I would be here today with a completed dissertation and body of research. Bora was able to convince me to pursue my dreams and goals of obtaining a PhD and to "do something". His support and patience have been endless and his guidance invaluable. I am grateful to him for taking me on as his student and for guiding me through the last 5 years of graduate school. I am also indebted to my dissertation committee members Prof. Andreas Kuehlmann and Prof. Paul Wright. They have been patient and have provided thoughtful evaluation of my research proposal and review of this dissertation. I would also like to acknowledge the support and encouragement from Prof. Haideh Khorramabadi and Prof. Tsu-Jae King Liu. As fellow females in the profession, they understand how tough it can be and know how much personal sacrifice is required. They have consistently encouraged me to stay in the program and complete the degree. I am very grateful to have met them along the way.

I would like to thank the many students and staff at the Berkeley Wireless Research Center for making my stay at Berkeley enjoyable and comfortable. In particular I would like to thank Gary Kelson, the director of the lab for making sure that students feel comfortable and have access to every possible piece of equipment required for their research. Tom Boot is tireless in his efforts to make sure that the center is clean and the environment is pleasing. Brenda has been wonderful with helping out with administrative issues. Brian Richards invested a lot of his time in helping me out with testing in the lab and with the various CAD issues arising from tools. The system administrators, Kevin Zimmerman and Brad Krebs always went out of their way to resolve any computer or networking related issues. Among the many BWRC students, I am particularly grateful to Louis Alarcon and Mubarak Mishra for their friendship and help.

My colleagues in our research group are some of the smartest and nicest people I have had the pleasure to work alongside. Socrates Vamvakos, Radu Zlatanovici, Joshua Garrett, Melinda Ler, Dejan Marković graduated while I was still working on my research. I am

indebted to them for helping me with qual practices, acting as sounding boards for ideas, and giving me feedback on my initial work. Liang-Teck Pang and Bill Tsang walked with me in May as we all participated in the graduation ceremonies. I am grateful to them for their insight and help during tapeout and with lab issues. They have also been extremely supportive and encouraging during the low points in my PhD career. Zhengya Zhang, a fellow Canadian, was always available to drive me back from the airport and onto the many BWRC retreats. He has been a good listener to both personal and professional issues, and always a good source of advice. Renaldi Winoto, has always been around to listen, encourage, and to help out with tapeouts and lab issues. Zheng Guo, Ji-Hoon Park, Vinayak Nagpal, Seng-Oon Toh have encouraged me during the last few stressful days of dissertation writing and lab testing. Their kind words of support will not be forgotten. I am thankful to Lauren Jones and Kenny Duong for their humour and upbeat attitude. The Scrabulous games with Kenny and Lauren were a great way to relax after a busy day.

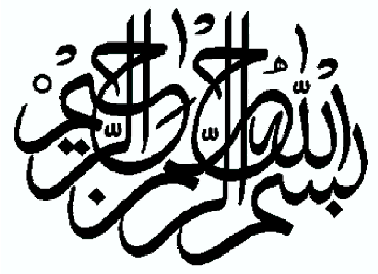
Ruth Gjerde and Mary Byrnes in the graduate office have been invaluable sources of support and encouragement. Mary and Ruth have helped me overcome many challenges during my stay at Berkeley. Their support, caring and compassionate attitude intermingled with inspirational stories of personal challenges helped me continue when things looked bleak.

This research was supported financially by MARCO/C2S2, SRC, BWRC member companies, and the Intel PhD Fellowship. During my studies I had the opportunity to complete two internships at Intel Research. I am thankful to Anthony Chun, Ernest Tsui, Kirk Skeba, Ram Krishnamurthy, Sanu Mathew and the rest of my colleagues at the Circuits Research Lab for providing guidance and support. ST Microelectronics and CMP provided fabrication of the filter prototype.

On the personal side, I would have been lost without my friends outside of the engineering department. In my first year, I got very sick and had to go to hospital. Daphne Taylor-Garcia, a fellow Canadian who I barely knew, was kind enough to stay with me until I recovered. Elisabeth Lamoureux, a friend from my Masters days, was ready with groceries and introduced me to the nicest person I have ever known: Erika Gasperikova. Elisabeth

and Erika helped me through the most darkest periods of my stay at Berkeley, and I am thankful to them for their love, support, and encouragement. Alessandra Nardi, who I met in my early days of my PhD, has become a close friend and confidante. She encouraged me to continue to focus on the PhD when the personal side of life looked bleak. Sierra Boyd acted as the ever-optimistic cheerleader, making sure that I was taking care of myself and keeping healthy. The ladies only outings were the best distraction from a lab full of male engineers.

Most of all, I am indebted to my family. My parents have always believed in my abilities and encouraged me to overcome my weaknesses, no matter the situation. They have stood by me during these last seven years, and have always told me to finish what I started. My siblings, Fahim and Samia, have always been available to listen to my issues, give me good advice and tell it like it is – no holes barred. I am grateful for their honesty, love, and encouragement. My sister-in-law Deeba and my brother-in-law, Farhan, were the proud family members, anxious to see me with a doctorate. Their love and support is very much appreciated. My nieces and nephews – Yusuf, Nuha, Emaan, and Abdullah – provided the best distractions from the tough PhD life and showed me that life is full of many other wonderful things beyond just research in a lab. Last but not least, I am grateful for the support of my husband, Iftikhar, who I met during my studies. He always encouraged me and pushed me to complete my PhD. Even though we were separated by many miles for two years, he never asked me to put him before my studies. I am grateful for his understanding, patience, compassion, and love. Finally, I am thankful to God for providing me with an opportunity to pursue a PhD at Berkeley. It has been a challenging but rewarding experience. — Berkeley, California, August 12, 2008


بسم الله الرحمن الرحيم

Chapter 1

Introduction

> *Integrated circuits will lead to such wonders as home computers — or at least terminals connected to a central computer — automatic controls for automobiles, and personal portable communications equipment.* – Gordon E. Moore, 1965

The design of digital integrated circuits has undergone a paradigm shift. No longer is it feasible to make incremental architecture changes, tweak design methodologies, and simply scale designs to a new technology generation and subsequently reap the benefits of improved performance, at minimal area or energy cost. Traditional digital design, based on minimizing cycle time, is quickly approaching its demise as technologies scale to the 65nm node and below. Process scaling is the first and foremost factor driving the need for innovation in architectures, circuit techniques, device technology and design methodologies [1, 2, 3].

Over the past 40 years, technology scaling has fueled tremendous growth in the semiconductor industry and enabled the development of a multitude of electronic products, from mobile personal communication devices to advanced medical imaging systems. At each new technology generation, the area required to implement a digital function is reduced by approximately half [4], resulting in large levels of functionality integration on a single chip. However, voltages have not scaled at the same rate, chip dimensions have increased leading to increased resistance and capacitance, and architectural innovations have boosted frequencies beyond those achievable via scaling alone [1]. The confluence of these phenomena have led to an exponential increase in power dissipation. This exponential increase

cannot continue due to limits on cost of cooling, packaging, and reliability. Power is now the primary design constraint for both portable and high performance applications. In this power-limited scaling regime, it is necessary to systematically design energy-efficient algorithms and architectures that consume the least power at the required performance.

At the current 65nm technology node and below, leakage power contributes significantly to total power dissipation. In a system that is memory intensive, leakage power can account for up to or even exceed 50% of the total power [5]. Various design techniques to combat leakage power such as power gating using sleep transistors, multiple threshold devices, and adaptive body bias provide flexibility to the designer but also increase design complexity as the design tradeoff space increases considerably with additional tuning parameters.

Technology scaling has also impacted global communication across a chip. At each new technology generation, global communication across a chip becomes more expensive. The delay in global wires increases even with an optimal number of repeaters [6]. Traditional design approaches that attempt to gain performance through wider issue machines will no longer work as these machines require longer wires. Since longer wires will increase in delay at each new technology generation, either the clock will have to slow down or additional pipeline stages will be required [2]. Thus, performance gains will be limited using traditional architectural modifications. Innovative architectures needing fewer global wires or those using new on-chip interconnects will be required to overcome global communication constraints. Design methodologies that can highlight tradeoffs between new devices, power mitigation techniques, new architectures and global communication techniques have become necessary in reaching optimal system design. In addition to all of the above, today process variability is becoming a major challenge that all designers must overcome. New devices, circuit techniques and design methods will need to be developed to overcome process limitations. The efficacy of future innovations will need to be evaluated in the context of today's complex systems without imposing an excessive burden on design time and cost.

Secondary and tertiary factors driving the need for a new design paradigm are limits on instruction level parallelism and number of gates per clock cycle [2]. These two factors, which have historically driven processor performance improvements, are no longer effec-

tive and are causing designers to rethink traditional design styles and methods. In [2], authors show a leveling out of the improvement in effective parallelism across Intel, Alpha, MIPS, HP, Power PC, and AMD processors by early 2004, indicating that traditional design cannot extract any more performance improvements through instruction level parallelism. Also in [2], authors show that the cycle time per fanout-of-four (FO4) delay has slowed down, indicating that the number of gates per cycle has reached its limits in traditional architectures.

The last but important factor driving the need for a paradigm shift in digital design is design cost-per-function and time to market [3]. Consumer demand for new applications on portable devices is primarily driving the increase in system complexity. Meeting these demands while maintaining performance, reliability and keeping power dissipation low increases design cost. Today, designers must understand the interactions between every level of design abstraction – from technology to architecture – to meet complex constraints and requirements. New methodologies that allow designers to quickly and systematically explore tradeoffs between various architectures and design tuning variables at each level of design abstraction can enable design innovation, and reduce design cost and design time.

The task of estimating tradeoffs between power (energy) and performance (delay) for each choice of design parameter at each level of design hierarchy for each different application, under today's extreme design constraints and small time-to-market windows, is inherently complex. The design tradeoff space is vast, spanning a multi-dimensional search space across multiple levels of design abstraction. Finding the most energy-efficient design is an overwhelmingly complex and time-consuming task.

This dissertation proposes a novel digital design methodology that attempts to systematically evaluate power-performance tradeoffs at each level of design hierarchy in the context of constraints from lower levels of design abstraction. It is a holistic approach that uses sensitivity information, which allows designers to systematically and rapidly traverse a vast design tradeoff space, leading to power-performance optimal architectures and enabling short design times. The problem that is posed in this work is extremely difficult to solve completely. Little formalism has been built around some of the earlier published works that

have attempted to describe sensitivity-based design methodologies. The solutions presented here are by no means comprehensive. However, a general framework and formalism is built in this work which can become a basis for further innovation and expansion. The most important aspects of the general problem are addressed and various formalized solutions presented.

The next section in this chapter highlights the primary design driver for the semiconductor industry: next generation mobile systems. The flexible digital filter benchmark presented in this dissertation is a necessary component in next generation mobile systems. Following the brief discussion on next generation mobile systems, this chapter presents a process scaling analysis, summarizing the move from performance-constrained designs to today's power-limited scaling regime. Then this chapter briefly introduces and reviews sensitivity-based circuit optimization and previously published works in this area. The final sections of this chapter provide an overview of the dissertation and the scope of the research.

1.1 Next Generation Mobile Systems

One of the primary design drivers for the semiconductor industry is mobile communication [7]. Mobile communication has steadily increased over the last 15 years, and today it is ubiquitous. From villagers in India to high-powered executives in America, wireless radio is the primary medium of communication. In his prophetic statement in the April 1965 issue of Electronics Magazine, Gordon Moore foresaw large scale electronic integration as the key enabler of personal portable communication [8]. In 1991, the number of world-wide fixed line telephone subscribers were estimated to be 546 million, and mobile subscribers were estimated to be just 16 million. In 2006, the number of mobile subscribers world-wide reached 2.6 billion with fixed line subscribers leveling off to 1.2 billion [9]. The graph in Figure 1.1 illustrates this phenomenal rise in mobile communication; it shows a yearly quadratic increase in mobile service subscribers over a period of 15 years. In 2002, the number of mobile subscribers on the globe surpassed fixed line subscribers.

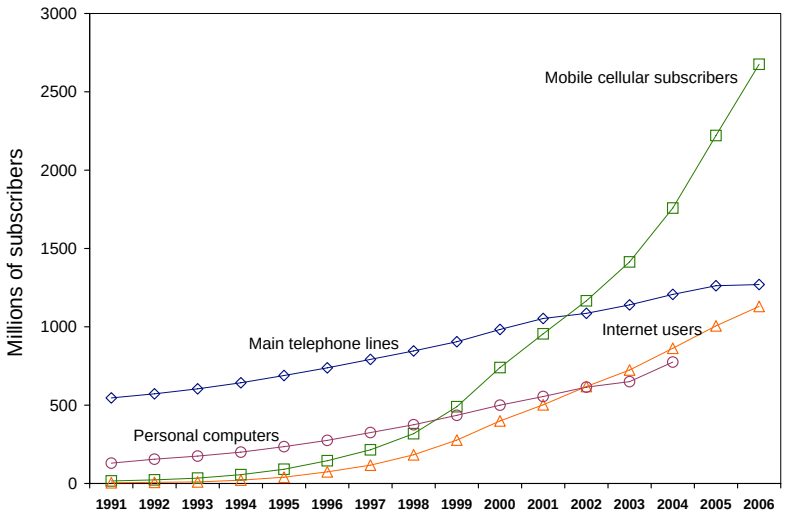


Figure 1.1. Mobile vs. Fixed Telephone Line Subscribers (1991–2006)

During the last 15 years, mobile services revenue has increased at rates not yet seen by other telecommunication revenue streams. Figure 1.2 shows this trend to 2004, where revenue from mobile services approaches fixed line related revenue. In the near future, revenue from mobile services will become the dominant revenue stream for telecommunication service providers [9]. By 2008, cell-phone semiconductor content revenue is projected to make up 20% ($60 billion) of the total semiconductor revenue ($300 billion) [3].

The mobile device market is clearly driving every phase of electronic system design [3]. The enormous growth in consumer demand for integration of voice, video, and data on a single mobile device with small form factor, low power, high reliability and security is creating an exponential increase in circuit complexity per device. The most significant challenge is the accelerated deployment of multi-mode, multi-standard wireless systems. The system and algorithmic complexity required to meet the functionality and flexibility demands is outpacing the scaling benefits of Moore's Law [8]. Innovative design methods, technologies, and system architectures will need to be realized in order to meet the increasing demand for new mobile applications and systems that must operate within the maximum 2W power limit and tight area limits.

For example, a straight-forward implementation of multi-mode operation requires sev-

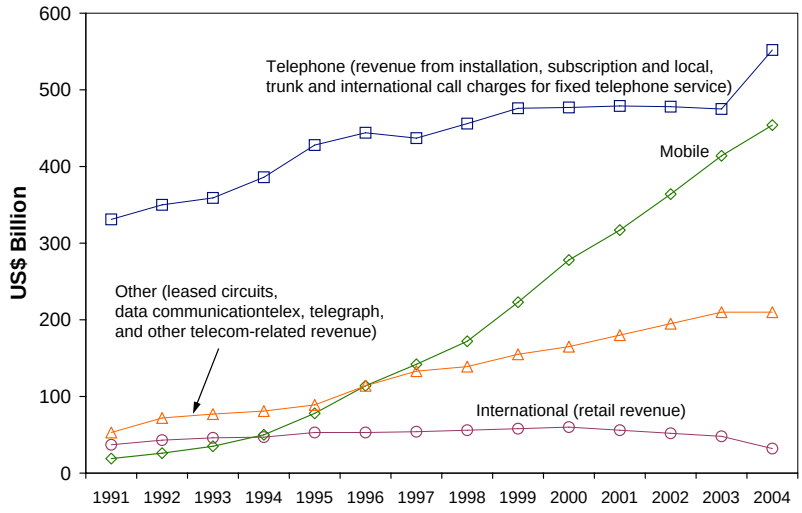


Figure 1.2. Telecommunications Revenue (1991–2004)

eral parallel radios which can be inefficient in terms of energy and area. Voice transmission and reception requires a mobile device to process 2.5G and/or 3G or 9 bands from 0.9 to 2.3GHz; listening to FM radio requires processing at 100MHz and DAB/Satellite radio requires processing at 2.2GHz; GPS requires 1.5GHz processing and DVB-H for digital video broadcast. Consumers also demand personal connectivity using Bluetooth and/or WiFi, both at 2.4GHz. UWB, WiMAX, NFC and RFID will soon become mainstream, all operating in multiple bands. This explosion in radios, processing tasks, and interference avoidance leads to a very complex system design problem. The design tradeoff space is very large; thus significant effort is required to select the optimal architecture which will result in the lowest power consumption for the required performance.

Chapters 6 presents a power-performance optimal digital filter kernel that is a basic constituent of multi-mode, multi-standard wireless radio systems. This block has been optimized using the design methodology proposed in this dissertation.

1.2 Technology Scaling

Technology scaling is the primary enabler for actualizing the technology and economic trends seen in Figures 1.1 and 1.2. In the same April 1965 article cited earlier, Gordon Moore predicted the exponential increase in the number of components that can be integrated on a single chip. He stated that the number of components per integrated function would double every year [8]. Today, this trend is commonly known as "Moore's Law". Since the inception of integrated electronics in the late 1950's, the yearly doubling of complexity on chip continued, as predicted by Moore, until the 1970s when it started to slowdown. In 1975, Moore revised his 1965 statement: he stated that number of transistors on chip would now double every two years. Over the last 40 years, technology scaling has held true to Moore's Law and facilitated over one million times higher integration complexity. Along with circuit and architecture innovation, the advances in semiconductor manufacturing, the inherent properties of technology scaling and device miniaturization have allowed engineers to provide smaller, faster components at little or no additional cost.

Even though smaller transistors mean that each transistor consumes less power, the very high transistor density and requirement for high performance results in overall higher system power dissipation. In the past, the amount of integration that was possible on a single chip was only limited by area. Today, it is also limited by power dissipation. Next, a brief scaling analysis is presented to illustrate why designs are now power-limited.

Three different scaling models exist that facilitate scaling analysis. The first is known as constant electrical field scaling, the second is termed fixed voltage scaling and the last is general scaling. These models are summarized in [10].

Constant field scaling is the ideal scaling scenario presented by Dennard et.al. [4]. Here, all device dimensions and voltages scale by the same factor, S. The recent trends show that minimum physical dimensions of transistors are reduced by a factor $S = 0.7$ in each technology generation. Hence, the area required to implement a digital system is approximately halved at each technology node. Due to the reduced capacitance, performance improves by a factor $1/S$ or 1.4. In order to maintain constant electrical fields, voltages also scale by

the same factor S, resulting in constant power for the same area. Unfortunately, this ideal scaling scenario is not feasible nor practical for actual designs.

Voltages cannot be scaled arbitrarily because new devices must be compatible with existing components. Historically, supply voltages were kept constant over multiple technology generations to maintain compatibility of chip interfaces. Here the fixed voltage scenario is more adept at modeling the trends to the early 1990s. When the voltage is kept fixed and device dimensions are scaled at each technology generation, there is a severe power penalty: a quadratic increase in power density.

Even as supply voltages began to scale down at around the 0.5μm node, manufacturers kept them higher than the ideal V_{DD} = feature size x 10V/μm to help increase performance. High supply voltages and innovative architectures further increased power dissipation and resulted in operating frequencies that were higher than ideal scaling scenario predictions.

A more general scaling model that scales device dimensions by S and voltages by U models today's trends more accurately. Since manufacturers have kept U greater than S – voltages scale down slower than device dimensions – the result has been improved performance at the cost of power dissipation. This trend is exemplified in the exponential rise in operating frequency of Intel's lead microprocessors and the increase in power dissipation by a factor of 2.5 per generation (Figures 1.3 and 1.4).

For practical reasons, this trend cannot continue as power dissipation is limited by the cost of cooling, packaging, and reliability. This is illustrated in Figure 1.4 by the 130W power ceiling reached in high-performance Intel microprocessors. The ceiling is much lower for mobile systems as forced air cooling is not possible and plastic packaging limits power dissipation to about 2W. Portable systems are also limited by battery life which dictate tight constraints on active and leakage power during standby and sleep modes.

Hence, power dissipation is now *the* limiting factor in both high-performance and mobile systems.

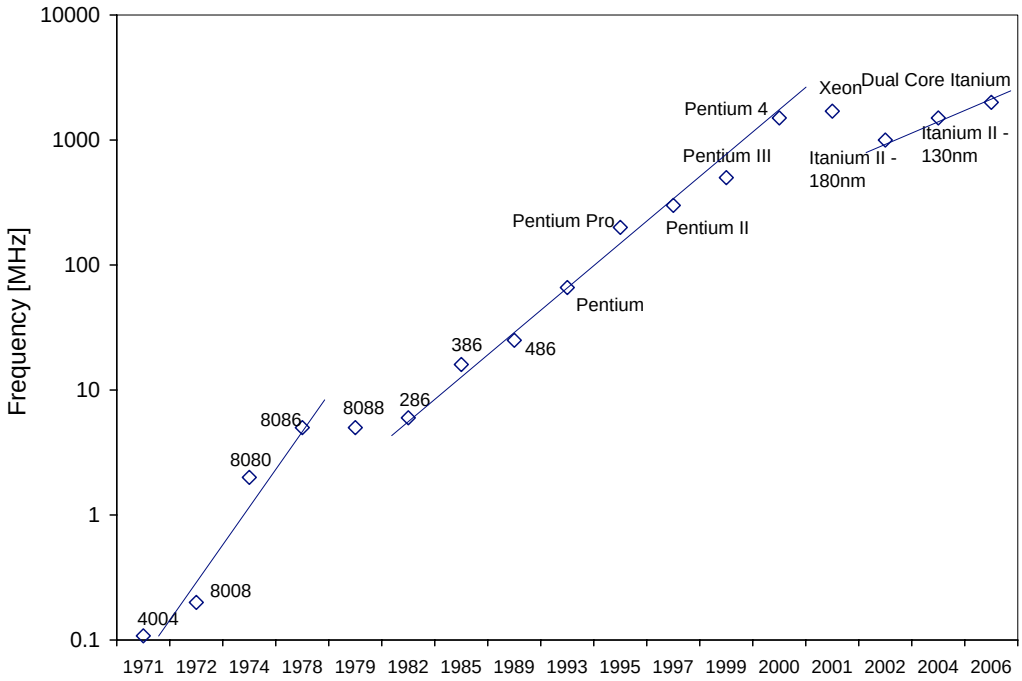


Figure 1.3. Exponential increase in frequency for Intel microprocessors (compiled from www.intel.com and www.i-probe.com/i-probe/ip_intel.html)

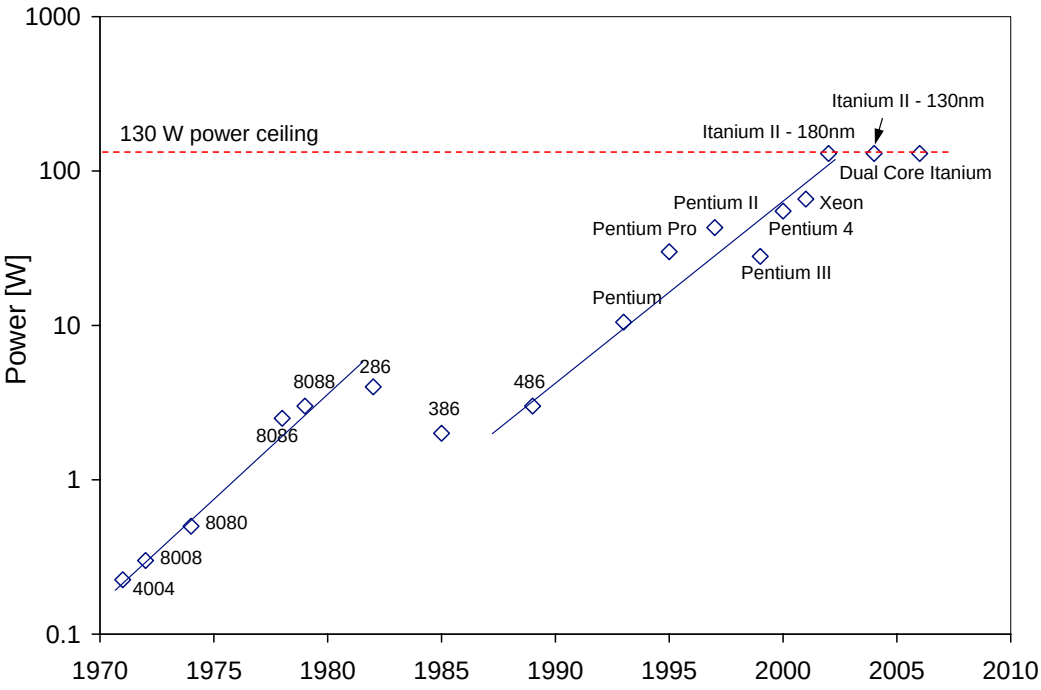


Figure 1.4. Power ceiling for Intel microprocessors (compiled from www.intel.com and www.i-probe.com/i-probe/ip_intel.html)

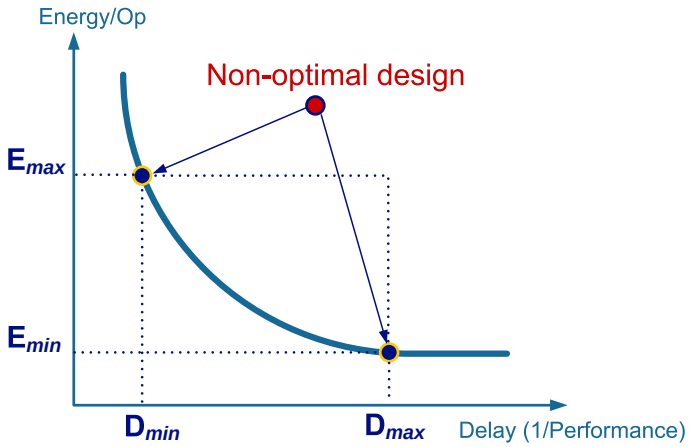


Figure 1.5. Energy-efficient system design

1.3 Energy-Efficient System Design

Under this power-limited scaling regime, energy-efficiency has displaced performance as the primary design constraint in the optimization of digital integrated circuits. Tight power constraints on a system maybe dictated by battery life, chip packaging, and/or cooling costs. The circuit designer must determine the energy-efficiency boundary for the system and try to maximize performance for a given energy-budget or minimize energy for a given performance target. Determining this energy-efficiency boundary allows the designer to better understand which tuning variables will impact energy the most for a given increase or decrease in performance. It allows a designer to tune a non-optimal design and push it to a point (or close to a point) on the energy-efficiency boundary, as illustrated in Figure 1.5.

The task of determining the energy-efficiency boundary for a system-on-chip (SoC) is inherently very complex, time consuming, and cumbersome as it spans a multi-dimensional search space across multiple levels of design abstraction. A systematic yet simple approach that allows designers to accurately evaluate the increase in energy for a unit increase in performance for each design parameter at each level of design abstraction can uncover opportunities for improving energy-efficiency and can reduce design time.

Design tuning variables from architecture, micro-architecture, logic and arithmetic, cir-

cuit and technology levels must all be accounted for when evaluating the power-performance tradeoffs for a system [11, 12]. The impact of wires and wire scaling must also be accounted for in the analysis: for example, modular design with shorter local wires can impact architecture organization [2, 6, 13]. Leakage power and variability in process parameters are increasingly becoming problematic for designers as technology processes go beyond 90nm and 65nm. These affect architecture, micro-architecture, and circuit choices that are available to meet design constraints. The overhead of including circuits that mitigate variability must be evaluated with respect to their overall benefits.

The confluence of a multitude of circuit design variables, architecture and micro-architecture choices, and flexibility in choosing technology parameters such as supply voltage and threshold voltage creates a challenging design environment. Process variations add another layer of complexity to the already difficult task ahead for the design engineer. A systematic design space exploration and optimization methodology, that begins at the architecture level and follows through to the technology level, can result in power-performance optimal system architectures. Such a methodology can also shorten time-to-market and improve design productivity. A key ingredient to the recipe are sound metrics that can be used to evaluate the power-performance tradeoffs for a particular digital function.

This dissertation attempts to address this design problem by proposing a hierarchical, sensitivity-based ASIC design methodology. The optimal design is reached when the marginal costs of all tuning variables at each level of design abstraction are balanced [14, 15, 16]. The design methodology is exemplified in the architecture selection, design,and optimization of two different digital subsystems that operate in a multi-mode, multi-standard wireless radio system. This work builds on prior work by Zlatanovici [17] and Markovic[12], and extends them to include a generic hierarchical methodology that can be incorporated into any off-the-shelf commercial cell-based ASIC design environment.

Next, this chapter will briefly review sensitivity-based circuit optimization and previously published related works.

1.4 Sensitivity-Based Circuit Optimization

The hierarchical design methodology proposed in this dissertation is founded on sensitivity-based optimization where energy efficiency is the primary design objective. The application of gradients or sensitivity analysis to circuit optimization was first introduced in the late 1960s for simulation and automated circuit network optimization in [18, 19, 20]. In the late 1980s and 1990s, gradients were used to solve the constrained circuit sizing problem [21, 22]. In 2002, authors in [14, 15, 16] simultaneously advocated the use of "hardware intensity" or sensitivity to evaluate energy-delay tradeoffs at various levels of design hierarchy. Here, the authors defined sensitivity as the normalized derivative of the energy-efficient curve with respect to a design tuning variable, such as gate size. A similar definition for sensitivity is used throughout this work.

Based on sensitivity, appropriate metrics to evaluate power-performance tradeoffs were proposed in various published works, with each author advocating a different metric [23, 24, 25, 15, 14, 16]. The energy-per-performance ratio [16] and hardware intensity [15] are relative gradients which are mathematically equivalent, and encompass all other metrics. Authors in [14] propose using absolute gradients instead. In fact, no single metric can quantify the energy-efficiency for all digital designs; the appropriate metric is dependent on the type of computation and the weight placed on energy or delay [16].

Zlatanovici in [17] and Markovic in [12] have demonstrated that circuit optimization based on a sensitivity-based analysis leads to energy-efficient design. In [17], a power-performance optimal 64-bit Ling Adder operating at 240ps and consuming 260mW at 1V supply is designed using a gate sizing tool that incorporates sensitivity-based, convex power-performance optimization. The adder was designed using a custom circuit design techniques. In [12], a 4x4 adaptive SVD chip is optimized and implemented using dedicated sensitivity-based analysis at the micro-architecture and circuit levels. This chip provides 250 Mbit/s throughput at 34mW using a 385mV supply. This SVD chip was synthesized and automatically placed and routed using commercial ASIC design tools.

1.4.1 Limitations of prior work

Unfortunately, Zyuban and Strenski [26] have not published results on an application of the theory for hierarchical design optimization. The sensitivity-based optimality conditions for circuits have been used in the design of a few custom circuits [27] and methodologies are dedicated to select circuits [28, 29]. The impact of interconnect has been largely ignored by all previous works. Interconnect is an important consideration for hierarchical design as global and intermediate interconnect affects both energy and delay in the deep sub-micron scaling regime. As technologies scale below 65nm, global wire delay increases, requiring power-performance optimal architectures to reduce global communication. Architectures that are more modular in nature or alternate global communication techniques such as on-chip wireless transmission or optical communication maybe more adept at meeting global communication constraints. Regardless, a sound and systematic design methodology is required to evaluate the tradeoffs in selecting the appropriate architecture.

1.5 Research Scope

The work presented in this dissertation addresses a very difficult problem that in the past, had very little formalism around it except at the circuit level. This dissertation builds on previous custom circuit optimization [12, 14, 15, 17] to produce a design methodology that can be automated within an exisiting standard cell design flow. The goals of the work are to:

1. Produce a hierarchical power-performance optimization framework using an efficient circuit optimizer for power-performance optimization at the leaf cell. The goal is to bridge the gap between architecture-level design and circuit and technology level design using sensitivity information.

2. Investigate the viability of using physical circuit parameters to estimate sensitivity. The focus is on modeling sensitivity to gate sizing, where sensitivity to gate sizing is

a function of C_{gate}/C_{wire} and C_{in}. The investigation of using this metric leads to a thorough study of the impact of wires on sensitivity-based optimization.

3. Investigate the use of composition rules to enable fast generation of energy-delay tradeoff curves for larger circuit blocks comprised of smaller leaf cells.

4. Formalize the composition process into an algorithm that can be implemented efficiently. The goal of the research is not to create a tool but to propose a methodology and formulate possible algorithms that could be implemented in the future.

5. Demonstrate the methodology in the power-performance optimization and design of a digital filter kernel for a multi-mode, multi-standard wireless radio transceiver.

1.6 Dissertation Overview

The dissertation is broken up into two main parts. The first part focuses on building models, using the models to construct the design methodology, and then formalizing the key ingredients into algorithms that can be implemented in software. The second part focuses on designing and implementing a key component of multi-standard wireless radio systems to demonstrate the viability and efficacy of the design framework.

Chapter 2 reviews the mathematical foundations of sensitivity-based design and gradient-based optimization which is later used to formalize the key ingredients of the design methodology. Chapter 3 follows with detailed presentation of the hierarchical power-performance optimization methodology, highlighting its important components which are formalized into implementable programs.

Chapter 4 focuses on reviewing existing low level models for delay, power, energy, area. In the same chapter, the impact of wire capacitance and wire resistance on sensitivity is investigated. Including interconnect information in the model for sensitivity impacts design choices at higher levels of design hierarchy such as architecture selection. Chapter 5 details the development of a model for sensitivity to sizing which includes the impact of interconnect.

Chapter 6 presents the architecture optimization of a multi-standard radio digital FIR. The chapter starts with a discussion of the design tradeoff space, followed by a description of the implementation and results of a taped out chip. This work was carried out mainly at an internship at Intel Research. The final chapter concludes and summarizes the key accomplishments and results of the work presented in this dissertation. It also gives directions for future research.

Chapter 2

Power-Performance Optimization

> *Overwhelming evidence is being amassed, however, that the digital computer may enter the network design decision process.* – Ronald. A. Rohrer, 1967

The hierarchical power-performance optimization of ASICs presented in this work lies in the realm of multi-objective combinatorial optimization problems. The nature of the problem is discrete, with multiple conflicting optimization criteria: minimize energy and minimize delay. There are multiple levels of design hierarchy, each with a vast number of tuning variables. The constraints or feasibility sets are not necessarily convex and neither are the objective functions. One common approach that has been used to address such problems in the past is to flatten the hierarchy and transform the conflicting objectives into a single objective function. For example, delay is minimized subject to energy and/or area constraints; or energy is minimized subject to delay and/or area constraints. Sometimes the objective functions and constraints are massaged into, or approximated by, continuous convex functions and subsequently solved using efficiency convex optimization techniques. Heuristics are then employed to arrive at a discrete solution. Unfortunately, these techniques are not very scalable to multiple tuning variables and multiple levels of design hierarchy.

The best answer to a multi-objective combinatorial optimization is to provide the designer with an entire set of Pareto optimal solutions [30], or a subset, having an image under the energy-delay tradeoff space. The Pareto set captures the notion of tradeoff, allowing designers to ascertain the best architecture, algorithm, circuit, and technology for the given

application and design constraints. Unfortunately, computing the Pareto set is hard because: (1) typically it is exponential in size; and (2) computing one of the Pareto optima is often NP-hard [30]. The best that can be done is to construct an estimate of the Pareto set and traverse it systematically using gradients or sensitivities, with the goal of obtaining the optimal digital system for the given constraints.

The use of gradients or sensitivity analysis has been applied to circuit optimization since the late 1960s for simulation and automated circuit network optimization [18, 19, 20]. By the late 1960s, digital computers were more prevalent in academic and industrial research labs and they were actively used as aids for designing circuits, as Rohrer mentions in his November 1967 article [18]. Optimization methods that could be translated into algorithms and computer-aided tools were being actively investigated at the time [31]. Specifically, gradient-based iterative optimization methods were applied to fixed-structure lumped, linear, time-invariant RLC network design problems [18, 20] and optimal design and synthesis of switching circuits [19]. Sensitivity of network performance to parameter variation was used as a guide to iterate to an optimum set of design parameters. Numerous methods were used to determine sensitivities – ranging from variational calculus [18] to generating expressions directly from forming adjoint networks [20]. A form of convexity analysis was also used in these optimization techniques to determine whether the resulting solution was a local or global minimum [31].

In their comprehensive article, describing the state-of-the-art computer-aided design in 1967, Temes and Calahan summarize the advantages of iterative optimization [31]:

> *The main advantage of design techniques based upon iterative optimization is then their flexibility. They can incorporate all kinds of constraints, can lead to compromise solutions reconciling conflicting requirements carrying different weights, and can accommodate prescribed active elements, nonlinearities, parasitics, as well as restrictions on the types and values of the elements. ... Also, such important practical considerations as the sensitivity of performance to small changes in the element values can be used as criteria in the optimization of the network.*

The advantages they list are exactly the reasons why research and development in the area of iterative optimization has advanced considerably since the late 1960s. Today, the use of

gradient-based optimization in the design of circuits and systems is prevalent. The basic principles of iterative optimization using gradients has not changed significantly since the late 60s but there has been substantial advancement in the efficiency and optimality of algorithms [32, 33].

The hierarchical design methodology proposed in this work is founded on sensitivity-based (or gradient-based) optimization where energy efficiency is the primary design objective [11, 15, 16]. The algorithms described in Chapter 3 that formalize the propagation of sensitivities across multiple layers of design abstraction are based on convex programs. The techniques described in this dissertation provide an estimate of the Pareto set which allows designers to choose how to tradeoff energy for delay using any number of design tuning variables at any level of design hierarchy.

The first section in this chapter reviews basic material on gradient-based optimization and convex programs. It is followed by a summary of how sensitivity analysis has been applied to the circuit sizing problem and then Section 2.3 introduces more recent interpretations and uses of sensitivity analysis to optimize circuits and micro-architectures for energy-efficiency.

2.1 Gradient-Based Optimization

The material presented in this section is recapped from Bertsekas's book on nonlinear programming [32] and from a book on convex optimization by Boyd and Vandenberghe [33]. For a more comprehensive treatment of the subject, the reader is referred to the sources cited earlier. Only those topics that are referred to in later chapters in this thesis are summarized.

An iterative, gradient-based optimization method starts off at a "best-guess" estimate of the solution vector, $\vec{x}^0$. Then successively improved solutions, $\vec{x}^1, \vec{x}^2, \vec{x}^3, \ldots$, are generated based on descent conditions – characterized by the gradient – with the goal of minimizing the objective function, f, to an optimum point, $f(\vec{x}^*)$. The gradient,$\nabla f(\vec{x})$ is the normal to the surface of the objective function at a particular point $f(\vec{x})$. For continuously differ-

entiable functions, the gradient at local or global minimum is equivalent to zero. A formal mathematical description of gradient methods is presented in the following, beginning with a few necessary definitions.

2.1.1 Modeling of Optimization Problems

Mathematical modeling of optimization problems can be represented by a constraint set X and a cost function f. The constraint set constitutes the available decisions or choices. The set X has finite dimensions and is a subset of $\Re^n$. For example, the set $X \subseteq \{0,1\}^n$ is a constraint set. The cost function f maps the set $X \subseteq \Re^n$ onto a scalar measure of the undesirability of choosing some $\vec{x}$ in the set X. The goal of the optimization is to find the optimal decision $\vec{x^*} \in X$ such that $f(\vec{x^*}) \leq f(\vec{x})$, $\forall \vec{x} \in X$. The decision $\vec{x}$ is an n-dimensional vector, $(x_1, x_2, \ldots, x_n)$. In the remainder of this work, the vector sign will be omitted for clarity as it will be assumed that x is a vector unless otherwise stated.

If the set X has infinite number of choices, for example $X = \Re^n$, then the optimization problem is continuous; otherwise, if X is finite (e.g. $X = \{0,1\}^n$), then the problem is discrete. Discrete optimization problems are generally more difficult to solve than continuous ones; these problems are usually solved using combinatorial or discrete mathematics. Continuous optimization problems are solved using calculus and maybe massaged into convex optimization problems which can generally be solved very efficiently.

If the cost function f is nonlinear or the constraint set X is specified by nonlinear equations and inequalities then the problem is classified as a nonlinear programming problem. An unconstrained nonlinear programming problem is given by:

$$\text{minimize } f(x) \text{ subject to } x \in \Re^n \tag{2.1}$$

A constrained nonlinear programming problem is given by:

$$\text{minimize } f(x) \text{ subject to } \begin{cases} x \in \Re^n \\ g_i \leq 0 \quad i = 0 \ldots n \\ h_j = 0 \quad j = 0 \ldots m \end{cases} \tag{2.2}$$

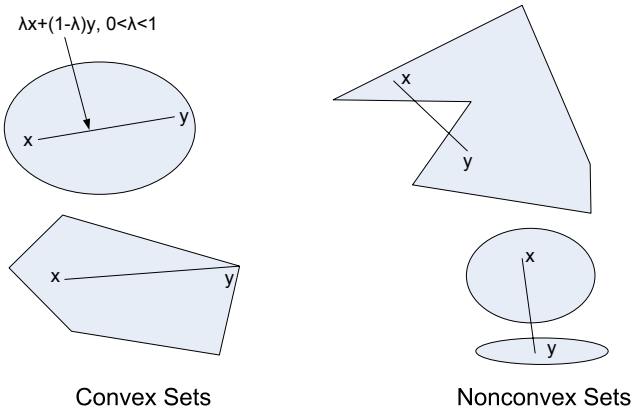


Figure 2.1. Convex and non-convex sets

If there are no equality constraints, then the problem is known as an inequality constrained optimization problem. If there are no inequality constraints, then problem is termed an equality constrained optimization problem.

General unconstrained optimization problems that are continuous in nature are usually solved by iteration of gradient methods and convergence analysis [32]. Optimality conditions play an important role in determining whether a global minimum exists.

A point $x^* \in X$ is a global minimum if $f(x^*) \leq f(x), \forall x \in X$. It is strict if the inequality is strict, that is: $x^* \neq x, \forall x \in X$. A point is a local minimum if $f(x^*) \leq f(x), \forall x \in \| x - x^* \| < \epsilon$ (in the neighborhood around x^*). It is a strict local minimum if the inequality is strict. Convexity plays an important role in determining whether a unique minimum exists for a particular optimization problem.

2.1.2 Convex Sets and Convex Functions

When $f(x)$ is a convex function, a local minimum is also a global minimum over X; if $f(x)$ is strictly convex, then there is at most one global minimum. This fact is extremely important in optimization. If an objective function can be shown to be convex, then it is known apriori that any local minimum found will also be a global minimum and hence the optimization can stop as soon as a local minimum is discovered. This fact leads to very

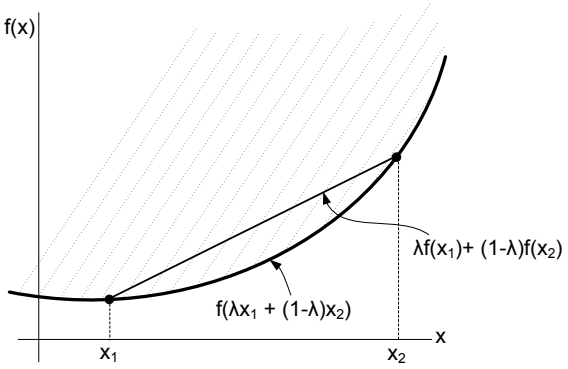


Figure 2.2. Epigraph of a convex function and Jensen's inequality

efficient algorithms that can solve convex optimization problems in very short amounts of time. Mathematically, convex sets and functions are defined as follows. A set $X \subseteq \Re^n$ is convex if for all $x, y \in X$ and $\lambda \in [0, 1]$, the vector $\lambda x + (1 - \lambda)y \in X$. This means that the points along any line segment connecting two points in the set X also belong to the set X. Figure 2.1 shows some examples of convex and non-convex sets.

A function $f : \Re^n \rightarrow \Re$ is convex if and only if its epigraph is a convex set; the epigraph of f is defined as:

$$epi(f) = \left\{(x, y) \in \Re^{n+1} \,|f(x) \leq y,\, x \in \Re^n, y \in \Re\right\} \tag{2.3}$$

The epigraph, which are the points above the graph, links convex sets to convex functions [33]. This is useful because results about convex sets can be translated into results about convex functions. For a convex function, Jensen's inequality holds:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y),\, \forall x, y \in \Re^n, \lambda \in [0, 1] \tag{2.4}$$

Equation (2.4) says that the line segment between $(x, f(x))$ and $(y, f(y))$, which is the chord from x to y, lies above the graph of f. That is, any point on the line $\lambda f(x) + (1-\lambda)f(y)$ lies inside the epigraph of f. Figure 2.2 illustrates the epigraph of a convex function modeled by equation (2.3) and Jensen's inequality.

If a function can be recognized as being convex, then by definition, a local minimum for the function is also the global minimum. A linear function is convex; any vector norm is

convex; and the weighted sum of convex functions, with positive weights, is also convex [32]. These hints help in determining whether any particular function is convex.

Gradient analysis is very useful when the objective function in an optimization problem is shown to be continuously differentiable and convex. For such functions, x^* is a global minimum if and only if the gradient is equal to zero, $\nabla f(x^*) = 0$ [32, 33]. There are multiple ways of solving such an optimization problem; they are discussed in [32] and [33]. A method based on iterative gradient analysis is described in the next section.

2.1.3 Gradients and Iterative Optimization Methods

The discovery of stationary points, x^* where $\nabla f(x^*) = 0$, is a hard problem to solve in general, but motivates iterative optimization methods. Iterative descent methods are computational methods for unconstrained minimization. A subset of these methods are termed gradient methods because they use gradient or variation analysis to determine descent and stopping conditions.

If the cost function $f(x)$ is continuously differentiable then gradients and Taylor series expansions can be used to compare the cost of a particular decision with the cost of decisions that are small variations of the original [32] . This type of variation analysis of decisions near the optimal, x^*, yields some necessary optimality conditions.

If the cost function $f(x)$ is twice differentiable, and $\nabla f(x^*) = 0$ and, in addition, if $\nabla^2 f(x^*)$ is positive semidefinite, then the vector x^* is a local optimum point. If only the $\nabla f(x^*) = 0$ condition is satisfied, then x^* is only a stationary point. And as mentioned earlier, if the cost function is convex, then x^* is a global optimum. In this case, $\nabla f(x^*) = 0$ and X is open are sufficient conditions for optimality. These optimality conditions help in constructing iterative optimization methods which are defined and detailed in [32]. A short general summary is given below.

Consider an unconstrained minimization of a continuously differentiable function, $f : \Re^n \rightarrow \Re$, and $f(x) \geq L$, $\forall x \in \Re^n$. An iterative method generally starts off with an initial guess of the solution vector, $x^0 \in \Re^n$. Then, successive solutions, $x^1, x^2, x^3, \ldots$ are

generated such that $f(x^{k+1}) < f(x^k)$, with the goal to decrease f all the way to $f(x^*)$. A gradient-based optimization method uses gradients to determine descent and stopping conditions.

Convergence analysis in iterative methods is important because a global minimum is not always guaranteed unless the objective function is convex. In fact, a local minimum is not even guaranteed. The goal of convergence analysis is to show that the descent method will *not* converge to a non-stationary point. The most that can be expected from a general gradient-based descent method is that will converge to a stationary point (i.e. $\nabla f(x^*) = 0$). It is also important to measure the speed of convergence in the given parameter space.

So far, the discussion has been focused on unconstrained minimization of a general objective function, $f(x)$. As seen briefly in the discussion of convexity, if the objective function or constraint set can be characterized as convex, then the optimization is simplified and is efficient. However, in general cases, the optimization can be slow and become trapped in local minima. The efficacy of a general iterative method is highly dependent on the initial guess. If the initial guess is appropriately chosen, the iterative optimization may lead to the global minimum. However, if a bad initial guess is chosen, then the minimization may get stuck at a local minimum or even worse, at a stationary point. This is illustrated in Figure 2.3. Thus, the goal is to avoid these pitfalls by exploiting properties of objective functions and constraint sets to efficiently arrive at the optimal solution.

Certain classes of problems can be solved efficiently by exploiting the properties of the objective functions and constraint sets, as alluded to by the discussion on convexity. Unfortunately, it is rare that problems will fit nicely into the classes of problems that can be solved efficiently. However, approximations and reformulation of the original problem may lead to a form of the problem that does fit nicely into those classes of problems that can be solved efficiently, such as convex formulations. Transformation of the original problem is carried out via change of variables, additional variables and constraints, and exploitation of duality.

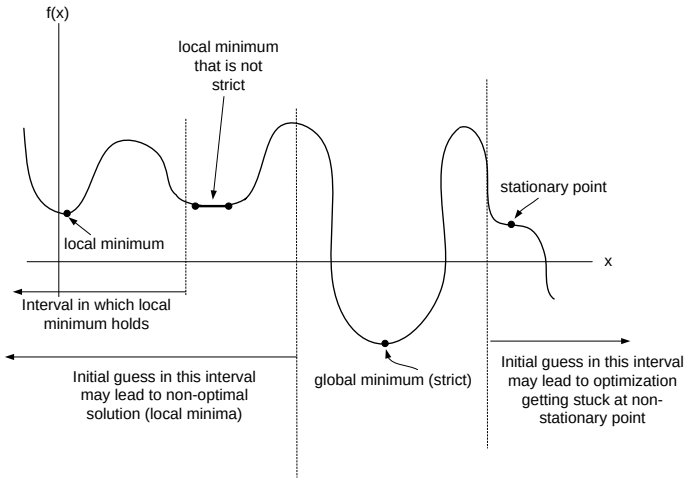


Figure 2.3. Optimizations leading to local and global minima, and stationary points

2.1.4 Lagrangian Theory and Methods

Unconstrained problems of the form minimize $f(x)$ (which may not be convex) are usually solved by iterative methods such as those discussed in the preceding sections. Lagrangian methods solve constrained optimization problems with equality and inequality constraints. Auxiliary variables known as Lagrange multipliers help augment the objective function with a weighted sum of the constraint functions. For example, consider the constrained optimization problem given in Equation (2.2). The *Lagrangian* is then defined as:

$$L(x, \lambda, \nu) = f(x) + \sum_{i=0}^{n} \lambda_i g_i(x) + \sum_{j=0}^{m} \nu h_j(x) \tag{2.5}$$

The vectors λ and ν are Lagrange multipliers that can be viewed as penalties for violating constraints. If the constraint penalties are set appropriately, minimizing the Lagrangian given in Equation (2.5) leads to an unconstrained optimization problem which approximates the original constrained optimization problem. Lagrange multipliers characterize the optimal solution and provide sensitivity information. They quantify, up to a first order, the variation in optimal cost caused by variations in the problem data, x [32]. It is assumed that f, g_i, and h_j are continuously differentiable. The basic Lagrange Multiplier Theorem

states that for a given local minimum, x^*, there exist scalars λ_i and ν_j such that:

$$\nabla_x L(x^*, \lambda^*, \nu^*) = \nabla f(x^*) + \sum_{i=0}^{n} \lambda_i \nabla g_i(x^*) + \sum_{j=0}^{m} \nu_j \nabla h_j(x^*) = 0 \tag{2.6}$$

The Lagrange dual function is defined as the minimum of the Lagrangian over x. The Lagrangian dual function is concave even when the problem in Equation (2.2) is not convex. The dual function gives lower bounds on the optimal value of Equation (2.2) which is verified in [33]. The *best* lower bound that can be obtained from the Lagrange dual function, $L(x, \lambda, \nu)$, is given by the following optimization problem:

$$\text{maximize } G(\lambda, \nu) = inf L(x, \lambda, \nu) \text{ subject to } \lambda \succeq 0 \tag{2.7}$$

The problem in Equation (2.7) is termed the Lagrange dual problem and the optimal set (λ^*, ν^*) are the optimal Lagrange multipliers, if they are optimal for Equation (2.7). The Lagrange dual problem in Equation (2.7) is a convex optimization problem, regardless of whether the primal problem in Equation (2.2) is convex. This is because the problem in Equation (2.7) is a maximization of a concave function which is equivalent to minimizing a convex function; and the constraint set is convex. The optimal solution to this problem, $G(\lambda^*, \nu^*)$ is the best lower bound on the solution to the original constrained optimization in Equation (2.2). That is:

$$G(\lambda^*, \nu^*) \leq f(x^*) \tag{2.8}$$

If the original problem is not convex, this condition is known as *weak duality*. If $G(\lambda^*, \nu^*) = f(x^*)$ holds, then this condition is termed *strong duality*. The bound given by Equation (2.7) is then tight. Strong duality may hold even in the case where the original problem is not convex. However, strong duality does not hold in general. In cases where the original problem is convex and Slater's condition on the inequality constraints is satisfied, then strong duality holds [33].

Sensitivity analysis stems from strong duality. If strong duality holds, then the optimal values, (λ^*, ν^*) provide a mechanism for variation analysis around the optimal point $(x^*, f(x^*))$. This information is important because it allows designers to understand which constraints have greater impact on the optimal solution. Further, if the optimal solution is

differentiable at $(x^*, f(x^*))$, then it can be shown that the optimal doublet (λ^*, ν^*) is related to the gradient at $(x^*, f(x^*))$. Readers are referred to Section 5.6 in [33] for a detailed treatment of sensitivity and perturbation analysis.

The important fact to keep in mind is that, if strong duality holds, then the optimal Lagrange multipliers resulting from the optimization in Equation (2.7) automatically provide the gradients at the optimal solution of the original problem in Equation (2.2). Most optimizers that solve the primal problem by solving the dual problem, also provide the user with the optimal Lagrange multipliers, thus making it easy to compute gradients [17].

Since dual feasible points establish a bound on how suboptimal a given feasible point is without actually knowing the exact value of $f(x^*)$, one can use the *duality gap* as a non-heuristic stopping condition in an optimization algorithm. The duality gap associated with a primal feasible point x and a dual feasible point (λ, ν) is defined as:

$$f(x) - G(\lambda, \nu) \tag{2.9}$$

If the algorithm generates a sequence of primal feasible points x^k and dual feasible doublets (λ^k, ν^k) and $\epsilon > 0$ is a threshold on the required accuracy, then the stopping criteria for an algorithm is given as:

$$f(x^k) - G(\lambda^k, \nu^k) \leq \epsilon \tag{2.10}$$

Strong duality must hold if ϵ is made arbitrarily small. If the duality gap is equivalent to zero, then x is primal optimal and (λ, ν) is dual optimal.

If strong duality holds, and the primal and dual optima are attained, then x^* minimizes $L(x, \lambda^*, \nu*)$ over x. In addition, since x^* minimizes $L(x, \lambda^*, \nu*)$ over x, it implies that the gradient at x^* must be equivalent to zero:

$$\nabla f(x^*) + \sum_{i=0}^{n} \lambda_i^* \nabla g_i(x^*) + \sum_{j=0}^{m} \nu_j^* \nabla h_j(x^*) = 0 \tag{2.11}$$

This results in the following conditions [33] which are termed the *Karush-Kuhn-Tucker*

(KKT) conditions:

$$\begin{aligned} g_i &\leq 0, \quad i = 0, 1, \ldots, n \\ h_j &= 0, \quad j = 0, 1, \ldots, m \\ \lambda_i^* &\succeq 0 \quad i = 0, 1, \ldots, n \\ \lambda_i^* g_i(x^*) &= 0 \quad i = 0, 1, \ldots, n \\ f(x^*) + \textstyle\sum_{i=0}^{n} \lambda_i^* \nabla g_i(x^*) + \sum_{j=0}^{m} \nu_j^* \nabla h_j(x^*) &= 0, \end{aligned} \tag{2.12}$$

The above says that for any optimization problem with differentiable objective and constraint functions, any pair of primal and dual optimal points must satisfy the KKT conditions when strong duality holds. If the original problem is convex, and the primal and dual points satisfy the KKT conditions, then these points are optimal and the duality gap is zero. This is a sufficient condition for optimality in the case where the original problem is convex. If the original problem is convex with differentiable objective and constraint functions that satisfy Slater's constraint qualification condition, then the KKT conditions provide necessary and sufficient conditions for optimality [33]. In some special cases, it is possible to solve the KKT conditions, and hence the optimization problem analytically. In others, algorithms that solve the KKT conditions can be used to solve the original optimization problem.

2.1.5 Convex Optimization

Convex optimization efficiently solves problems whose objective functions and constraints are convex functions and convex sets. Consider the constrained optimization problem posed in Equation (2.2). The requirement that the feasible constraint sets be convex requires that the inequality constraint functions g_i are convex, and the equality constraint functions h_j are linear. The latter requirement is highly problematic because it is too restrictive. Fortunately, for most circuit optimization problems, equality constraints are not present in the problem formulation. The application of convex optimization to the optimum design of digital and analog circuits has been explored by a number of researchers over a period of 20 years [21, 34, 35, 36, 37].

Convexity of the problem formulation has implications on iterative gradient descent methods described earlier. If the optimization problem is convex, then it can be shown that the unconstrained minimization using a gradient-based descent method will converge, and once the gradient $\nabla f(x)$ is small at a particular point, then the point is nearly optimal [33]. Convergence using Newton's Method is also very fast [33]. Constrained minimization as in Equation (2.2) can be solved in a number of efficient ways. One of the most efficient ways is to use interior point methods [33].

The biggest challenge in choosing to solve an optimization problem using convex optimization is to formulate the problem such that the objective function and feasibility sets are convex. In general, most problems are not convex by nature; however, if enough effort is spent in reformulating and transforming the problem such that it becomes convex or is approximated as a convex problem, then it can be solved extremely efficiently using the algorithms described in the preceding discussion. A class of problems that are not convex in their natural form but can be massaged into a convex formulation will be discussed shortly. However, first a discussion on solving constrained minimization problems is required.

2.1.6 Interior Point Iterative Methods

Interior point iterative methods are used to solve convex optimization problems that include inequality constraints, as in Equation (2.2) but with only one equality constraint of the form $Ax = b$ where A is a p by n real matrix with rank $A = p < n$. It is assumed that an optimum x^* exists. It is also assumed that the problem is strictly feasible; that is, there exists some x in the domain of f that satisfies $Ax = b$ and $g_i(x) < 0$ for $i = 0, 1, \ldots, n$. Thus, Slater's constraint qualification holds and there exists a dual optimal (λ^*, ν^*) such that together with x^*, the KKT conditions are satisfied.

Interior point methods apply Newton's Method to solve the optimization problem, (or equivalently, the KKT conditions) in an iterative manner. One example of an interior point algorithm is the *barrier method*. The barrier method approximates the inequality constrained problem as an equality constrained problem and applies Newton's Method to

iterate to a solution. The inequality constraints are made implicit in the objective function by applying an indicator function as follows:

$$\text{minimize } f(x) + \sum_{i=0}^{n} I(g_i(x)) \text{ subject to } Ax = b \tag{2.13}$$

where $I : \Re \rightarrow \Re$ is the indicator function defined as:

$$I(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0 \end{cases} \tag{2.14}$$

Since the indicator function is not differentiable, it must be approximated so that Newton's Method can be used to iterate to a solution. The Logarithmic barrier method uses the log function to approximate the indicator function given in 2.14:

$$\hat{I}(u) = -(1/t)log(-u) \text{ where t¿0} \tag{2.15}$$

The parameter t sets the accuracy of the approximation. The approximated indicator function in Equation (2.15) is convex and nondecreasing, and is ∞ for $u > 0$ [33]. Substituting Equation (2.15) into Equation (2.13) results in a differentiable convex optimization which can be solved using Newton's Method. Since this optimization is only an approximation of the original problem, it is important to understand how well the resulting solution approximates the actual optimum. In [33], the authors show that as t grows large, the approximation improves; however, when t is large, it is difficult to minimize by Newton's Method. Hence, a sequence of problems is solved by increasing t slightly at each step of the iteration and using the previous solution as the starting point for the next iteration.

There are many types of problems that are of the form in Equation (2.13) which have twice differentiable objective and constraint functions. These are: linear programming problems, quadratic programming problems, quasi-convex quadratic programming problems, and geometric programs in convex form. The first three are self-explanatory. Geometric programs are discussed next. Many other problems that do not have the required form can be reformulated and transformed such that they fit the required optimization template.

2.1.7 Geometric Programs

Geometric programs refer to a family of optimization problems that are not convex but can be massaged into convex optimization problems via a change of variables and a transformation of the objective and constraint functions.

A *monomial function* is defined as:

$$f(x) = cx_1^{a_1}x_2^{a_2}\dots x_n^{a_n} \tag{2.16}$$

where $c > 0$ and $a_i \in \Re$. A *posynomial* is the sum of monomials:

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}}x_2^{a_{2k}}\dots x_n^{a_{nk}} \tag{2.17}$$

Posynomials are closed under addition, multiplication, and nonnegative scaling; monomials are closed under multiplication and division [33].

An optimization of the form:

$$\text{minimize } f(x) \text{ subject to } \begin{cases} x \in \Re^n \\ g_i \leq 1 \quad i = 0\dots n \\ h_j = 1 \quad j = 0\dots m \end{cases} \tag{2.18}$$

where $f(x)$ and $g_i(x)$ are posynomials and h_j are monomials is called a geometric program. The domain of the problem is the positive reals, and the constraint $x \succ 0$ is implicit. Geometric programs are not convex in general but can be transformed into a convex optimization via a change of variables. In order to accomplish this, first set $y_i = log(x_i)$ so $x_i = e^{y_i}$. Then $f(x) = e^{a^T+b}$ where $b = log(c)$. The objective function, equality and inequality constraint functions in Equation (2.18) can be transformed in this way to yield an exponential of an affine function in the case of the equality constraints and sums of exponentials of affine functions in the case of the objective and inequality constraint functions [33]. Next, by taking the logarithm of the objective, inequality constraint, and equality constraint functions, the result is a convex optimization problem which is referred to as a

geometric program in convex form [33]. This optimization problem is given as:

$$\begin{array}{lll} \text{minimize} & \tilde{f}(y) = log(\sum_{k=1}^{K} e^{a_k^T y + b_k}) & \\ \text{subject to} & \tilde{g}_i(y) = log(\sum_{k=0}^{K_i} e^{a_{ik}^T y + b_{ik}}) \leq 0, & i = 0, 1, \ldots, n \\ & \tilde{h}_j(y) = e^{d_j^T y + w_j} = 0, & j = 0, 1, \ldots, m \end{array} \tag{2.19}$$

The functions $\tilde{f}$ and $\tilde{g}_i$ are convex and $\tilde{h}_j$ are affine, the resulting problem given by Equation (2.19) is convex.

Geometric programs have been used extensively to optimize gate size in custom circuit design [21, 34, 36, 37], where transistor or gate size is allowed to take on values from a continuous set.

2.2 Discrete Optimization

All previously discussed optimization techniques apply to continuously differentiable objective functions and constraints. In fact, up until now, only problems with a single objective function have been addressed. Unfortunately, as will be shown in Chapter 3, the problem that is posed in this thesis is significantly more complex because it is discrete and the feasibility sets and the multiple, conflicting objective functions are not necessarily convex. The variables in the problem addressed in this dissertation take on discrete values in a finite set. Hence, it is a discrete optimization problem. A large number of practical problems fall into this category and there are many diverse methods for solving them. A subset of these methods rely on the solution of continuous optimization subproblems and duality.

Some general problems that fall into the category of discrete optimization are integer-constrained network optimization, unimodal problems, generalized assignment and facility location problems, the traveling salesman problem, and separable resource allocation problems [32]. *Branch and bound* is an exhaustive search method that can be used to produce an optimal solution. It relies on upper and lower bound estimates of the optimal cost. The upper bounds are usually obtained via heuristics and the lower bounds are obtained through *integer constraint relaxation* or via *Lagrangian relaxation* using the weak duality theorem.

A second general mechanism of solving discrete optimization problems to first make them continuous and formulate them as known types of problems such as convex optimization, solve them, and then use heuristics to make the solution discrete. This will not necessarily produce the optimal but it may come close; in some cases it is possible to determine how close the estimated solution is to the actual [33].

Another important subset of algorithms used to solve difficult discrete optimization are known as *approximation algorithms*. The goal of these algorithms is to provide the best possible solution and guarantee that the solution satisfies certain properties [30]. These types of algorithms have recently seen an increase popularity due to their wide applicability to real-world problems.

2.2.1 Branch and Bound

The branch-and-bound method is an iterative exhaustive search method that explores the entire feasible set and enumerates each feasible solution. It can be very time-consuming (in some cases, it can become exponential in time) but in principle, it will yield the exact optimum. The basic principle is to partition the feasible set of solutions into smaller subsets and determine the lower and upper bound on the cost of the solutions in the subsets. If the cost lies outside the lower and upper bounds, then the subset is eliminated from further consideration. The progressive refinement of the feasible set is captured in an acyclic graph known as the *branch-and-bound-tree* [32]. The root of the tree contains the set of all feasible solutions. Singleton solutions are stored at the leaf nodes. Other nodes contain subsets of the entire feasible set stored at the root. At each non-terminal node, an algorithm exists that calculates the lower bound to the minimum cost over the subset at the node; and calculates the feasible solution in the subset that serves as the upper bound to the minimum cost over the subset. The bounds are saved and those nodes or subsets that contain solutions that fall outside the best current bounds are discarded.

Branch-bound-methods typically use continuous optimization to obtain lower bounds to the optimal costs and to construct feasible solutions [32]. It is important to ensure that the

lower bounds are as tight as possible for the branch-and-bound approach to succeed, since it leads to fewer iterations. Sometimes it is possible to reformulate a problem such that more constraints are added that accelerate the branch-and-bound solution and improve the lower bound, but do not affect the feasible set of solutions [32].

In the Lagrangian relaxation approach to obtaining lower bounds, constraints are made implicit in the objective function by forming the Lagrangian function, dual function, and dual optimization problem as discussed earlier. By the weak duality theorem, the dual value obtained through maximization of the dual function, provides a lower bound to the optimal primal value $f(x^*)$ as does the optimal dual value. Thus solving the dual problem gives a lower bound that can be used in the branch-and-bound procedure used to solve the original problem.

For a convex cost and linear inequality constraints, the lower bound obtained via Lagrangian relaxation is no worse than the lower bound obtained via constraint relaxation (continuous constraints). If both the cost function and constraint functions are linear then the lower bounds are equivalent [32]. It must be noted that using Lagrangian relaxation can lead to solutions that violate some of the constraints. Also, it may be difficult to maximize the dual function as it may be non-differentiable. In this case cutting plane and sub-gradient methods must be employed [32].

2.2.2 Approximation Algorithms

Approximation algorithms were formally introduced in 1966 by Ronald L. Graham [38] and used to generate near-optimal solutions to optimization problems that could not be solved efficiently. That is, they are usually used to solve NP-hard problems. Today, approximation algorithms are applied to many classical and new problems, including multi-objective combinatorial optimization [30], where exact polynomial time algorithms are known but not feasible due to large feasible solution sets. Unlike heuristics, which find reasonably good solutions in a reasonable amount of time, approximation algorithms provide provably good quality solutions with a provable bound on running time.

In some cases, it is possible to prove certain properties relating to the approximation of an optimum. For example, in ρ-approximation algorithms, it has been proven that the approximate solution $\hat{x}$ will not be more (or less, depending on the situation) than a factor ρ times the optimum solution x^* [39]. That is:

$$\begin{aligned} x^* \leq \hat{x} \leq \rho x^*, \quad & \text{if } \rho > 1 \\ \rho x^* \leq \hat{x} \leq x^*, \quad & \text{if } \rho < 1 \end{aligned} \tag{2.20}$$

The factor ρ is a relative performance guarantee. An approximation algorithm has an absolute performance guarantee or bounded error ϵ if it can be shown that:

$$(x^* - \epsilon) \leq \hat{x} \leq (x^* + \epsilon) \tag{2.21}$$

Approximation algorithms use all the concepts previously discussed in solving a particular problem. In this dissertation, an approximate economic equilibrium algorithm [30] is modified and applied to the hierarchical sensitivity-based optimization of ASICs. The problem is formulated as a convex optimization problem by exploiting the KKT conditions and using interior-point methods to iterate to a solution. The algorithm will be discussed in detail in Chapter 3.

2.3 Gradients, Sensitivity, and Optimality

Optimal circuit design requires knowledge of the energy-efficiency boundary which is comprised of the Pareto optima for the particular problem. The gradient of the energy-efficiency curve provides the designer with insight on how energy and delay change as the design variables are either lowered or increased in value. If the sensitivity to a design variable in the energy-delay tradeoff space is readily available, then the designer can systematically make sound decisions that tradeoff energy for delay in an iterative manner that will eventually lead to an energy-efficient design.

Sensitivity is defined as the ratio of the relative increase in energy and the corresponding relative gain in performance achieved by tuning a design parameter such as gate size or supply voltage [15, 16]. For example, if the energy-efficient curve for a circuit, with respect

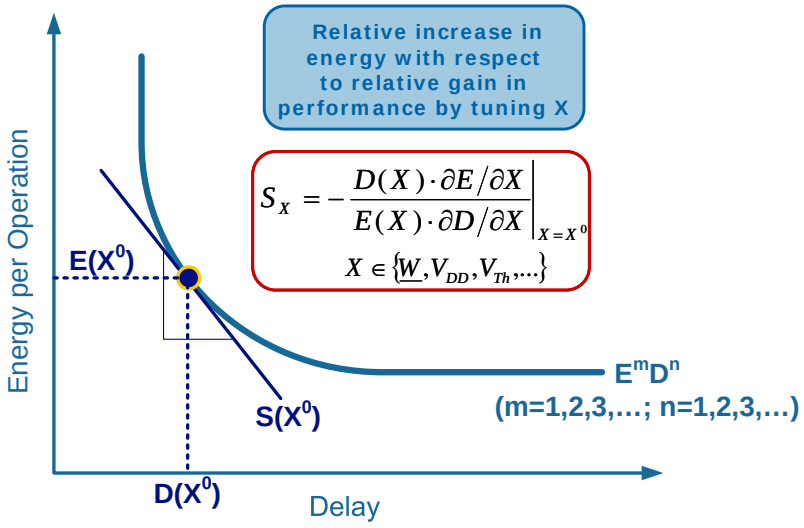


Figure 2.4. Definition of sensitivity

to a circuit tuning variable such as gate size, is plotted in the energy-delay coordinate space, then a specific value of sensitivity is the normalized derivative taken at a specific point on this curve, as shown in Figure 2.4. The energy-efficiency curve represents the Pareto optimal solutions set of the power-performance optimization.

Analytically, the sensitivity to a design parameter x is given as:

$$S(x) = -\frac{D\partial E}{E\partial D}\bigg|_x, \quad 0 \leq S(x) \leq \infty \tag{2.22}$$

Equation (2.22) is equivalent to the normalized gradient of the energy-efficiency curve of a particular block, with respect to the tuning variable. This is also termed *hardware intensity* in [15]. A simple interpretation of Equation (2.22) shows that $S(x)$ is the percent energy increase per percent improvement (reduction) in delay for an energy-efficient design [15]. The absolute gradient with respect to gate sizing, or the tangent of the energy-efficiency curve at a particular point is defined as follows [12]:

$$\Theta(x) = -\frac{\partial E}{\partial D}\bigg|_x, \quad 0 \leq \Theta(x) \leq \infty \tag{2.23}$$

In order to make a clear distinction between Equation (2.22) and Equation (2.23), the absolute gradient (or tangent) of the energy efficiency curve will be termed *absolute*

sensitivity, Θ. Analytical expressions for the absolute gradient can be derived if a few simple assumptions are made such as fixing the input capacitance of a combinational block. A derivation based on the Alpha-Power Law MOSFET Model [40] is provided in [12].

The authors in [11] state that the absolute gradients with respect to different tuning variables must be equal for an optimal design. They claim (but never show) that this is equivalent to the optimality condition in [15] which states that the hardware intensities with respect to different tuning variables must be balanced. In fact, for a single block, where two (or more) tuning variables, X and Y are employed to tune a block, the optimum condition using normalized gradients is equivalent to the optimum condition using absolute gradients. This is illustrated by comparing the optimality conditions for both absolute and normalized gradients in Equation (2.24).

$$\begin{aligned} \Theta(X) = \Theta(Y) \quad &\Rightarrow \quad -\tfrac{\partial E}{\partial D}\big|_X = -\tfrac{\partial E}{\partial D}\big|_Y \\ S(X) = S(Y) \quad &\Rightarrow \quad -\tfrac{D(X)}{E(X)} \cdot \tfrac{\partial E}{\partial D}\Big|_X = -\tfrac{D(Y)}{E(Y)} \cdot \tfrac{\partial E}{\partial D}\Big|_Y \end{aligned} \tag{2.24}$$

As shown in Equation (2.24), the optimality condition for both cases are equivalent if the energy and delay points (at the optimal point) on the X and Y energy-efficient curves are equal. This will be the case for a single block that is being optimized using two or more tuning variables. The optimality condition implies that the energy-efficiency curves are tangent to one another and have the same slope at the optimal point. If a curve is generated where two variables are varied (for example, supply voltage and gate sizes), then at each point on the aggregate energy-efficiency curve, the sensitivity to both tuning variables will be balanced.

In Figure 2.5, if two variables X and Y are tuned for a block, and the sensitivity of Y in the energy-delay space is greater than the sensitivity of X as shown by the initial design point $[D(X^0, Y^0), E(X^0, Y^0)]$, then decreasing Y, from Y^0 to Y^1, and increasing X from X^0 to X^1 results in a more energy-efficient design for the same delay; this is given by design point $[D(X^1, Y^1), E(X^1, Y^1)]$, where sensitivities are balanced.

In the case where the energy-efficiency curve for the system-level block is available and multiple tuning variables are employed at the same level of hierarchy, then the optimality

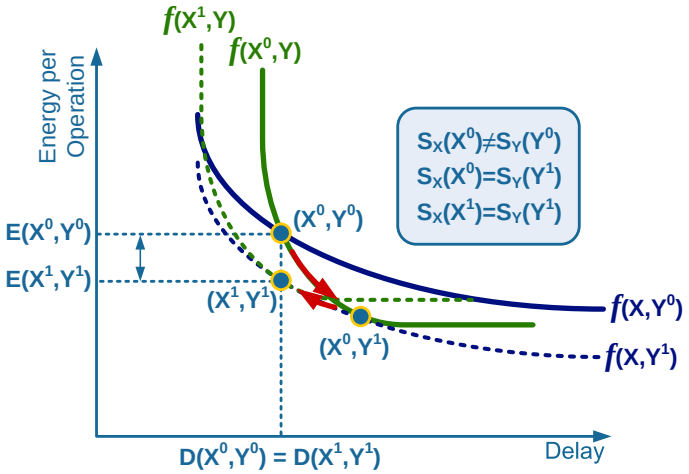


Figure 2.5. Circuit optimality using multiple tuning variables

condition based on absolute gradients is equivalent to the optimality condition using normalized gradients, or hardware intensity. Optimization that exploits the tuning variable with the largest capability for energy reduction will eventually lead to the optimal point where the energy-reduction potentials of all tuning variables are balanced [16, 15, 11]. This is intuitive: energy can be maximally reduced by decreasing the tuning parameter with the larger sensitivity, and performance can be maintained by increasing the parameter with the smaller sensitivity, hence resulting in an overall energy reduction. An optimization that iterates using this reasoning will eventually reach a fixed point where sensitivities to all tuning variables are balanced.

An example from [28] is reproduced here to illustrate in detail how balancing sensitivities to circuit tuning variables leads to maximum energy reduction. In Figure 2.6, the energy-efficiency curves are given for: varying gate size while supply voltage and pipeline depth remain fixed (curve A); varying gate sizes and supply voltage, while pipeline depth remains fixed (curves B, C, and D); and varying gate sizes and pipeline depth, while supply voltage remains fixed (curve E). According to [28], these curves are derived such that the sensitivities to the various tuning variables are balanced at each point on the energy-efficiency curve.

If the initial design point is given by the 14 FO4 point on the curve which meets the target cycle time, D_0, then the objective is to find the best method to maintain performance

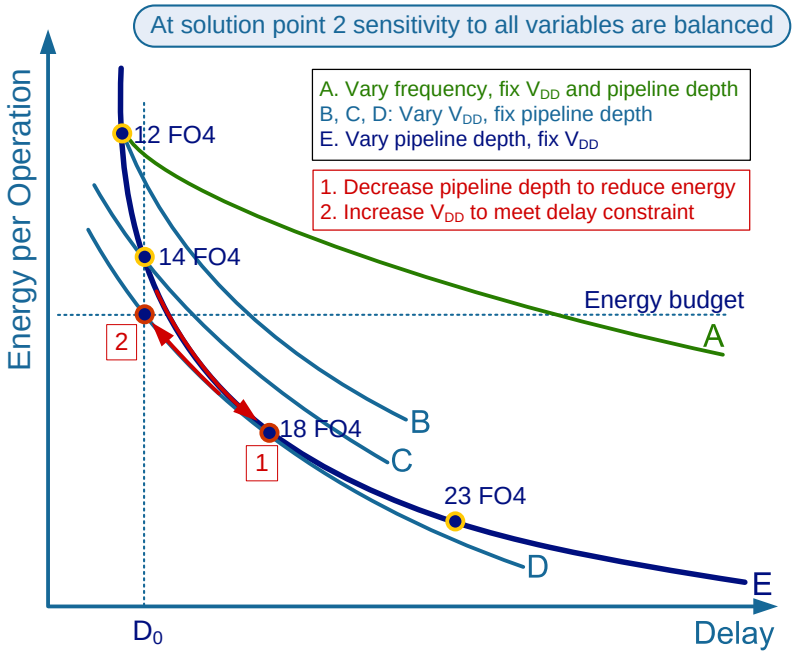


Figure 2.6. Multi-variable optimization using sensitivity balancing

while meeting the energy budget. Since the sensitivity to tuning pipeline depth has the highest sensitivity to energy reduction, the first step is to decrease pipeline depth to reach the 18 FO4 point labeled by the boxed number 1 on Figure 2.6. At the 18 FO4 point, the tuning variable with lowest sensitivity is supply voltage. Hence, supply voltage is increased to meet the target performance at the given energy budget, at the point labeled by boxed number 2. At the optimal point that meets the energy and delay targets, the sensitivities to sizing, pipeline depth, and supply voltage are balanced.

2.3.1 Sensitivity and Design Hierarchy

At the system level, the optimality condition of balanced sensitivity must be slightly modified as different components of a design will contribute differently to the total energy and the total delay. The sensitivity of each block in a system will vary as each component contributes different amounts of complexity to the overall system [15, 16]. For example, one cannot expect that an adder will have the same hardware intensity as a multiplier or

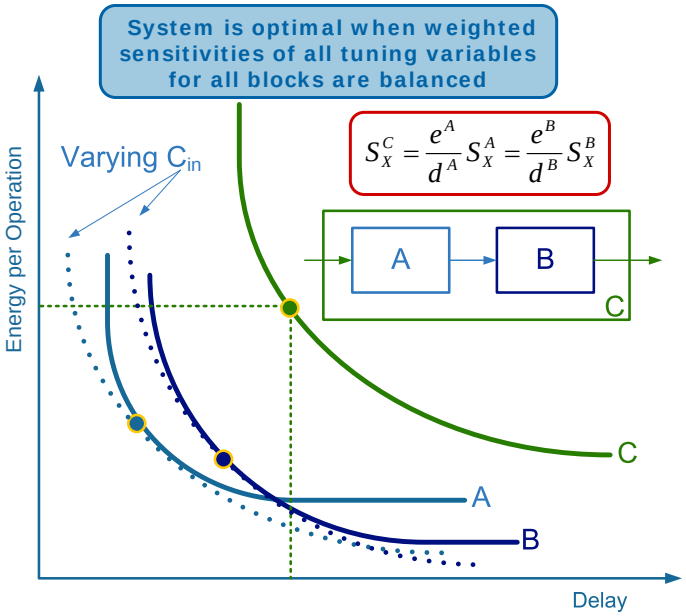


Figure 2.7. System optimality

a floating point unit (FPU) as the total effective switched capacitance of an adder is very different from a multiplier or FPU.

At the system level, the optimality conditions are derived using *aggregate sensitivity* which gives the sensitivity of a system to a tuning variable in terms of a function of weighted sensitivities of its components. For example, consider system C which is a series connection of two blocks, A and B, as illustrated in Figure 2.7. The optimal system design is achieved when the sensitivity of C, with respect to tuning variable X, is equivalent to the weighted sensitivities of block A and block B [26]. For example, if a block A and block B have differing complexity, are designed independently and each requires a different percentage of total cycle delay (e.g. latch will use a different portion of the cycle delay than the logic), then each block will have a different sensitivity to sizing. The block that has the higher delay weight and lower energy weight will need to be designed more aggressively. If it is assumed that the sensitivity to supply voltage is given as 2, then the optimality condition using sensitivity balancing implies that the sensitivity to sizing for the entire system should also be 2. However, this does not imply that the hardware intensity of block A or block B must also be 2 because each contributes a differing amount to the system energy and delay.

In the case where gate sizing and supply voltage are jointly optimized, the system optimality condition is given by: $S_C(W) = S_C(V)$ where $S_C(V)$ is the sensitivity to supply voltage and $S_C(W)$ is sensitivity to gate sizing. Since C is composed of two blocks, A and B, the optimality conditions for each block when designed independently are given by: $S_A(W) = S_A(V)$ and $S_B(W) = S_B(V)$. However, since A and B contribute differently to total system energy, their contribution to aggregate sensitivity must be appropriately scaled resulting in the following system optimality conditions [26].

$$\begin{aligned} e_A &= \frac{E_A}{E_C} \\ e_B &= \frac{E_B}{E_C} \\ d_A &= \frac{D_A}{D_C} \\ d_B &= \frac{D_B}{D_C} \end{aligned} \tag{2.25}$$

$$\begin{aligned} S_C(W) &= \frac{e_A}{d_A} S_A(W) = \frac{e_B}{d_A} S_B(W) \\ S_C(V) &= \frac{e_A}{d_A} S_A(V) = \frac{e_B}{d_A} S_B(V) \end{aligned}$$

The weights are the ratio of the contribution of the energy of each block to the total energy and the contribution of each block to the total delay. The weights also depend on the number of times a particular block maybe instantiated in a system. In addition, for each different input capacitance, there will be a different energy-efficient curve for a particular circuit. The energy-efficiency curve that envelopes all the possible input capacitances is the overall energy-efficiency curve for a particular block, as shown in Figure 2.7.

The optimality conditions in Equation (2.25) can be expanded using the definition of sensitivity, and then simplified. The result is shown in Equation (2.26) which states that the tangents of the energy-efficiency curves for the system and its sub-blocks must be equal for an optimal system. This translates directly to balanced absolute gradients.

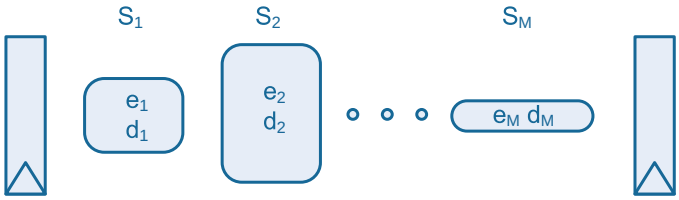


Figure 2.8. Composite pipeline stage

$$\begin{gathered}\frac{D_C}{E_C}\cdot\frac{\partial E_C}{\partial D_C}\bigg|_W = \frac{E_A/E_C}{D_A/D_C}\cdot\frac{D_A}{E_A}\cdot\frac{\partial E_A}{\partial D_A}\bigg|_W \\ \frac{\partial E_C}{\partial D_C}\bigg|_W = \frac{\partial E_A}{\partial D_A}\bigg|_W\end{gathered} \tag{2.26}$$

$$\Theta_C(V) = \Theta_C(W)$$
$$\Theta_C(W) = \Theta_A(W) = \Theta_B(W)$$
$$\Theta_C(V) = \Theta_A(V) = \Theta_B(V)$$

Composite Pipeline Stage

The results in Equations (2.25) and (2.26) hold for M blocks composing a single pipeline stage as shown in Figure 2.8. The optimality condition for aggregate sensitivity is given in terms of system block sensitivities which are summarized in Equation (2.27).

$$\begin{aligned}S_{agg}(V) = S_{agg}(W) = \tfrac{e_i}{d_i}\cdot S_i(W) = \tfrac{e_i}{d_i}\cdot S_i(V) \quad &\text{for } i = 1\ldots M \\ \Theta_{agg}(V) = \Theta_{agg}(W) = \Theta_i(W) = \Theta_i(V) \quad &\text{for } i = 1\ldots M\end{aligned} \tag{2.27}$$

Multi-stage Pipeline

In a more complex system where there are multiple stages in a pipeline as shown in Figure 2.9, the system optimality or aggregate sensitivity can be derived in a similar fashion as for a composite pipeline stage. In this case, it is assumed that each stage has the same delay.

First, assume that there are N pipeline stages, each with a different time slack available

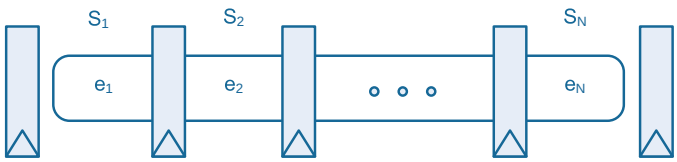


Figure 2.9. Multi-stage pipeline

and differing amounts of logic. Each stage is a single block followed by a register and contributes energy, E_i, to the total energy such that $E = \sum_{i=1}^{N} E_i$

The energy weight assigned to each block is giving by $e_i = E_i/E$ and $\sum_{i=1}^{N} e_i = 1$. The energy weight represents the fraction of the total system energy budget assigned to each stage. Each stage has its own sensitivity to sizing given by S_i. The aggregate sensitivity (e.g. to sizing or supply voltage) for the entire system is S_{agg}. The clock period for the system is given by D and each individual stage delay is $D_i = D$. Using the definition for sensitivity given in Equation (2.22), an increase in clock cycle by ∂D through retuning the circuits in all stages increases the total energy by:

$$\partial E = \sum_{i=1}^{N} \partial E_i = -\sum_{i=1}^{N} \frac{E_i}{D_i} S_i \partial D \tag{2.28}$$

Hence, by rearranging Equation (2.28) the aggregate sensitivity is given by:

$$S_{agg} = -\frac{\partial E \cdot D}{\partial D \cdot E} = \sum_{i=1}^{N} e_i S_i \tag{2.29}$$

Equation (2.29) gives a nice result as it says that the aggregate sensitivity for multi-stage pipeline can be expressed through the sum of the weighted sensitivities of its individual stages. If the same analysis is carried out using the definition of absolute sensitivity given in Equation (2.23), the result is somewhat different as shown in the following derivation which starts off similar to Equation (2.28).

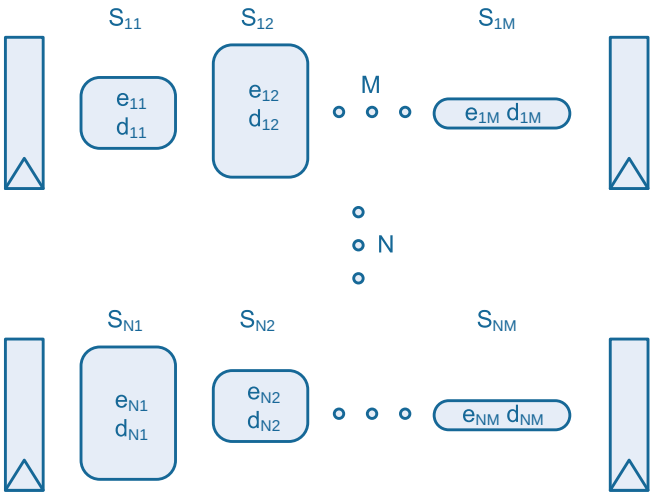


Figure 2.10. Multi-stage composite pipeline

$$\begin{aligned} \partial E &= \textstyle\sum_{i=1}^{N} \partial E_i \\ \partial E &= -\textstyle\sum_{i=1}^{N} \Theta_i \cdot \partial D_i \\ \Theta_{agg} &= -\frac{\partial E}{\partial D} = \frac{\sum_{i=1}^{N} \Theta_i \cdot \partial D_i}{\partial D} \\ \Theta_{agg} &= \textstyle\sum_{i=1}^{N} \Theta_i \end{aligned} \tag{2.30}$$

As is clear now from Equation (2.30), the absolute gradient definition of sensitivity does not lend itself well to composability of block sensitivities because they add in absolute terms. This is problematic. In an optimal system, aggregate sensitivity to sizing must be balanced to system sensitivity to supply voltage. If absolute gradients are used, then the definition of aggregate sensitivity supplied by Equation (2.30) would not account for the different amounts of complexity in each stage of the pipeline, and the contribution of energy from each stage to the system energy. In the remainder of this work, sensitivity will be defined using normalized gradients.

Multi-stage Composite Pipeline

A more general and realistic representation for a general system is one with multiple pipeline stages and each stage consisting of multiple blocks as shown in Figure 2.10. If

the energy for each block is given by E_{ij} and the delay is D_{ij}, then the system optimality conditions for a multi-stage composite pipeline can be derived using a similar analysis as for the preceding two cases. They are given as:

$$\begin{aligned} \frac{e_{ij}}{d_{ij}} \cdot S_{ij} &= \frac{e_{ik}}{d_{ik}} \cdot S_{ik} \quad 1 \leq j,k \leq M \\ S_{agg} &= \sum_{i=1}^{N} \frac{e_{ik}}{d_{ik}} \cdot S_{ik} \quad \text{for any sub-block k in stage i} \end{aligned} \tag{2.31}$$

When a system is optimal, the aggregate sensitivity is equal to the weighted sum of the sensitivities for each pipeline stage. The sensitivity of each pipeline stage is given in terms of the sensitivity of each component: it is equal to the weighted sensitivity of any one its components.

Hierarchical energy-delay tradeoff analysis requires calculation or estimation of sensitivity at each level of design hierarchy. This can be a compute intensive and over-whelming task if the design is large and there are numerous tuning variables. The results in Equations (2.27), (2.30), and (2.31) are important because they show that system optimality conditions can be easily expressed through sensitivities for smaller blocks. Hence, energy-efficiency curves for large systems can be estimated from smaller blocks without having to calculate or generate them through other tedious and time-consuming means.

2.4 Sensitivity-Based Design Methodology

Sensitivity can be used as a guide to systematically traverse the energy-delay design tradeoff space. The use of energy-efficient curves and sensitivities to evaluate the tradeoff between energy and delay under tight energy constraints has been applied in various custom circuit design frameworks [15, 11, 37]. The main idea behind the optimization methods is equalization of sensitivities at all levels of hierarchy for all tuning variables, as described in Section 2.3. For example, if circuit design tools can calculate sensitivities for various synthesized blocks, then they can use the sensitivity information to uncover opportunities for reducing power. For example, if there are substantial differences in sensitivity between blocks with respect to different tuning variables, then power-efficiency of a design can be

improved by adjusting the blocks and tuning variables until the sensitivities are balanced. An example of such a process was depicted earlier in Figure 2.6.

Methods for efficient power minimization at the circuit and micro-architecture levels that are based on energy and delay analytical models and derived analytical expressions for sensitivity of various circuit tuning variables is presented in [11]. The authors show that significant power savings can be obtained without a delay penalty when sensitivities to sizing, supplies, and thresholds are equalized.

Calculation of the sensitivities is the major challenge to implementing tools that use this type of sensitivity analysis in a design methodology. In [16] the author relies on simulation and calculation of derivatives to quantify the hardware intensity or energy-per-performance ratio (EPR) with respect to various design parameters such as supply voltage or device-to-wire capacitance ratio. The drawback of both of the above approaches is the inability to incorporate them easily in an automated hierarchical synthesis-based design flow commonly used for ASICs.

In [15] the energy-efficiency analysis is carried out through simulation of circuits over a range of values for the design parameter under consideration. Each simulation is tuned for a particular value of sensitivity and the circuit blocks are optimized accordingly. Designs that have multiple pipeline stages are optimal when the sensitivity is equalized across all stages. This condition is met through a series of tuning variable adjustments based on calculation of sensitivity from energy and delay data obtained via simulation and analysis tools.

The authors of [11] take this approach one step further. They derive analytical expressions for absolute gradients based on energy and delay models that are functions of circuit tuning parameters such as device size, supply voltage and threshold voltage. The expressions are derived for each individual gate in a circuit block. Each tuning variable is adjusted to achieve minimal energy at each stage in the circuit block. The adjustment is based on calculated sensitivities of each stage with respect to each design variable. A similar approach is used for blocks with multiple levels of hierarchy where sensitivity information from lower level blocks is used in the optimization at higher levels of abstraction

[11]. Unfortunately, these expressions can become complicated and involve energy and/or delay calculations and are specific for the sub-circuits used in the design.

In [37], the authors employ a different method. Rather than calculating sensitivities, they generate energy-efficient curves in the energy-delay tradeoff space for various adder architectures and implementation strategies. The tuning variable is device size and the curves are generated using software developed in Matlab. The underlying framework uses a static timing formulation based on tabulated delay models. The optimization problem is posynomial in nature and can be mapped onto a convex optimization problem easily. The software minimizes delay based on given energy constraints. Once the energy-efficient curves are generated, the designer can easily determine which design strategy will be best suited to the given energy and delay constraints. Sensitivities are inherently present in the generated curves since they are just derivatives calculated at various points on each curve; however, the software never explicitly calculates them.

Each of the above methodologies serves a designer well if it is a one-time optimization of a custom-designed circuit since each approach requires considerable execution time. In addition, some manual intervention by the designer is required at some or all stages of the optimization. Unfortunately, these methods would not be appropriate for a synthesis-based automated design environment commonly used for ASICs. However, as suggested in [16], if simple relationships between sensitivity and physical parameters of a circuit or design can be discovered and modeled accurately, they can be incorporated into a synthesis environment and used to guide optimization. Chapter 5 describes models for sensitivity that are derived from simple physical properties of circuit, obviating the need to calculate derivatives.

2.5 Summary

In summary, the best solution to a multi-dimensional, multi-objective combinatorial power-performance optimization is one that provides an energy-efficiency boundary which represents the Pareto optimal solution set. The challenge lies in constructing the boundary and traversing it systematically to arrive at the best solution that meets all the design

constraints. Sensitivity provides a means of evaluating the energy-efficiency of a design at any level of design abstraction with respect to any tuning variable. It represents the normalized gradient of the energy-efficiency curve at a particular point on the boundary. An optimal system is constructed by balancing sensitivities to all tuning variables, across all levels of design abstraction.

Sensitivity can be derived from analytical models or estimated by simulation. Sensitivity is also a by-product of iterative, gradient-based optimization methods. In the next two chapters, a hierarchical power-performance methodology is described that uses sensitivity analysis. Sensitivity is obtained in two ways: one is via models based on physical properties of a circuit or as a by-product of optimization algorithms based on iterative interior point methods.

Chapter 3

Hierarchical Power-Performance Optimization

Optimization at the architecture level can have a major impact on power.
– A. Chandrakasan and R. W. Brodersen, 1995

Power-performance optimization that spans all levels of design can yield significant reductions in power for a given performance target [41]. Chandrakasan and Brodersen showed that a design optimization spanning architecture, algorithm, micro-architecture, circuit, and technology resulted in three orders of magnitude power savings for their portable media terminal benchmark at the cost of increased latency and area [41]. Their power optimizations included low threshold devices, architecture driven voltage scaling, parallelism, gated clocks, power gating, accounting for switching activity in placement and routing, and in architecture design. Today, even though these techniques are now widely employed in optimizing designs for power and performance, they are still disjointly optimized. Architecture optimizations are rarely carried out in the context of technology or circuit constraints as the teams that design architecture, circuit, and technology are usually different. This leads to architectures that are not well-suited to the underlying algorithm, circuit style or technology. For example, if an architecture is memory intensive but is mapped to a technology that is not optimized for memory applications (i.e. high leakage cells), then the resulting system will be sub-optimal. This dissertation proposes integrating optimizations at various levels

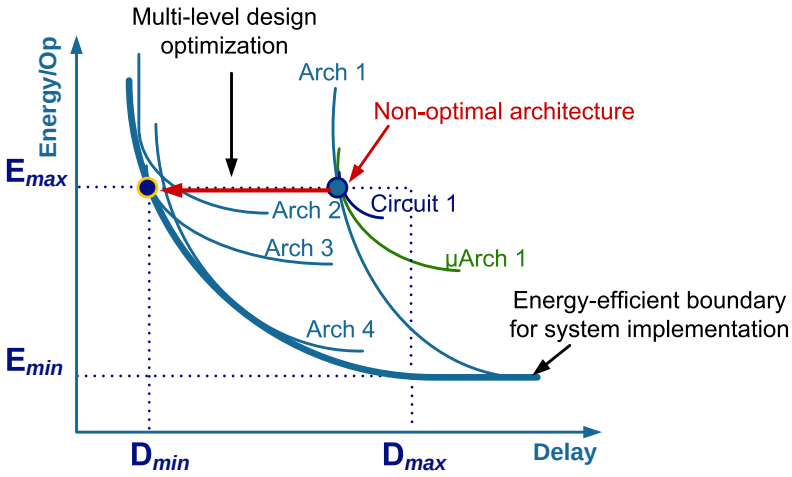


Figure 3.1. Energy-delay tradeoffs at multiple levels of design abstraction

of design hierarchy by propagating sensitivities to tuning variables at lower levels of design (e.g. circuit or technology) to higher levels of design (e.g. architecture). This mechanism allows designers to make power-performance tradeoffs at higher levels of design abstraction in the context of lower level constraints.

Chandrakasan and Brodersen present an excellent overview of various power optimization techniques available to the designer at each level of design abstraction, from technology to architecture [41]. A few of these techniques (i.e. gate sizing, pipelining, parallelism, power gating, and clock gating) are employed in the design of the benchmark presented in Chapters 6 to show how low-level constraints can affect architecture level choices. Power-performance optimization at the architecture level has the greatest potential for power reductions at a given performance target, hence it is important to design an architecture that is optimal in the energy-delay tradeoff space.

Fast architecture exploration over a large design tradeoff space in the context of lower level design constraints is one of the most important objectives of energy-delay optimization. For example, by plotting the energy-delay tradeoff curves at each level of design abstraction as illustrated in Figure 3.1, it can be deduced that Architecture 3 achieves higher performance than Architecture 1 for the given E_{max} energy constraint. The architecture composite curve is obtained by constructing energy-delay tradeoff curves for the

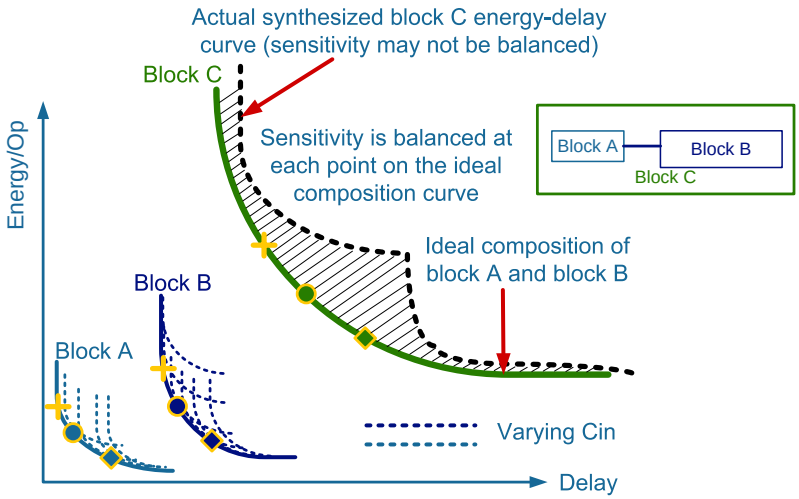


Figure 3.2. Gap between ideal and synthesized energy-efficiency boundaries

circuit tuning variables such as gate size and the tradeoff curves for the micro-architecture level tuning variables such as pipeline depth. The architecture tradeoff curve represents the points in the energy-delay space where the sensitivities of circuit tuning variables are balanced to micro-architecture and architecture tuning variables [16, 15]. These points are considered to be the Pareto optima for the given architecture.

The Circuit 1 tradeoff curve represents the circuit implementation of one of the system blocks at a particular supply voltage. At the intersection of the Circuit 1 tradeoff curve and the Micro-architecture 1 tradeoff curve, the sensitivity to gate sizing is equal to the sensitivity to pipeline depth. In Chapter 2, it was shown that this optimality condition can be captured in an equation that gives the optimal aggregate sensitivity as a function of sensitivities to tuning variables at lower levels of design abstraction, allowing construction of higher level energy-efficiency curves. Fast construction of architecture Pareto optima allows designers to rapidly determine the architecture that is best suited to given design constraints.

A second objective of power-performance optimization is to systematically narrow the gap between the ideal (or optimal) energy-efficiency boundary and the actual synthesized one, as shown in Figure 3.2. The figure shows the composition of two blocks, A and

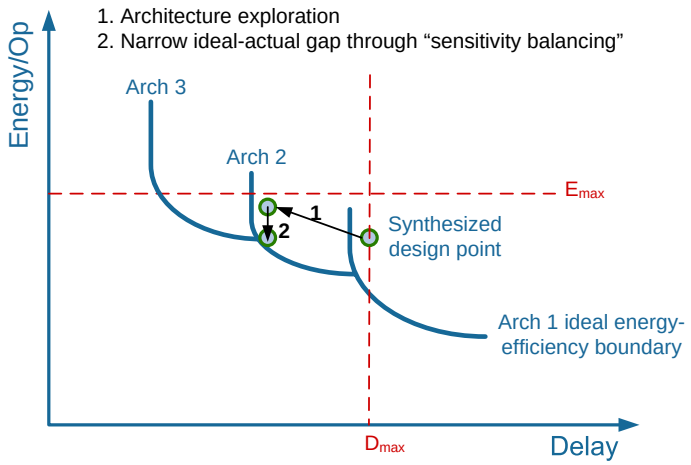


Figure 3.3. Two objectives of hierarchical optimization

B, to form the system level block, C. The energy-efficiency boundaries of A and B are composed according to the optimality rules described in Chapter 2 to generate the ideal energy-efficiency boundary of block C which is shown as the blue curve. When block C is synthesized, the energy-efficiency boundary generated is shown as the black dotted curve, where sensitivities may not be balanced. The goal of the hierarchical energy-delay optimization is to narrow the gap (depicted by the hashed space between the ideal and synthesized energy-efficiency curves) by balancing sensitivities as best as possible. It may not be possible to balance sensitivities exactly due to heuristic optimization, quantization effects, saturation of variables, and inaccurate estimation of energy, delay, and capacitance. However, it may be possible to balance sensitivities within a given threshold. This means that the optimization reaches a fixed point once sensitivities to all tuning variables are within a threshold, ϵ, of each other.

As seen from Figures 3.1 and 3.2, knowing the Pareto optima allows for improvement of the current solution by bringing it closer to the estimated optimum for particular energy and delay constraints. For example, in Figure 3.3, architecture exploration can be used to determine the optimum architecture for the given energy and delay constraints; and sensitivity balancing can be used to bring the synthesized solution closer to the Pareto optimum.

The construction of the optimal energy-efficiency boundary at the architecture level or at the system level requires energy-delay tradeoff information in the form of sensitivities to various design tuning variables for all blocks in the system. As mentioned in the previous chapter, the problem of constructing the Pareto optimal curve is a multi-objective combinatorial optimization problem with discrete variables; this implies that the constraints or feasibility sets may not necessarily be convex.

A common approach to solving the above problem (for example, in the case of gate sizing) is to flatten the hierarchy and transform the multiple objectives into a single objective such that delay is minimized subject to an energy constraint or vice versa. The problem is then relaxed into a continuous domain and transformed into a convex problem using models and variable transformations. Once a fixed point solution is reached, heuristics are used to return the solution to the discrete domain. Unfortunately, this approach does not scale very well to large number of tuning variables or multiple levels of design hierarchy.

The approach used in this work is somewhat different: hierarchy is preserved allowing for scalability. The methodology is both top-down and bottom-up as the greatest energy-efficiency in design is achieved when design decisions at the top level of hierarchy (architecture) are considered in the context of constraints at lower levels of hierarchy. This is achieved through *composition* which will be described in Section 3.2.1. Constraints from higher levels of hierarchy are propagated down to lower levels, and sensitivities of tuning variables to energy and delay are balanced upward through the various design abstraction layers.

Sensitivities to tuning variables such as gate size are not directly calculated but estimated or modeled through physical properties of the block. For example, in Chapter 5 details are presented on modeling sensitivity to gate sizing using physical properties of a circuit such as gate and wire capacitance. Once sensitivities for lower level blocks are modeled or estimated, composition rules are used to construct energy-efficiency boundaries for larger blocks at higher levels of design abstraction. Sensitivity can also be approximated using the parameter t of the barrier method used in interior point algorithms. Section 3.5 describes in detail why this is the case.

The next three sections describe the hierarchical design methodology in detail. Following this, a convex formulation of the problem is presented using an approximation algorithm. This formulation is a starting point for implementation of the methodology as an automated design tool that may be used to augment existing electronic design automation (EDA) tools.

3.1 Design Methodology Overview

The design methodology consists of two distinct stages. The first stage constructs the Pareto optima points in the form of optimal aggregate sensitivities for a given architecture in the energy-delay space. This may entail any number of points depending on whether there is a specific sensitivity that is being targeted or whether a designer requires an energy-efficiency boundary that spans a range of energy and delay constraints.

The energy-efficiency boundaries of various architectures are then used for fast architecture exploration to determine the best architecture for the given design constraints. Once the architecture becomes fixed, the second stage of the optimization attempts to narrow the gap between the ideal energy-efficiency boundary and the actual synthesized design point. This is done by trying to balance sensitivities across blocks and design tuning variables using the rules described in Chapter 2.

The input to the design methodology is a hierarchical netlist and design constraints such as input slopes, input loads, output loads, energy budgets and delay targets. After the netlist and constraints are parsed, they are converted into hierarchical constraint graphs [42] that store connectivity information along with constraint information on each of the nodes. The edges store maximum wire capacitance constraints so that long wires are avoided.

Using a fast convex optimizer such as one described in [17], models for sensitivity are constructed for a small subset of leaf cells. These are the basic building blocks of the system (e.g. adder, register). Once these are available, sensitivity for lower level blocks are estimated using the models and energy-efficiency tradeoff curves for larger blocks are generated using composition rules for optimal aggregate sensitivity. The energy-efficiency

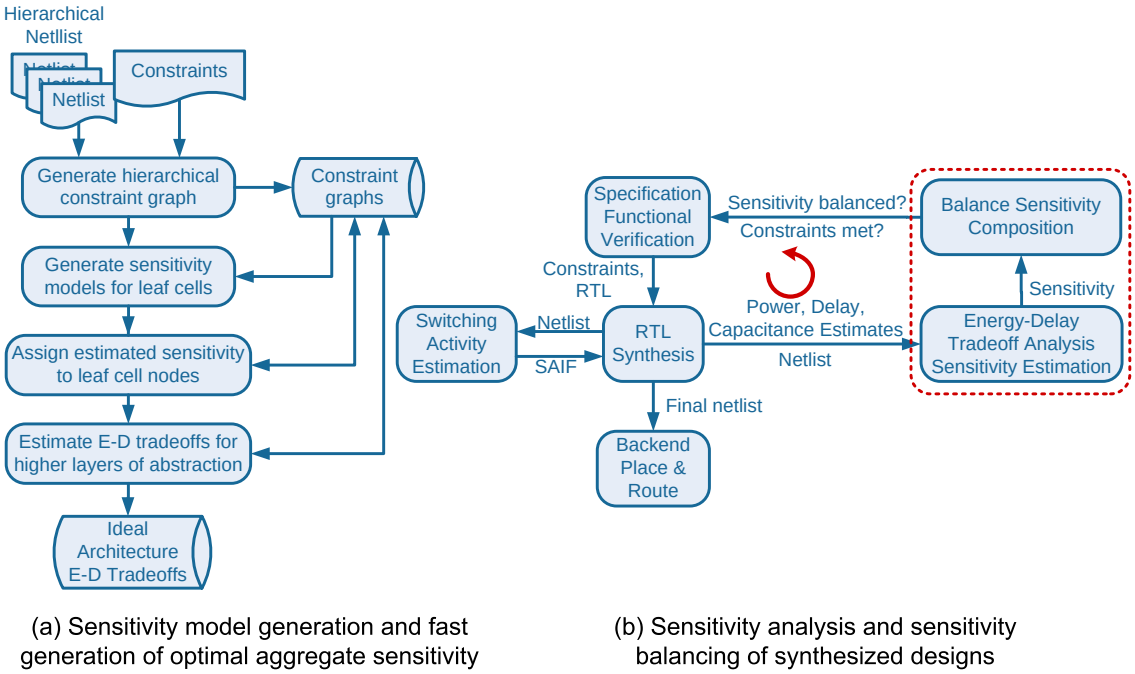


Figure 3.4. Fast architecture exploration

curves at all levels of design abstraction take into account connectivity, throughput, power, and capacitance constraints.

Figure 3.4(a) depicts the flow just described. In Section 3.4.1, constraint graphs and their generation are described in detail. Sensitivity models can be generated from physical circuit parameters as exemplified in Chapter 5. The flow in Figure 3.4(a) is required to run only once at the start of a project. The generated energy-delay tradeoffs can then be used any number of times in optimizing a system using sensitivity analysis. Figure 3.4(b) shows how sensitivity analysis and sensitivity balancing is incorporated into a standard synthesis flow. The process is an iterative one. Once the synthesized netlist is available along with physical parameters (e.g. wire and gate capacitance) of the synthesized block, sensitivity of the block is estimated using sensitivity models generated in the first phase of design (Figure 3.4(a)). If the aggregate sensitivities across blocks and design parameters are balanced (within a given threshold) according to the rules defined in Chapter 2, then synthesis concludes and the best possible design is output.

3.2 Fast Architecture Exploration

Fast architecture exploration is the first phase in the proposed design methodology (Figure 3.4(a)). It requires generation of energy-efficiency boundaries at all levels of design: from technology through to architecture. This can be a time consuming and cumbersome task if previously published methods such as simulation [15] or analytical models [11] are used. Fortunately, there is another means of generating sensitivity information obviating compute intensive tasks. Physical properties of a circuit can be used to estimate sensitivity to a first order. For example, Chapter 5 shows that there is a first order linear relationship between sensitivity to sizing and the ratio of total gate capacitance to total wire capacitance.

Since architecture selection is done very early on in the design process, the sensitivity models need only be accurate to a first order to allow a designer to quickly understand the tradeoffs of various choices in the energy-delay space. Once an architecture is selected that best meets the design constraints, then finer-grained optimization can be used to further reduce power for a given performance target.

Fast architecture exploration benefits from generation of energy-efficiency boundaries at the architecture level that include lower level constraints. The inclusion of lower level constraints such as technology or sizing information can impact choice of optimal architecture. Unfortunately, architecture tradeoffs in the context of lower level constraints requires tradeoff curves for all architecture components. If multiple tuning variables such supply voltage and threshold voltage are used in addition to circuit sizing, then each of the tradeoff curves for those variables need to be generated and then the composite curve will need to be constructed where sensitivities are balanced. If these tasks are performed for multiple architectures, then the entire analysis process may take a long time unless there is a faster way to obtain the architecture tradeoff curves.

The mechanism proposed here requires only sensitivity models for lower level building blocks. As mentioned earlier in Chapter 2, balancing block sensitivities for system optimality means that the optimal aggregate sensitivity is either equal to a weighted sum of block sensitivities or is equal to the "normalized" block sensitivity. The weight or normalization

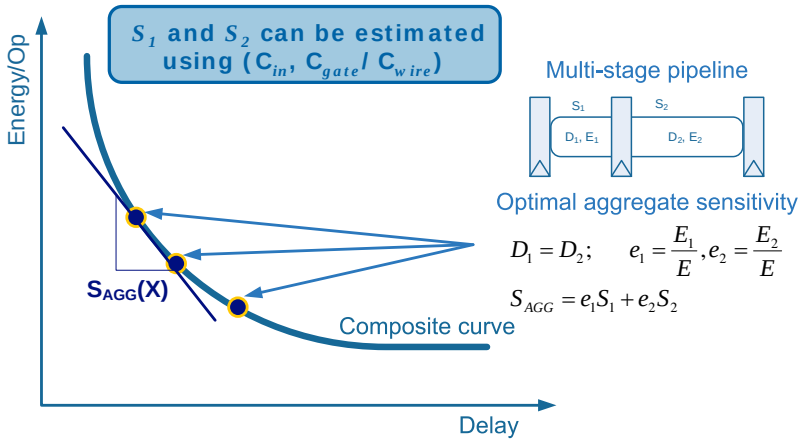


Figure 3.5. Relationship between optimal aggregate sensitivity and system energy-efficiency boundary

factor for a block is based on the ratio of its contribution to the total energy and to the total delay of a system. These balancing conditions described in Chapter 2 lead to composition rules for optimal aggregate sensitivity. General composition rules were described in Chapter 2. The next section describes design composition via a filter example.

3.2.1 Design Composition

Hierarchical composition is a key ingredient to the design methodology that allows it to scale to very large designs. Design composition rules for a particular circuit are based on calculation of optimal aggregate sensitivities as outlined in Chapter 2. The set of optimal aggregate sensitivities provides an ideal energy-efficiency boundary (or Pareto optimal set) for the system. The boundary can be constructed for a synthesized design or a design optimized using custom continuous gate sizes and/or supply voltages and/or threshold voltages. The optimization depends on which models (synthesized or custom) are used at the building block level. Composition rules for each system must be derived using the general rules given in Chapter 2, for constructing optimal aggregate sensitivity. Figure 3.5 gives a pictorial view of the relationship of optimal aggregated sensitivity to the ideal energy-efficiency boundary of a system that is constructed from a multi-stage pipeline. The optimal aggregate sensitivity conditions are given on the right side of the figure which

N-TAP FILTER	DELAY MODEL	ENERGY MODEL	OPTIMAL AGG. SENSITIVITY
Transpose	D_{MAC}	$(N-1)E_{MAC} + E_{mult}$	$\frac{e_{MAC}}{d_{MAC}} S_{MAC} + \frac{e_{mult}}{d_{mult}} S_{mult}$
Transverse	$D_{mult} + (N-1)D_{add}$	$N(E_{mult} + E_{add}) + (N-1)E_{reg}$	$\frac{e_{mult}}{d_{mult}} S_{mult} = \frac{e_{add}}{d_{add}} S_{add} = \frac{e_{reg}}{d_{reg}} S_{reg}$
P-Parallel Transpose	$D_{Transpose}/P$	$P \cdot E_{Transpose}$	$S_{Transpose}$
Pipelined Transverse	D_{MAC}	$(N-1)E_{MAC} + E_{mult}$	$\frac{e_{MAC}}{d_{MAC}} S_{MAC} + \frac{e_{mult}}{d_{mult}} S_{mult}$

Table 3.1. Composition Rules for Filters

provide a means to calculate optimal aggregate sensitivity for the system. The optimal aggregate sensitivity is the normalized gradient of the system energy-efficiency boundary. This is the Pareto optimal set of design points for the entire system.

Table 3.1 shows the composition rules derived for a set of different filter architectures. The number of taps is N. For example, the first row in the table is calculated by observing that the critical path in the transpose filter is the delay of a single multiply-accumulate (MAC) block. The total energy of the filter is addition of $(N-1)$ MACs and a single multiplier. The transpose filter is a multistage pipeline system; so using the generic rules presented in Chapter 2, the optimal aggregate sensitivity is the addition of the weighted sensitivities of the MAC and multiply blocks. The weights are dependent on the energy and delay contribution of each block to the total energy and delay of the filter. The weights for the transpose filter would be as follows: $e_{mult} = \frac{E_{mult}}{(N-1)E_{MAC}+E_{mult}}$ and $d_{mult} = \frac{D_{mult}}{D_{MAC}}$ for the multiplier; and $e_{MAC} = \frac{(N-1)E_{MAC}}{(N-1)E_{MAC}+E_{mult}}$ and $d_{MAC} = \frac{D_{MAC}}{D_{MAC}} = 1$ for the MAC. If the delay and energy of a multiplier is approximated to be almost the same as the MAC (i.e. delay and energy of the register and adder are negligible compared to the multiplier), then the weights can be approximated as: $e_{mult} = \frac{1}{N}$, $d_{mult} = 1$, $e_{MAC} = \frac{(N-1)}{N}$, and $d_{MAC} = 1$. The generic composition rules of the composite pipeline stage given in Chapter 2 are used to obtain the optimal aggregate sensitivity for the transverse filter architecture.

It should be noted that leakage energy and dynamic energy must be calculated separately and then accumulated. Activity factors must also be taken into account when estimating dynamic energy. These composition rules provide a means to construct the energy-efficiency

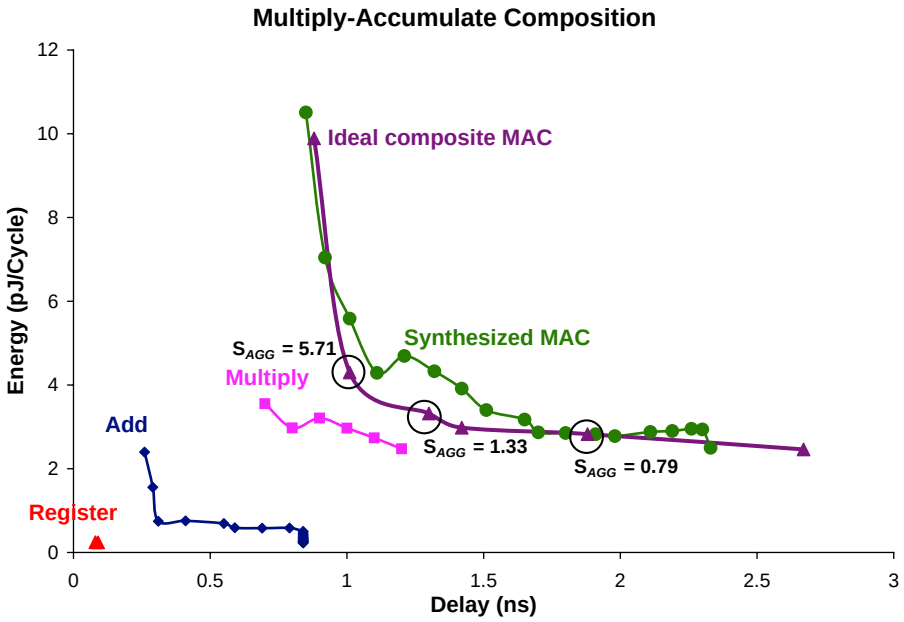


Figure 3.6. Design composition of multiply-accumulate (MAC) block

boundary for any filter architecture. An example is given in Figure 3.6 and Figure 3.7. Figure 3.6 shows the construction of the ideal energy-efficiency boundary for the MAC block from energy-delay tradeoff curves for multiply, add, and register blocks. The ideal energy-efficiency boundary is based on calculating optimal aggregate sensitivity. The figure shows different values of optimal aggregate sensitivity on the energy-efficiency boundary. This boundary is compared to the energy-delay tradeoff curve obtained from direct synthesis.

Figure 3.7 shows the calculation of the ideal energy-efficiency boundary for a 32-tap transpose filter using the results of composition for the multiply-accumulate block. The ideal composition curve is compared with the synthesized results. The composition must take into account leakage energy and dynamic energy, and include best estimates for activity factors. Figure 6.13 in Chapter 6 shows the entire filter architecture tradeoff space for 32-tap filters which is generated using this methodology.

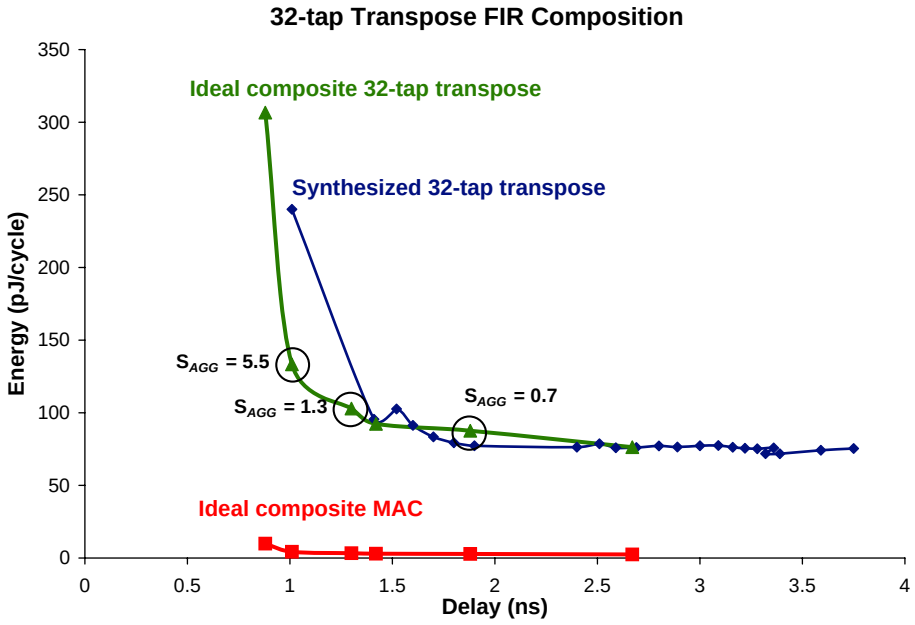


Figure 3.7. Design composition of 32-tap transpose filter from MAC composite curve

Design Composition Flow

Since it is not exactly clear from the figures and table, how the composition process works, it is useful to describe it through the composition for a 32-tap transpose filter. Once the composition rules are derived as shown in Table 3.1, the optimal aggregate sensitivity is calculated based on the derived equations. The optimal aggregate sensitivity gives a relationship between the system sensitivity and the sensitivities of the system components. As long as this relationship is satisfied, the sensitivities are ideally balanced and the design is optimal. Once the sensitivities for the MAC and the multiplier are available either through simulation or modeling, then blocks are composed such that the relationship dictated by the optimal aggregate sensitivity equation holds.

The first step is to derive the optimal aggregate sensitivity for the MAC block. The optimal aggregate sensitivity for the MAC is based on the composite pipeline stage composition rules. The sensitivity of the MAC is equal to the weighted sensitivities of the add, multiply, and register blocks. The optimal aggregate sensitivity of the MAC is given by: $\frac{e_{reg}}{d_{reg}}S_{reg} = 1.77$ or $\frac{e_{add}}{d_{add}}S_{add} = 1.66$ or $\frac{e_{add}}{d_{add}}S_{add} = 1.72$. Note that the sensitivities are not

BLOCK	DELAY (ps)	ENERGY (pJ)	SENSITIVITY
Register	0.09	0.24	1.03
Add	0.59	0.58	2.53
Multiply	1.2	2.48	1.25
MAC	1.88	2.82	$\tilde{1}.7$

Table 3.2. Multiply-accumulate Block Composition

exactly balanced, but they are all within a 0.06 ($\tilde{6}\%$) threshold. Also, note that the energy is not a simple addition of the block energies as activity factors and leakage energy are accounted for differently when composing designs (Chapter 4 discusses this further). The results of the composition procedure is summarized in Table 3.2 for one point on the curve shown in Figure 3.6.

The 32-tap transpose filter composition is now easy given the ideal optimal aggregate sensitivities for the MAC and the derived rules in Table 3.1. In Figure 3.7, the contribution of the single multiplier is negligible compared to 31 MAC blocks. For the example point, the aggregate sensitivity for the filter will be approximately the same as $S_{MAC} = 1.7$ block since $e_{MAC} \approx 1$ and $d_{MAC} = 1$.

Limitations

Achieving the optimal aggregate sensitivity might not be possible in practice, due to a number of factors which include heuristic optimization in synthesis, quantization effects, poor estimation of wire capacitance, and inaccurate estimation of power/energy. In addition, composition of delay where block delays do not add linearly and when variables or sensitivity saturate, can also lead to inaccuracies. These limitations and issues are discussed later in this chapter and the next. A representation of this phenomenon was shown earlier in Figure 3.3. An actual example is shown in the difference between the ideal curve and the synthesized one in Figure 3.7. Some of these issues can be addressed, but not completely eliminated: we can use simulation to capture switching activity for a design; an iteration of place and route can be performed to obtain a more accurate estimate of wire capacitance. However, the quantization effect can only be reduced by adding more cells. In addition, the approximated curve is based on models of sensitivity and approximation of the contri-

bution of a block to the total energy and delay of the system; hence it may not match the synthesized curve.

3.3 Sensitivity Balancing Across Layers of Hierarchy

The ultimate goal is to minimize the difference between the synthesized energy-efficiency boundary and the ideal estimated energy-efficiency boundary for the system, leading to a more energy-efficient system. This is shown in the optimizations given in Equations (3.1) and (3.2).

$$\min \ \parallel S_X^C - OptAggS(S_X^A, S_X^B) \parallel \tag{3.1}$$

$$\min \ \parallel S_X^C - S_Y^C \parallel \tag{3.2}$$

The optimization above is an example for a system C (see Figure 3.3) which is comprised of two blocks A and B, and two tuning variables X and Y. The variable S_X^C refers to the actual sensitivity to tuning X in block C which is composed of block A and B. The constraints are minimum and maximum conditions on sensitivity, energy, and delay. The $OptAggS(\cdot)$ function refers to the calculated optimal aggregate sensitivity for the design C in terms of sensitivities of A and B to the respective tuning variables. The calculation is carried out based on composition rules. The optimal aggregate sensitivity can be either the one derived for a custom circuit implementation using the linear model for sensitivity to sizing for a custom-designed critical block such as the 64-bit adder mentioned earlier; or it can be the one derived from the synthesized version of the model.

Equation (3.1) minimizes the difference between the ideal energy-efficiency boundary and the current design point. Equation (3.2) minimizes the difference between sensitivity to different design tuning variables at a particular level of design hierarchy. When the difference between the previous and current iteration of the optimization is within a given threshold or when a fixed point is attained, the optimization is complete.

This is an elegant way to balance sensitivities across layers of hierarchy as the optimal aggregate sensitivity, which is our target, automatically provides us with a point on the ideal energy-efficiency boundary for the entire system. In addition, since the optimal aggregate

sensitivity is computed in terms of sensitivities to tuning variables of lower level blocks, we automatically assess energy-delay tradeoffs at higher levels of abstraction in terms of lower level energy-efficiency constraints.

3.4 Steps to Automation

The previous sections described a general method for propagating sensitivity across different layers of hierarchy while constructing the energy-efficiency boundary for a an entire system. This section explains how this methodology may be automated within a standard cell (ASIC) design flow. The presentation is only of a possible framework and is not rigorous. The development and implementation of an eventual algorithm is left as a topic for another dissertation. The problem is not easy to solve as the optimization spans a large design space and multiple layers of hierarchy.

The first important step to automating the above design methodology is to choose an appropriate data structures that can store a netlist for the system and can easily accommodate constraints such as interconnect, input and load capacitances, sensitivity, maximum delay, and maximum energy. The data structure must also be able to accommodate hierarchy and store architecture tradeoff information in the form of sensitivity. The next subsection describes a data structure that is well-suited to the sensitivity-based hierarchical optimization problem tackled in this dissertation.

3.4.1 Data Structures

The communication constraint graph structure described in [42] is modified slightly and used to represent the system hierarchically. The constraint graph consists of computational modules (of any size) communicating through point-to-point unidirectional channels that are connected to modules via means of input/output ports [42]. Each node in the constraint graph represents a port of a computational module, and each directed arc represents a point-to-point connection between two modules. Representation using these graphs allows for delineation between different architectures for the same function. An example is given

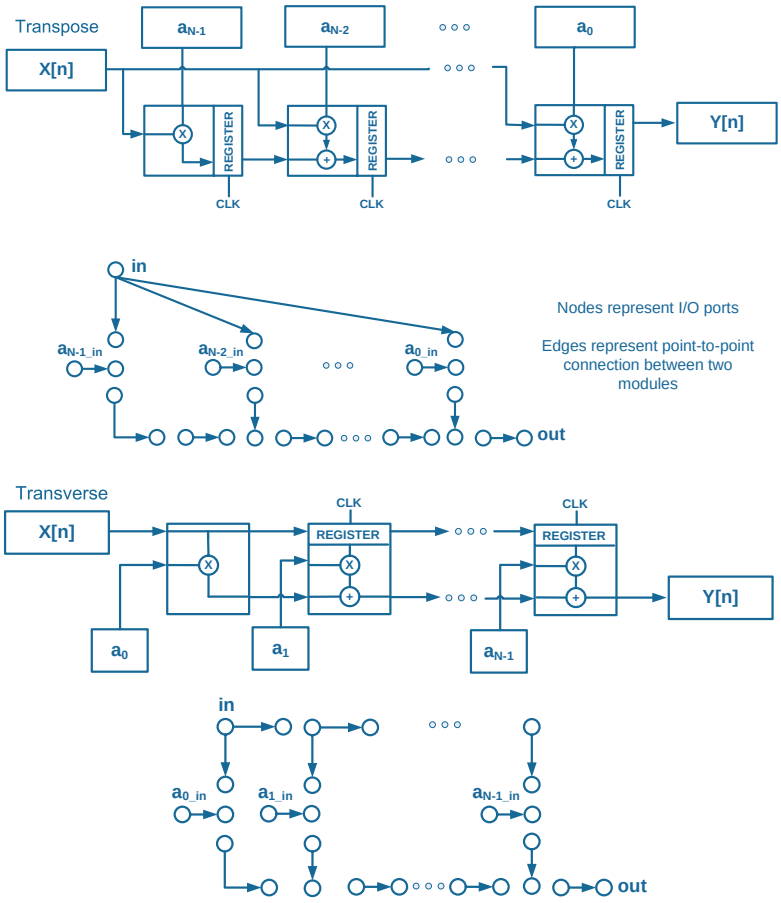


Figure 3.8. Interface constraint graphs for transpose and transverse filters

in Figure 3.8 for two different filter architectures (transpose and transverse) which perform the same function but have very different architectures and delay profiles.

The interface constraint graph contains the netlist if the nodes are grouped as shown in Figure 3.9. The red nodes represent add blocks, the blue nodes represent multiply blocks, and the green nodes represent registers in the constraint graph. Constraints are assigned to each node in the form of sensitivity, input capacitance, maximum gate capacitance, maximum delay and maximum energy. The sensitivity model generation is described in Chapter 5. Each edge of the graph is assigned a maximum wire capacitance constraint to prevent long wires. The cost of the netlist graph is in terms of delay, energy, and area. The cost of the constraint graph is in terms of sensitivity, input capacitance, gate capacitance,

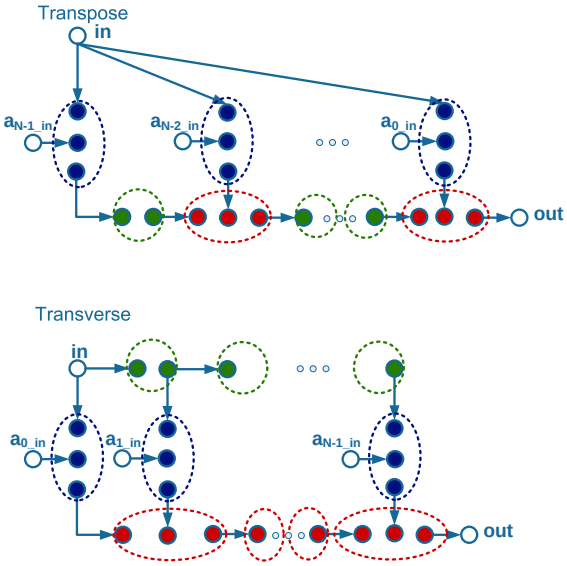


Figure 3.9. Examples of embedded netlists in filter constraint graphs

load capacitance, and wire capacitance. Constraints at the input and output nodes depend on the constraints of the constituent blocks and connectivity. Constraints can be modified by insertion of buffers, memory, repeaters, mux and demux circuits. Topologies can then be automatically evaluated using the mechanism presented in [42]. These graphs are used to store architecture tradeoff information for different system implementations.

Hierarchy can be incorporated into the constraint graph as modules represented by the nodes can be of any size. The modules themselves can contain additional constraint graphs representing the implementation of the module in terms of its building blocks. A simple example is shown in Figure 3.10. In the example, the top-level system D consists of three building blocks A, B, and C. Each of these blocks is implemented using NAND, NOR, and NOT gates. An interface constraint graph is constructed for each of A, B, and C. Then a separate interface graph is constructed for system D. This type of hierarchical representation allows constraints to be generated and propagated from lower levels of hierarchy to higher levels using sensitivity information as explained in the next section.

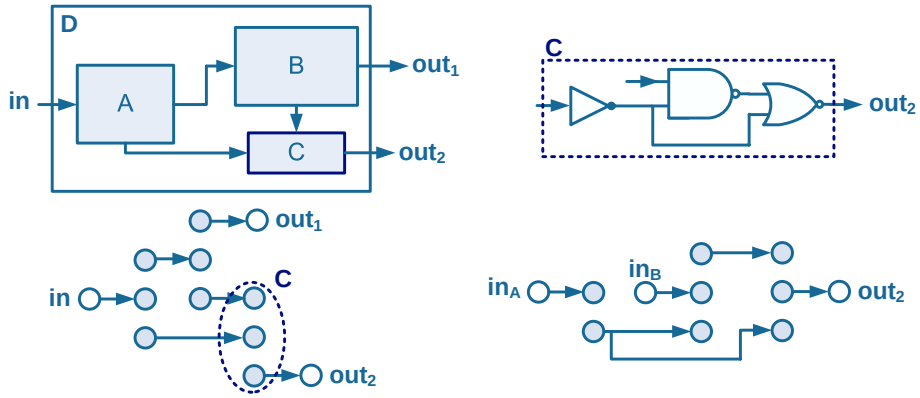


Figure 3.10. Example of a hierarchical constraint graph

3.4.2 Constraint Generation and Propagation

Constraint information is propagated from the top level block down to lower level blocks based on system cycle time and energy constraints. The lower level implementation propagates its delay and energy constraints upwards along with sensitivity information. The nodes store models for sensitivity in addition to maximum delay and energy constraints. The arcs store information about maximum wire capacitance constraints. Figure 3.11 shows a simple constraint graph for two blocks, A and B, which are connected in series to form a third block, C. The optimal aggregate sensitivity is given by the equation shown in the figure. The models for sensitivity in terms of gate capacitance and wire capacitance are stored at the nodes for any number of different input capacitances. The optimality constraint gives a specific gate capacitance to wire capacitance ratio for blocks A and B for a chosen sensitivity target. This in turn provides the optimality point for block C. Figure 3.11 shows that the optimal point for the targeted sensitivity exceeds the maximum delay target for block C. Either the maximum delay constraint must be adjusted or an alternate architecture must be chosen for block C. The mapping of the sensitivity model to the energy-delay tradeoff space is shown in the left side of Figure 3.11.

The example in Figure 3.11 shows that the optimal aggregate sensitivity constraint propagates sensitivity information to higher levels of abstraction providing the optimal point on the system energy-efficiency boundary. The system level maximum energy and

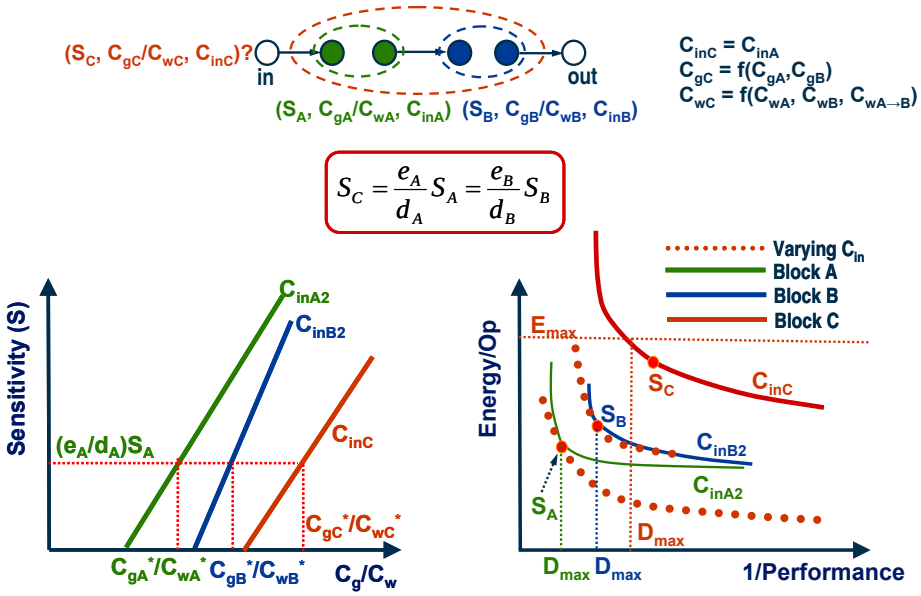


Figure 3.11. Example of constraint propagation

delay constraints are propagated down to lower level blocks via constraints on their size and estimated contribution to overall delay and energy. If the estimates are incorrect, they are propagated back through the node constraints and the system level constraints are updated accordingly. This is done through modeling delay and energy at the system level in terms of delay and energy equations which are functions of energy and delay for smaller blocks (or sub-constraint graphs).

The architecture in Figure 3.11 can be adjusted to meet the maximum delay constraint by inserting a register between block A and B. The mechanism for modifications to architecture by insertion of buffers or registers can be automated based on the sensitivity and constraint information using modified algorithms described in [42]. The constraint graph for the example in Figure 3.11 would be adjusted as shown in Figure 3.12.

3.4.3 Sensitivity Balancing Formulations

Two possible formulations have been explored so far that relate to balancing sensitivities across all blocks of a system. The first is a simple, iterative exhaustive search using branch

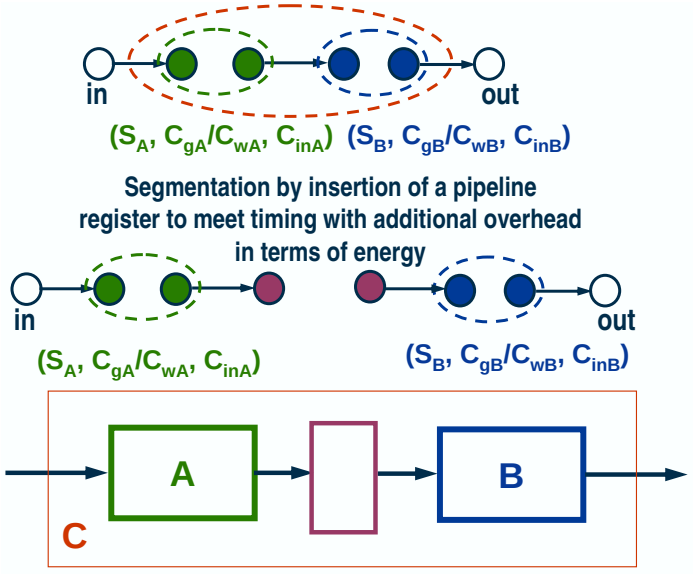


Figure 3.12. Example of register insertion to meet delay target

and bound. The second is a convex formulation using an approximation algorithm developed for finding *economic equilibrium*. Both branch and bound algorithms and approximation algorithms were introduced earlier in Chapter 2.

Exhaustive Search Using Branch and Bound

The first possible implementation of balancing sensitivity across a system is to employ a branch and bound strategy that exhaustively searches the design space. Unfortunately, the running time can be exponential in the worst case (i.e. where every single possible design point is explored). The likelihood of encountering the worst case scenario is small since the tuning variable with sensitivity that has the largest imbalance will be chosen to tune the design at each iteration. The design points in the branch and bound graph represent a particular system with a sensitivity that is estimated using a sensitivity model. The sensitivity of each tuning variable is compared with others to see if they are balanced within a given threshold. If they are not, then at each branch in the tree, the appropriate tuning variable is adjusted and then compared with the optimal aggregate sensitivity. Branching in this manner continues until a fixed point is reached or when the sensitivities are balanced

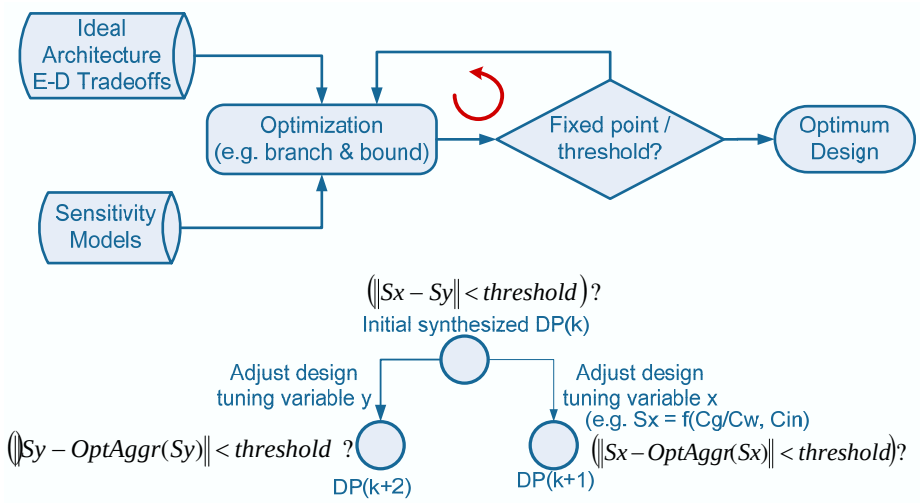


Figure 3.13. Conceptual representation of branch and bound

within the given thresholds. Figure 3.13 shows a conceptual representation of this method. Solutions where the difference between sensitivities exceeds the smallest difference found so far are pruned. The circuit and architecture tuning variable adjustments are stored in the respective interface constraint graphs which are associated with each design point.

Sensitivity information is used for both branching and cost of the branch and bound graph. The cost measures the difference between sensitivities (i.e. how close is the algorithm to ideally balancing sensitivities). Branching is based on the resulting design point that comes closest to having balanced sensitivities. The lowest cost found is used as a bound for the next iteration. A fixed point can be reached if no improvement is made on lowering the difference between sensitivities; or, when sensitivity to tuning variables is within a given threshold *and* estimated sensitivity is within a threshold of optimal aggregate sensitivity.

Convex Formulation Using Approximation Algorithm

The convex formulation for balancing sensitivity uses an approximation algorithm that was originally used to find economic equilibrium [30]. In the economic equilibrium problem, there are m traders, T_i, $i = 1, 2, \ldots, m$, and n commodities, C_j, $j = 1, 2, \ldots, n$. A utility function measures the *Constant Elasticity of Substitution* which represents the trader's

utility for a bundle of commodities. The utility function has the form shown in Equation 3.3 where α_j are constant parameters and $\rho \in (-\infty, 1] \setminus \{0\}$:

$$u(x) = \left(\sum_{j=1}^{n} \alpha_j \cdot x_j^{\rho} \right)^{\frac{1}{\rho}} \tag{3.3}$$

Every trader T_i has an initial set of commodities, $C_i = \{C_{i,j},\ j = 1, 2, \ldots, n\}$ that she wants to trade, and a utility function $u(x)$. $C_{i,j}$ is the amount of commodity C_j held by trader T_i. The consumption set X_i is the set of possible commodity bundles currently held by trader T_i. Each trader wants to maximize the utility at a certain price p:

$$\max\ u_i(x_i) \text{ subject to } x_i \in X_i \text{ and } (p, x_i) \leq (p, C_i) \tag{3.4}$$

In an economy where is the commodity is a price and each trader has a budget $C_i = e_i C$, where C is fixed, each trader is solving the following optimization problem:

$$\max\ u_i(x_i) \text{ subject to } x_i \in X_i \text{ and } (p, x_i) \leq e_i \tag{3.5}$$

The equilibrium price is one where all the traders spend their money and all goods are sold leading to the following Economic Equilibrium Problem, where over-lined variables represent vectors:

$$\begin{aligned} \overline{x_i} = \operatorname{argmax} \left\{ u_i\left(x_i\right) \mid x_i \in X_i; (x_i, \overline{p}) \leq e_i \right\}, \quad \forall \leq i \leq m \\ \textstyle\sum_{i=1}^{m} \overline{x_{ij}} \leq \sum_{i=1}^{m} e_i C, \quad \forall 1 \leq j \leq n \\ (\overline{p}, \textstyle\sum_{i=1}^{m} \overline{x_i}) = (\overline{p}, \sum_{i=1}^{m} e_i C) \end{aligned} \tag{3.6}$$

The classic general approach to solving the optimization in Equation 3.6 is exponential in time [30]. Thus, approximate equilibrium and ϵ-equilibrium were introduced [30].

In [30], it is shown that the economic equilibrium problem can be solved using a convex formulation that exploits the Karsh-Kuhn-Tucker (KKT) conditions of the utility maximizing problem, and when the economic equilibrium conditions can be formulated as a convex feasibility problem. This is the case for linear utility functions. Lagrangian theory is used in [30] to massage the optimization into a convex feasibility problem. The proof is not repeated here but the reader is referred to the cited text. The running time is polynomial if the utility function is linear [30].

The sensitivity balancing or sensitivity equilibrium problem can be similarly formulated. Instead of traders, there are blocks, B_i, and the commodities are energy and delay functions with respect to gate size represented by the vector w_i: $E_i(w_i)$ and $D_i(w_i)$. In order to massage the problem into a convex optimization, a linear utility function is required in terms of w_i. Total energy or total area provide a linear function. The equilibrium "price" is the equilibrium sensitivity s. If the utility function is total energy and the total delay is used as a budget for all blocks then the sensitivity equilibrium problem is formulated as:

$$\begin{aligned} \overline{w_i} = \operatorname{argmax} \left\{ -E_i\left(w_i\right) \mid w_i \in W_i; \left(D_i, \overline{s}\right) \leq d_i(w_i) \right\}, \quad \forall \leq i \leq m \\ \textstyle\sum_{i=1}^{m} \overline{D_i(w_i)} \leq \sum_{i=1}^{m} d_i D, \quad \forall 1 \leq j \leq n \\ \left(\overline{s}, \textstyle\sum_{i=1}^{m} \overline{D_i(w_i)}\right) = \left(\overline{s}, \textstyle\sum_{i=1}^{m} d_i D\right) \end{aligned} \tag{3.7}$$

In Equation 3.7, convexity holds because energy and delay can be modeled as convex functions (see Chapter 4). One of the limitations of the above formulation is that the blocks do not operate independently as they are connected. Each block affects neighboring blocks in the form of input capacitance and load capacitance. The effect of neighboring blocks can be modeled by additional constraints.

3.5 Sensitivity Approximation Using Interior Point Algorithms

In Chapter 5, a method of estimating sensitivity to sizing is outlined using simple properties of a circuit such as total gate capacitance, input capacitance, and total wire capacitance. A linear model is developed and shown to reasonably estimate sensitivity of any type of circuit block. In this section, it is shown theoretically that sensitivity to sizing can be automatically obtained if interior point algorithms are used to solve a convex optimization problem. Zlatanovici shows in his thesis [17] that the continuous gate sizing problem can be formulated as a convex optimization of the form min $f_0(x)$ subject to $f_i(x) \leq 0, i = 1, \ldots, m$ and $Ax = b$, where $f_0, \ldots, f_m$ are convex and twice differentiable. If it is assumed that the problem is solvable and strictly feasible then Newton's method can be applied to a sequence of equality constrained problems or to a sequence of modified KKT conditions.

The barrier method described in Chapter 2 is an interior point algorithm which reduces the inequality constrained problem to a sequence of linear equality constrained problems. Its only requirement is that functions be in convex form (i.e. linear programs, quadratic programs, quasi-convex quadratic programs, or geometric programs). This is the case with the gate sizing problem formulated and implemented in Zlatanovici's convex optimizer [17]. The optimizer is used in this thesis to develop models for sensitivity.

The goal is to approximately formulate the inequality constraint problem as an equality constrained problem so that Newton's method can be be applied. The inequality constraints are made implicit in the objective function. For example, consider the general energy-delay optimization problem: min $d(x)$ subject to $e'(x) \leq 0$, where $d(x)$ represents delay and $e'(x) = e(x) - e_{max}$ represents energy. For simplicity assume that $x \in \Re_+^n$. Both energy and delay are functions of a set of optimization variables x which represent tuning knobs such as gate sizes. The functions $d(x)$ and $e(x)$ can represent delay and energy, respectively, at any level of design abstraction: circuit level, micro-architecture level, or architecture level. From Chapter 2, the sensitivity to optimum value x^* is given by:

$$S\left(x^*\right) = -\frac{\nabla e'(x^*)}{\nabla d(x^*)} \cdot \frac{d(x^*)}{e'(x^*)} \tag{3.8}$$

Now assume that $d(x)$ and $e(x)$ are both convex functions that are twice differentiable and an optimal solution x^* for the optimization problem exists and is strictly feasible: $e'(x) < 0$. Then the optimization can be rewritten to make the inequality constraints implicit in the objective function as shown in the following:

$$\min\ d(x) + I\left(e'(x)\right) \text{ where } I : \Re \rightarrow \Re \tag{3.9}$$

The indicator function was previously defined in Chapter 2 and repeated here for convenience:

$$I(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0 \end{cases} \tag{3.10}$$

Since the objective function in Equation (3.9) is no longer differentiable in general, it can be approximated by a log function as explained in Chapter 2.

Using the theory outlined in Chapter 2, the constrained delay minimization problem becomes:

$$\min \ -t \cdot d(x) - log(-e'(x)) \tag{3.11}$$

For $t > 0$, $x^*(t)$ is defined as the solution to Equation (3.11) at each t. From Boyd and Vanderberghe (Chapter 11) [33], the sets of points $\{x^*(t), t > 0\}$ represents the central path associated with the original optimization problem.

The central path has a simple mechanics interpretation in terms of potential forces acting on a particle in the strictly feasible set. Each constraint is associated with a force acting on a particle when it is at the position x:

$$E(x) = -\nabla \left(-log(-e'(x))\right) = \frac{1}{e'(x)} \cdot \nabla e'(x) \tag{3.12}$$

The potential associated with the total force field generated by the constraints is the logarithmic barrier $\phi(x) = -\left(\frac{1}{t}\right) log(-e'(x))$. There is another force $D(x)$ associated with the particle at position x:

$$D(x) = -t \cdot \nabla d(x) \tag{3.13}$$

Equation (3.13) represents the objective force field acting on the particle to pull it in the negative gradient direction, i.e. toward a smaller $d(x)$. The parameter t scales the objective force, relative to the constraint forces. The central point $x^*(t)$ is the point where the constraint forces exactly balance the objective forces felt by the particle: $E(x^*(t)) = D(x^*(t))$. Then this implies the following:

$$\begin{aligned} -t \cdot \nabla d(x^*(t)) &= \tfrac{1}{e'(x^*(t))} \cdot \nabla e'(x^*(t)) \\ -t &= \tfrac{\nabla e'(x^*(t))}{\nabla d(x^*(t))} \cdot e'(x^*(t)) \end{aligned} \tag{3.14}$$

Equation (3.14) is very similar in form to Equation (3.8). If Equation (3.8) is divided by $d(x^*(t))$ (note $d(x^*(t)) > 0$) then:

$$\begin{aligned} t &= \tfrac{S(x^*(t))}{d(x^*(t))} \\ S(x^*(t)) &= t \cdot d(x^*(t)) \end{aligned} \tag{3.15}$$

Hence, sensitivity can be directly derived from the parameter t that results from the log barrier interior point method.

According to [33], the point $x^*(t)$ is $\frac{m}{t}$ sub-optimal. The value $\frac{m}{t^{(0)}}$ can be chosen to be approximately the same order as $f_0(x^0) - p^*$, where p^* is the optimum dual feasible point for the Lagrangian. Then $x^*(t)$ minimizes the Lagrangian: $L(x, \lambda) = d(x) + \lambda \cdot e'(x)$ for $\lambda = lambda^*(t)$. Every central point yields a dual feasible point and hence a lower bound on the optimum value, $d(x^*)$. Then:

$$\lambda^*(t) = -\frac{1}{t \cdot e'(x^*(t))} = -\frac{d(x^*(t))}{S(x^*(t)) \cdot e'(x^*(t))} \tag{3.16}$$

Equation (3.16) relates sensitivity to the Lagrange multiplier in a constrained delay minimization; it also connects t to the Lagrange multiplier. In the case where the barrier method is used, sensitivity provides a lower bound on the optimum delay at any point during the optimization.

In the above, if a convex formulation for a continuous optimization of a constrained delay minimization is possible, then employing an interior point method of solution (e.g. log barrier method) gives an easy mechanism for estimating the sensitivity and hence the lower bound on delay at any iteration of the Newton step. The method converges quickly according to [33], so this is a feasible method of carrying out leaf cell sizing optimization and obtaining sensitivity information as a by-product.

3.5.1 Saturation of Sensitivity and Variable Bounds

In some cases, sensitivity may saturate to very high values (i.e. along the steepest part of the energy-delay curve at the minimum delay point) or to zero (i.e. minimum energy point). Also, each design variable has certain bounds or limitations on their effectiveness. For example, if gate sizes are reduced to very low values for a memory intensive design such as the distributed arithmetic programmable filter designed in Chapter 6, the increased delay can lead to an increase in (leakage) energy per cycle due to longer cycle time.

Constraints on sensitivity can be added to avoid saturation of sensitivity to either very high values or values close to zero. Target minimum and maximum values of sensitivity can be added to the constraint functions in the sensitivity balancing problem formulation. The

constraints could limit the exploration to remain at the knee of the energy-delay tradeoff curve for all blocks.

If a desired aggregate sensitivity is not input to the algorithm or if constraints on sensitivity are not set, then the algorithm can get stuck at either extreme: zero sensitivity or infinite sensitivity. This can occur when sensitivities to all tuning variables for all blocks are zero or infinity. The algorithm will stop when this is the case because the algorithm sees a balance at these extreme points.

3.6 Limitations

There are certain limitations to using the hierarchical approach described in this chapter to find optimal architectures for systems implementing a specific digital function. The methodology is well-suited to early stages in design where the architecture has not been finalized and a few candidates exist as possible choices. Energy and delay models must be developed for each architecture in terms of lower level building blocks. Models for sensitivity must be generated for the simplest building blocks and optimal aggregate sensitivity must be calculated for each architecture. These tasks do not need to be repeated. They need to be performed only once at the beginning of the optimization and then iterative optimization can be carried out using the models to optimize a system for the given energy and delay constraints.

Five of the most important issues that must be addressed by the models is how to compose designs such that interconnect, leakage energy, activity factors for switching energy, non-linear addition of delay, and variability are all accounted for accurately. Variability is not addressed in this dissertation; it is left as a topic for future research. Composition that takes into leakage energy, activity factors, and non-linear addition of delay in a manner that is reasonable for architecture analysis and optimization is discussed in Chapter 4. Interconnect and its impact on architecture selection is also discussed in the same chapter.

3.7 Summary

This chapter described a hierarchical methodology that spans technology, circuit, micro-architecture and architecture levels. Sensitivity information is propagated up through the design hierarchy using composition and optimal aggregate sensitivity rules. The method allows a designer to quickly and systematically traverse the large energy-delay tradeoff space without the need for long simulation or compute-intensive derivative calculation. The methodology scales as the optimization is hierarchical. An example of generating energy-efficiency boundaries for system level designs was given in the form of a digital filter.

A framework for automating the design methodology was provided. The best data structure that captures hierarchy, constraints, and delineates between different architectures for the same function is an interface constraint graph. Energy-delay tradeoff information can be stored for different architectures using this hierarchical data structure.

Two possible implementations of the sensitivity balancing problem were provided in the form of either an exhaustive branch and bound search or using an an approximation algorithm based on economic equilibrium. The algorithm was modified to fit the sensitivity balancing problem. The formulation is convex and running time is polynomial.

This chapter also outlined a method of estimating sensitivity to sizing automatically via the use of the log barrier variant of the interior point method for solving a continuous convex optimization. This is useful at the leaf cell level. Chapter 5 gives an alternate method of estimating sensitivity if continuous convex optimization is not possible. The next chapter reviews and summarizes models for energy, delay, and interconnect required for this hierarchical power-performance optimization methodology.

Chapter 4

Models and Constraints

> *Taking all physical aspects of each component into account when designing complex digital circuits leads to unnecessary complexity that quickly becomes intractable.*
> – R. Rabaey et. al., *Digital Integrated Circuits: A Design Perspective*, 2003

Modeling and simulation develop an understanding of interaction between different parts of a system and of the system as a whole. A model is a simplified representation of a system at some particular point in time or space. Models are created with the intention to promote understanding of a system or one of its components. If a model is too simple, it runs the risk of neglecting relevant interactions; if it is too detailed, then the model becomes so complex that it detracts from understanding the system or component behavior. The accuracy of a model dictates the accuracy of the analysis and/or optimization. Good models benefit designers by helping them understand dynamic complexity within a short period of time.

In circuit design, accurate transistor models are necessary in understanding, analyzing and designing any circuit system. Without models that accurately reflect transistor behavior, circuit simulation and optimization would not be possible. Complex relationships between physical properties of a circuit are abstracted using models, thereby easing the analysis and design of circuits. Models also provide a means for shortening design times thereby reducing time to market.

Models for energy, delay, area and interconnect are the basic foundations for circuit

analysis and optimization. There are many different ways of modeling energy, delay, area, and interconnect. The first part of this chapter summarizes well-known models that abstract delay, energy, and area in terms of circuit parameters such as gate capacitance, input capacitance, and wire capacitance and resistance. These models provide the foundation for the work in this dissertation.

The second part of this chapter introduces models for sensitivity. The work in this dissertation requires sensitivity models to construct power-performance optimal systems. Analytical models have been used in the past in other sensitivity-based optimization techniques [11, 12, 14, 29] and are briefly summarized in Section 4.6. Chapter 5 describes an alternate and simpler method for constructing a model for sensitivity to sizing. Once a model for sensitivity is available, composition rules can be used to propagate sensitivity to higher levels of design abstraction as described in Chapter 2. Section 4.7 discusses the subtleties of determining aggregate sensitivity based on the delay, energy, area, and interconnect models described in this chapter.

4.1 Analytical Delay, Energy, and Area Models

In this section, analytical models for delay, energy, and area are summarized. A subset of these are used in the implementation of the custom circuit optimizer [17] which is briefly described in Section 4.2. The optimizer is used to optimize small circuits such as adders and inverter chains. The results of the optimization are used to develop models for sensitivity to sizing. In the next subsection models for delay of varying complexity and accuracy are summarized.

4.1.1 Delay

The foundation of the delay model is a linear function of load capacitance as outlined in the method of logical effort by Sutherland et. al. in [43]. The total delay for a given gate is the sum of delay caused by capacitive load driven by the gate and its topology, and a fixed parasitic delay that is also dependent on the topology of the gate. The basic delay

model can be enhanced to account for slopes, multiple paths from inputs to outputs, and interconnect.

The logical effort delay model that accounts for multiple paths is given by:

$$t_D = \sum_{i=1}^{N} p_i + b_i \cdot g_i \cdot h_i \tag{4.1}$$

The variable p_i represents the intrinsic parasitic delay of gate i, b_i represents the branching effort at input of gate i, g_i represents the logical effort of gate i, and h_i represents electrical effort of gate i. The electrical effort is given by: $h_i = \frac{C_{g_{i+1}}}{C_{g_i}}$, where C_{g_i} is the relative size of gate i. Equation (4.1) can be minimized with respect to relative sizes of gates, C_{g_i}, such that the input capacitance is less than a given maximum value and the relative size of any gate is at least 1. The optimal design in terms of minimum delay is one where stage efforts are equal: $g_i \cdot b_i \cdot h_i = g_j \cdot b_j \cdot h_j$ [43].

Equation (4.1) is posynomial in nature and can be used to model delay in a convex optimization program. In [17], Equation (4.1) is extended to account for dynamic gates such that the delay expression remains posynomial as shown in below, where y_i contains the keeper sizes.

$$t_D = \sum_{i=1}^{N} p_i + (1 + \frac{y_i}{2C_{g_i}}) \cdot b_i \cdot g_i \cdot h_i \tag{4.2}$$

The additional load due to the inverter used by the keeper is accounted for by lumping it into the wire capacitance at that node [17]. Signal slopes are accounted for by adding an additional term $\eta \cdot t_{slope,in}$ to Equation 4.2. These are propagated through the path by an additional equation:

$$t_{slope,out} = \lambda + \mu \cdot \frac{C_L}{C_{in}} + \nu \cdot t_{slope,in} \tag{4.3}$$

These models are similar to level-1 models used by commercial synthesis tools. In [17], Equations (4.2) and (4.3) are massaged such that they remain posynomial and can be used in the implementation of a convex circuit optimizer.

4.1.2 Energy

Switching or dynamic energy and leakage energy are the two dominant components of total energy. Switching energy is proportional to switched capacitance and modulated

by activity factors and supply voltage. Equation (4.4) below gives the total switching energy which is a function of α_i, the switching activity at node i; C_i which is the switched capacitance at node i; and V_{DD}, the supply voltage:

$$E_{dynamic} = \sum_{nodes} \alpha_i \cdot C_i \cdot V_{DD}^2 \tag{4.4}$$

Dynamic energy is a strong function of supply voltage due to the V_{DD}^2 term in Equation (4.4).

In [17], leakage energy is modeled simply by computing the average leakage power of a gate via simulation and multiplying it by the size of the gate, C_{g_j} and the cycle time, T_{cycle}:

$$E_{leakage} = T_{cycle} \cdot \sum_{gates} C_{g_j} \cdot P_{leak,j} \tag{4.5}$$

If T_{cycle} is computed as the maximum of all path delays, then Equations (4.5) and (4.4) are generalized posynomials.

Leakage energy depends on sub-threshold current models. Analytically, the sub-threshold current is defined as [44]:

$$I_{ds} = \mu \cdot \frac{W}{L} \left(\frac{kT}{q}\right)^2 e^{\frac{V_g - V_{TH}}{\eta kT/q}} \left(1 - e^{-\frac{V_{ds}}{kT/q}}\right) \tag{4.6}$$

In Equation (4.6), q is the electronic charge, T is the temperature, V_g is the gate voltage, V_{ds} is the source-to-drain voltage, k is Boltzmann's constant, η is the MOSFET body-effect coefficient, L is MOSFET channel length, W is the MOSFET width, and μ is the carrier mobility. The first part of Equation (4.6) accounts for sub-threshold diffusion current and the second exponential term accounts for drain induced barrier lowering (DIBL) effect [45]. Equation (4.6) can be rewritten as:

$$I_{ds} = I_0 \cdot W \cdot 10^{\frac{(V_{gs} - V_{TH}) + \gamma V_{ds}}{S}} \tag{4.7}$$

where $S = 2.3 \cdot \frac{\eta kT}{q}$ is the sub-threshold slope; I_0 is the MOSFET current per unit width at the reference threshold for the given technology, and W is the relative width of the transistor. The static leakage current for a gate with inputs in state S_{in} and where $V_{gs} = 0$, $V_{ds} = V_{DD}$ is given by [44, 46, 47]:

$$I_{leakage} = W \cdot I_g(S_{in}) \cdot e^{\frac{V_{TH} - \gamma V_{DD}}{V_0}} \tag{4.8}$$

where $V_0 = \frac{\eta kT}{q}$, W is the relative size of the gate, and I_g is the normalized leakage current of the gate. The total leakage energy for a circuit is given by:

$$E_{leakage} = T_{cycle} \cdot V_{DD} \cdot \sum_{gates} W_i \cdot I_{g_i}(S_{in_{g_i}}) \cdot e^{\frac{V_{TH} - \gamma V_{DD}}{V_0}} \tag{4.9}$$

Switching energy and leakage energy are modeled separately and accumulated to form the expression for total energy:

$$E_{total} = E_{dynamic} + E_{leakage} \tag{4.10}$$

The model for total energy does not account for crowbar current which is negligible relative to switching and leakage energy. When these models are transformed into posynomials and used in optimization, the formulation becomes a generalized geometric program which can be solved efficiently [33].

The dependence of energy on V_{DD} and V_{TH} is clear from Equations (4.9) and (4.4), and can be used to extend the optimizer in [17] to use supply voltage and threshold voltage as tuning variables. The delay model can be modified to include dependence on V_{DD} and V_{TH} as will be shown in Section 4.5.

4.1.3 Area

Area can be modeled as a linear combination of gate sizes; u is the area of the gate when C_{g_i} is 1:

$$A = \sum_{i=1}^{N} u_i \cdot C_{g_i} \tag{4.11}$$

The expression is posynomial in C_{g_i} and hence can be added to the optimization if desired. In this work, the focus is on constrained delay minimization subject to an energy constraint. The model for area is included here for completeness.

4.2 Constrained Optimization

The general constrained delay minimization problem cannot be solved analytically. A numerical optimizer is required for a complete solution [48]. This section describes the gate

sizing problem formulation in the form of a convex optimization problem that can be solved using a geometric program. It is same formulation that optimizer in [17] is based upon. The optimizer uses tabulated models for accuracy. A discussion on the use of more accurate tabulated models, their drawbacks, and the effect on optimality of the resulting solution is given in Section 4.3. The models and problem formulation are extended in Section 4.5 to account for supply voltage and threshold voltage adjustment.

The energy-delay optimization problem can be formulated as either an energy-constrained delay minimization problem or as a delay-constrained minimization problem. Using the models in Section 4.1, both formulations result in generalized geometric programs, as the objective and constraints are generalized posynomials. The formulations are given in Equations (4.12) and (4.13).

$$\text{minimize } t_D \text{ over } C_{g_i} \text{ subject to } \begin{cases} E \leq E_{max} & \\ C_{in} \leq C_{in,max} & \\ C_{g_i} \geq 1 & 1 \ldots N \\ t_{slope,j} \leq t_{slope,max} & j = 0 \ldots M \end{cases} \tag{4.12}$$

$$\text{minimize } E \text{ over } C_{g_i} \text{ subject to } \begin{cases} t_D \leq t_{D,max} & \\ C_{in} \leq C_{in,max} & \\ C_{g_i} \geq 1 & 1 \ldots N \\ t_{slope,j} \leq t_{slope,max} & j = 0 \ldots M \end{cases} \tag{4.13}$$

There are multiple paths in a circuit and multiple critical paths. The latest arriving signal at the output of a gate is used to determine the overall delay of the circuit. This is given by the recursive equation:

$$T_i = \text{maximize } (T_j + D_i) \text{ over } j \in FI(i) \tag{4.14}$$

$FI(i)$ is the fan in of gate i, T_i is the maximum delay of all paths starting from the primary inputs and ending at gate i. The overall delay of the circuit is given by the maximum of all the latest arriving signals at the output of all gates:

$$D = \text{maximize } T_i \text{ over } i = \text{maximize } \{T_i | \text{output gate}_i\} \text{ over } i \tag{4.15}$$

Equation (4.15) is a recursive expression. The resulting optimization that takes into account multiple paths is given by:

$$\text{minimize } t_D \text{ over } C_{g_i} \text{ subject to } \begin{cases} E \leq E_{max} & \\ C_{in} \leq C_{in,max} & \\ C_{g_i} \geq 1 & 1 \ldots N \\ t_{slope,j} \leq t_{slope,max} & j = 0 \ldots M \\ T_i = 0 & \text{for primary inputs} \\ T_j \leq D & \forall j \\ T_j + D_i \leq T_i & \text{for } j \in FI(i) \end{cases} \tag{4.16}$$

The optimization problem in Equation (4.16) can be extended to account for differences in high-to-low delays and low-to-high delays as described in [17].

A simple static timer is used in implementing Equation (4.14) [17]. It does not account for false paths, but estimates activity factors via logic simulation using a large number of random input vectors. The activity factors are important for computing energy. The static timer is implemented in C and a Matlab-based optimizer is used to solve the generalized geometric program. Both are integrated into a standalone optimizer. The static timer module can be replaced by a more sophisticated one as long as the interface is preserved. The optimizer framework is summarized and depicted in Figure 4.1. Inputs include the tabulated and analytical models for energy and delay, a circuit netlist, constraints and optimization objective, and tuning variables. The operation is iterative until a fixed point is reached.

4.3 Tabulated Models

When there is a need for more accuracy in delay estimation, analytical models such as the convex models given above may fall short of what is required. Tabulated models (non-convex) can replace analytical expressions. Required delay and slope values can be looked up in a table during the optimization. The use of tabulated models impact the optimality of the final result. In [17], the optimization using tabulated models is considered

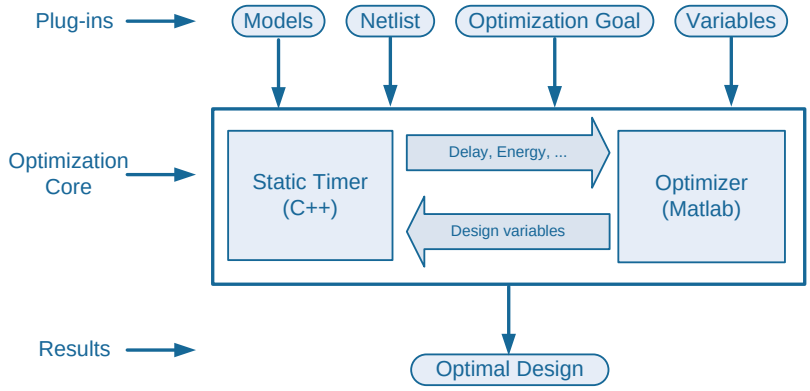


Figure 4.1. Matlab/C combinational circuit optimization framework for sizing, supply voltage, and threshold voltage optimization

"near-convex" and its optimality is verified against a "near-optimality" boundary which is obtained by first performing the optimization using analytical models and then measuring the delay and energy of the design obtained using accurate tabulated models. Effectively, the design obtained using analytical models is evaluated using tabulated models. In practice, the use of tabulated models results in gate sizes that give a near-optimal or optimal design. This has been shown in a practical design of a high-performance 64-bit adder [17, 27].

The explanation given in [17] for the near-optimal solution is somewhat unsatisfactory. A closer analysis using optimization theory presented in Chapter 2 shows that using tabulated models results in non-convex optimization. Solving the KKT conditions as described in Chapter 2 results in weak duality and a duality gap that is non-zero. By evaluating the analytical model solution using the tabulated models, Zlatanovici [17] is essentially providing a practical means of measuring the non-zero duality gap. The resulting optimal doublet (λ^*, ν^*) from the optimization only provides an accurate measure of variation around the optimal point found using analytical models. The doublet only *estimates* the gradient at the point obtained using tabulated models.

4.4 Wire Capacitance and Wire Resistance

Since interconnect is becoming an important issue for designs in new technologies, a summary of interconnect models and their impact on delay and energy is presented in this section. Additionally, effect of interconnect on sensitivity analysis and optimal architecture selection is also presented here.

4.4.1 Effect of Interconnect on Delay

Interconnect adds complication to the logical effort model for delay. Stage efforts are no longer equalized in the minimum delay solution: the effort of the gate driving the wire will be greater than the effort of the gate following the wire. The effect of interconnect can be modeled in a few different ways.

Short Wires

When wires are short, wire capacitance is treated as parasitic capacitance and lumped in with parasitic gate capacitance. If the average length of the wire is known and the average size of a gate is given, then the average ratio of parasitic diffusion capacitance to parasitic wire capacitance can be computed [43]. This ratio is then used to estimate the delay due to parasitic wire capacitance. The total delay due to parasitic capacitance is then given by sum of delay due to parasitic capacitance of logic gates and delay due to parasitic wire capacitance. The fixed delay attributed to the parasitic diffusion capacitance of a gate is given by p_i and the fixed delay attributed to parasitic wire capacitance at each node is given by p_{w_j}. The delay equation becomes:

$$t_D = \sum_{i=1}^{N} p_i + b_i \cdot g_i \cdot h_i + \sum_{j=1}^{total\_nodes} p_{w_j} \tag{4.17}$$

Medium and Long Wires

In the case of long or medium length wires, the most common model is to include the wire as an additional side load at the node where the wire appears. This model assumes

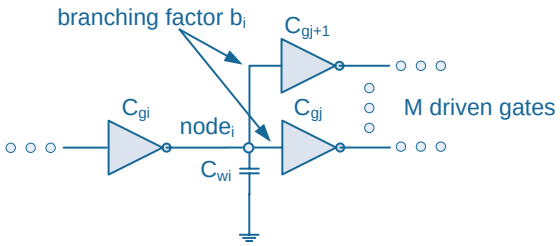


Figure 4.2. Wire capacitance as a side load at a circuit node

that the wire lengths are constant and independent of the sizes of neighboring gates. In [17], Equation (4.1) is extended by including interconnect as an additional load at a node. The resulting equation for the delay at a node i with interconnect capacitance C_{w_i} is given by:

$$\begin{aligned} t_{D_i} &= p_i + g_i \cdot \frac{C_{w_i} + \sum_{j=1}^{M} b_i C_{g_j}}{C_{g_i}} + \eta \cdot t_{slope,in} \\ t_{slope,out_i} &= \lambda + \mu \cdot \frac{C_{w_i} + \sum_{j=1}^{M} C_{g_j}}{C_{g_i}} + \nu \cdot t_{slope,in_i} \end{aligned} \tag{4.18}$$

The variable M represents the total number of gates driven by gate i. It should be noted that Zlatanovici has omitted the branching factor due to the wire in this node equation. Figure 4.2 shows the representation of the node given in Equation 4.18. The resistive effect of interconnect is accounted for by adding an additional term shown below, where α is a fitting parameter close to one[17]:

$$t_{D_i} = t_{D_i}(R = 0) + \frac{R_{w_i} \cdot (0.5 \cdot C_{w_i} + C_{after\_wire})}{\alpha} \tag{4.19}$$

Equation (4.19) is originally developed in [49]. The wire capacitance and resistance must be estimated prior to optimization. An early floor-plan of the design provides a good starting point for gauging wire capacitance and resistance. Equation (4.18) is used in the circuit optimizer developed in [17]. Unfortunately, the optimizer presented in [17] does not account for wire resistance.

An alternate and slightly more accurate model is given by Sutherland et. al. in [43]. They model the effect of interconnect by including a branching factor at the wire driving a gate; it is given as: $(C_{gate} + C_{wire})/C_{gate}$. The branching factor accounts for the current that is split between wire capacitance side load and the path following the interconnect.

The revised delay equation for multiple path delay excluding wire resistance becomes:

$$t_D = \sum_{i=1}^{N} p_i + g_i \cdot \left(b_i h_i + \frac{C_{w_i}}{C_{g_i}}\right) + g_{i+1} h_{i+1} b_{i+1} \left(\frac{C_{g_{i+1}} + C_{w_i}}{C_{g_{i+1}}}\right) + \eta \cdot t_{slope,in} \quad (4.20)$$

The Elmore model for wire segment delay can be used to include wire resistance [10]. For long and medium wires, wire resistance must be included as follows:

$$t_D = t_D(R = 0) + \sum_{i=1}^{N} \frac{R_{w_i} \cdot (0.5 \cdot C_{w_i} + C_{g_{i+1}})}{R_o C_o} \quad (4.21)$$

In Equation (4.21), R_{w_i} and C_{w_i} are the wire resistance and capacitance of wire segment i, and $R_o C_o$ is the delay of a minimum-sized inverter in the given technology. Equation (4.20) and Equation (4.21) are used in Chapter 5 to develop models for sensitivity to sizing.

4.4.2 Effect of Interconnect on Energy and Area

Interconnect increases total energy and total area of a system. The total switched capacitance of a wire, including a multiplicative factor for crowbar current is modeled in [6] as follows:

$$E_{wire} = c_w \cdot L \cdot V_{DD}^2 \left(1 + \frac{4.5}{3\sqrt{3}} \cdot c \cdot \frac{\hat{w}}{\hat{l}}\right) \quad (4.22)$$

In Equation (4.22), c_w is the total switched capacitance due to interconnect per unit length, L is the length of the wire, c represents the crowbar multiplier. The variables $\hat{l}$ and $\hat{w}$ represent the optimal wire segment length and repeater width of normalized to their delay-optimal values. The wire segment length between repeaters which have placed optimally with respect to delay is given by l_{opt}, and the optimal repeater width is given by w_{opt}. Since exact optimal device width is not usually used due to limited cell sizes in a standard cell library, the actual wire segment length and repeater width are given by: $l = l_{opt} \cdot \hat{l}$ and $w = w_{opt} \cdot \hat{w}$. As one can see from Equation (4.22), as total interconnect capacitance increases in a design, total energy also increases. Equation (4.22) can be added on to Equation (4.10) to account for additional energy due to interconnect.

Area also increases due to increase in total interconnect length. Wire length distributions can be estimated prior to the routing step. The basis for the calculation is Rent's

Rule [50, 51, 52] which was discovered in the 1960s by E. F. Rent at IBM. He found a trend between the number of pins (or terminals) at the boundary of an integrated circuit and the number of gates. On a log plot, the data points from various IBM designs formed a straight line, implying a power-law relationship: $T = k \cdot N^p$ [50]. The variable T is the total number of I/O pins, k is the average number of I/O's per gate, N is the total number of gates, and p is Rent's exponent which denotes the degree of wiring complexity with $p = 1$ being the most complex wiring network. It should be noted that Rent's Rule is an empirical result based on interconnect properties of existing designs. The constant p should be fitted to the types of architectures and constraints that are dominant in today's systems.

The wire length distribution in a system is determined through recursive application of Rent's Rule. The shortest wires are estimated by applying Rent's Rule at the logic level where the system is partitioned into logic gates. Rent's Rule is then applied to interconnects between closest neighboring gates to determine the number of interconnections between them. Longer wires are estimated by clustering gates into blocks recursively until the longest interconnections are found.

There are a number of different ways to apply Rent's Rule. The first was by Donath in a 1979 article [51] where he developed the wire length distribution function for placed and partitioned designs. It was later revised by Donath in 1981 [52] and is summarized below in Equation (4.23) where f_k is the fraction of wires with length k, g is a normalized constant, L is a constant related to the size of the array and adequacy of placement, and γ is a constant characteristic of the logic and related to Rent's exponent p by the equation $2 \cdot p + \gamma \approx 3$ [52].

$$\begin{aligned} f_k &= g/k^\gamma \quad \text{when} \quad 1 \leq k \leq L \\ f_k &\approx 0 \quad \text{when} \quad k > L \end{aligned} \tag{4.23}$$

Since Donath's 1981 publication, there have been many revisions and versions of the wire length distribution function. One of the most useful variations (for architecture analysis) is by Davis et. al. [53] that provides distributions for local, semi-global, and global wiring. In [53], the wire length distribution is described by an *Interconnect Density Function* (IDF), $I(l)$ and a *Cumulative Interconnect Distribution Function* (CIDF) which gives the total

number of interconnects that have length less than or equal to l (measured in gate pitches). It is defined below in Equation (4.24) where x is a variable of integration representing length:

$$I(l) = \int_1^l i(x)dx \tag{4.24}$$

In order to derive the wire length distribution of an integrated circuit, the circuit is divided into N logic gates, where N is related to the total number of transistors N_t by $N = N_t/\alpha$ where α is function of the average fan-in and the average fan-out in the system. The gate pitch is defined as the average distance between logic gates and is given by $\sqrt{A_c/N}$ where A_c is the total area of the chip. An example of how Equation (4.24) is applied to three blocks connected to one another is detailed in [53] and summarized in [54]. It will not be repeated here for sake of brevity, but the reader is encouraged to consult the cited texts. Once wire length distributions are derived for a system, they can be used to calculate total interconnect area based on wire widths for all layers of metal in a technology, and also used to estimate total interconnect capacitance for energy and delay tradeoff analysis.

4.4.3 Effect of Interconnect on Sensitivity and Architecture Selection

Interconnect scaling has an increasing effect on power and performance of digital circuits. Wire scaling studies, based on ITRS projections, show that digital designs become communication bound rather than capacity bound as global and intermediate wires become slower relative to logic [13, 55, 56, 6, 57]. As technologies scale to 65nm and beyond, interconnect delay becomes a significant portion of the total delay, even for gate-dominated paths [57]. In Figure 4.3, data provided by Texas Instruments shows that at 130nm, intra- and inter-cell communication delay is 36% of total circuit delay; in 65nm technology, it jumps to 54% [57].

The increase in interconnect delay is mainly due to an increase in wire resistance. As wire aspect ratios cap at 2.2, wire resistance grows quickly with scaling. Wire delay is equivalent to 2-3 FO4/mm^2 for a 65nm process [6]. In his PhD dissertation, R. Ho shows that even for an optimally repeated wire, the total delay along the wire is linear with total

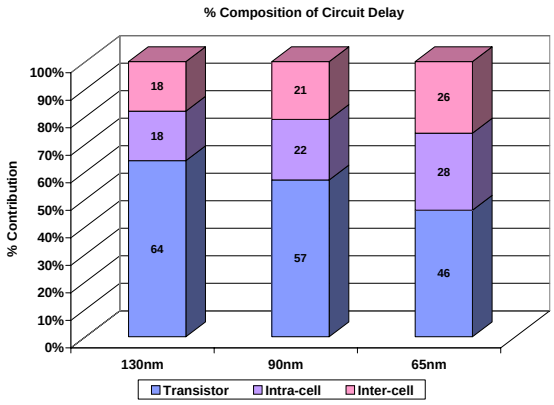


Figure 4.3. Interconnect delay contribution in a gate-dominated design

wire length, hence making extensive use of long or intermediate wires prohibitive for designs in 65nm or below technologies [6]. This limitation has implications on architecture design.

In the past wider issue machines were used to provide performance gains; but with scaling, wires in such architectures grow in length, requiring slower clocks or additional pipeline stages so that the amount of state that is reachable in a single clock cycle remains constant [2]. Hence, the performance gains from instruction-level parallelism are reduced as technologies scale to 65nm and beyond, limiting the scalability of conventional architectures. Increasing global wire delay and faster transistors also imply that centrally located large memory-oriented architectures do not scale with technology [55]: modular designs are more desirable. The increasing amount of interconnect delay in both wire- and gate-dominated designs implies tradeoffs between the size and partitioning of structures. The requirement for high performance also limits the size of pipeline stages and constrains placement. If a slower clock is required to meet interconnect delay constraints, then more parallel architectures are required to meet performance needs. At the same time, energy expended per cycle must meet power budgets. In light of these studies and tradeoffs, architecture optimization must consider the impact of interconnect on energy-efficiency at early stages in the design process. The designer must have a vehicle for understanding the tradeoffs between centralized monolithic architectures versus modular, distributed parallel architectures to arrive at an energy-efficient system that meets performance constraints.

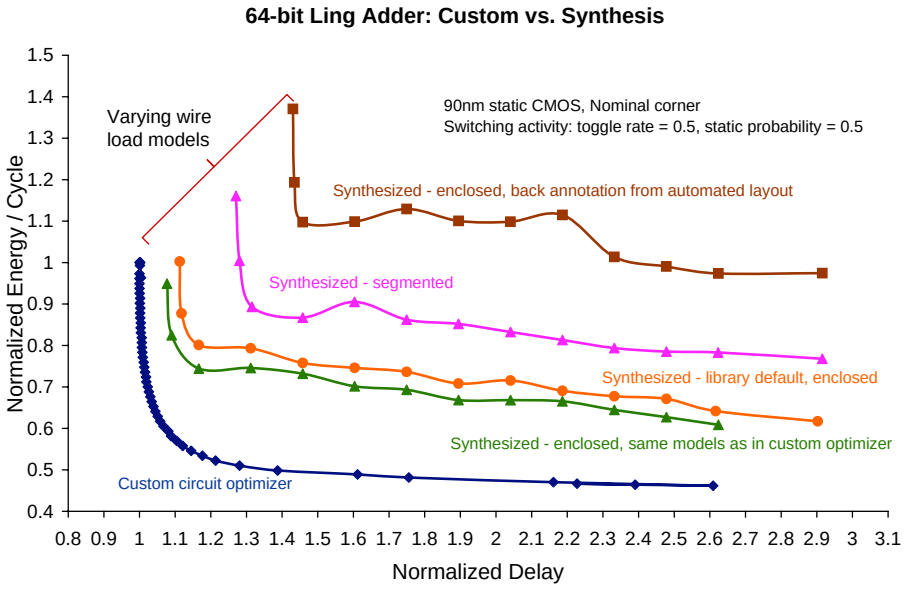


Figure 4.4. Effect of wire load models on energy-delay tradeoffs for a 64-bit Ling adder

The impact of interconnect on design can be included via models for sensitivity as is done in Chapter 5. However, the estimation of wire capacitance is dependent on wire load models used in synthesis-based design. Addressing the issue of interconnect on architecture selection is an entire thesis in itself so this section focuses on wire load models and their effects on energy-delay tradeoffs.

Wire load models affect the estimation of C_{wire} which is critical in estimating the sensitivity of a synthesized design as described in the next chapter. In Figure 4.4, a plot of the energy-delay tradeoffs of a static CMOS 64-bit Ling adder is shown. The synthesized version of the adder is exactly the same at the gate-level as the custom-sized version shown by the blue curve. The wire loads used for the custom version are extracted capacitances from a custom layout of the adder. The wire loads are varied for the synthesized versions. The library default uses an enclosed model which is shown by the orange curve. The model that uses an segmented wire load model is shown by the magenta curve. The brown and green curves use custom wire load models: the brown one uses back annotation of wire capacitances from automated layout of the adder design, and the green curve uses the same values as for the custom optimized design.

As one can see the energy-delay tradeoffs vary greatly depending on what sorts of wire load models are available. There is up to a 43% variance in the minimum delay when compared to the custom sized adder in the same technology. In terms of energy, there is up to a 60% variance from the custom optimized design. In Figure 4.4, each different wire load model yields a different sensitivity at any given energy-delay point. This can easily be seen by the position of the knee of the curve in the energy-delay tradeoff space.

If this adder was used as a building block in a larger design such as an integer execution unit, it would be difficult to decide whether this particular block should be included or an alternate be used if the wire load models are not accurate or have large variance. It would also be difficult to estimate sensitivity based on just gate capacitance and wire capacitance information under such large variation. Mitigation of these issues requires reasonable estimates of wiring capacitance from the technology and at least one pass of place and route. These numbers should be correlated to wire load models made available in the library. If there is a large variance between the default models that come with the library, then the custom generated wire load models based on accurate information from place and route should be used.

4.5 Extension to Supply Voltage and Threshold Voltage Optimization

The optimization framework at the circuit level can be extended to include tuning variables such as supply voltage and threshold voltage. This can be achieved by extending the models used for delay and energy to account for threshold and supply voltage. Energy is modeled as a function of supply voltage and threshold voltages in Section 4.1 earlier. In this section, delay is modified to include dependence on supply voltage and transistor threshold voltage.

In [58], compact models are used to extend the model for delay and energy to include dependence on supply voltage and threshold voltage. These models are transformed into posynomials using a change of variables and the optimization tool described in Section 4.2

is employed to explore optimum supply and threshold for different process technologies (180nm, 130nm, 90nm) in [58]. The model for delay uses the gate equivalent resistance [10] which is computed from analytical saturation current models [59, 58] used in BSIM version 3 [60]. The equivalent resistance, R_{EQ} is given as [17]:

$$R_{EQ} = \frac{3}{4} \cdot V_{DD} \cdot \frac{(\beta_1 \cdot V_{DD} + \beta_0 + V_{DD} - V_{TH})}{W \cdot K \cdot (V_{DD} - V_{TH})} \left(1 - \frac{7}{9} \cdot \frac{V_{DD}}{V_A}\right) \tag{4.25}$$

In Equation (4.25), V_{DD} is the supply voltage; V_{TH} is the threshold voltage; V_A is the Early voltage [44]; W is the size of the gate; K is the size of the gate with $W = 1$; β_0 and β_1 are technology parameters. The delay is then given by:

$$t_D = c_2 \cdot R_{EQ} + c_1 \cdot R_{EQ} \cdot \frac{C_L}{C_{in}} + (\eta_0 + \eta_1 \cdot V_{DD}) \cdot t_{slope,in} \tag{4.26}$$

This equation is transformed into a posynomial expression in [58] and used in the optimizer developed by Zlatanovici [17] to explore optimum supply and thresholds for different process technologies.

The Alpha-Power Law [40] model can also be used to show the dependence of delay on supply voltage, V_{DD} and transistor threshold voltage, V_{TH} [10, 47]:

$$t_D = \frac{C_L V_{DD}}{2} \cdot \left(\frac{K_N}{(V_{DD} - V_{TH_N})^{\alpha_N}} + \frac{K_P}{(V_{DD} - |V_{TH_P}|)^{\alpha_P}} \right) \tag{4.27}$$

Equation (4.27) is a curve fitted expression depending on the technology parameters K_N and K_P for N and P devices, respectively; and on α_N and α_P for N and P devices, respectively. The value C_L is the total load capacitance, and V_{TH_N} and V_{TH_P} are the threshold voltages for N and P transistors, respectively.

4.6 Analytical Models for Sensitivity

In the previous sections it was shown that energy and delay are functions of gate size, supply voltage, and transistor threshold voltage. These three circuit tuning variables impact energy and delay differently. For example, switching energy is a quadratic function of supply voltage whereas it is a linearly related to total switched capacitance. The effectiveness of each of these design tuning variables is captured by sensitivity as defined in Chapter 2.

Sensitivity can be analytically derived or it can be obtained via circuit simulation. In this dissertation, it is shown that sensitivity can also be modeled using circuit properties such gate capacitance and wire capacitance. Other authors have used alternate means of calculating sensitivity.

Zyuban and Strenski use simulation to determine normalized sensitivity to supply voltage and gate sizing [15]. Markovic in his PhD dissertation [12] provides analytical models for absolute sensitivity to sizing, sensitivity to supply voltage, and sensitivity to threshold voltage. The models he describes are based on the Alpha-Power Law [40] delay model by Sakurai and Newton. The analytical sensitivity models are summarized in the following subsections. The basic delay model for a gate which is used as a starting point is not too different from Equation (4.27):

$$t_{D_i} = \frac{K_D \cdot V_{DD}}{(V_{DD} - V_{ON} - \Delta V_{TH})^{\alpha_D}} \cdot \left(\frac{W_{out}}{W_{in}} + \frac{W_{par}}{W_{in}} \right) \tag{4.28}$$

Equation (4.28) is a curve fitted expression that depends on parameters V_{ON} and α_D which are intrinsically related [12, 40]; ΔV_{TH} is the change in threshold voltage from the standard value given by the technology; K_D is a fitting parameter; $\frac{W_{out}}{W_{in}}$ is the electrical fanout of a gate; and $\frac{W_{par}}{W_{in}}$ is the intrinsic delay of a gate.

The energy of a gate is given by independent expressions for switching and leakage energy as were summarized in Section 4.1. The optimization approach taken by Markovic uses the minimum delay point as a reference under the standard supply voltage, V_{DD}^{ref} and threshold voltage, V_{TH}^{ref}. From this reference point, Markovic adjusts gate size, supply voltage, and threshold voltage until the sensitivities to each tuning variable are balanced. The energy is minimized for each new delay target $D = D_{min}\,(1 + d_{inc}/100)$, where the delay is increased by the increment d_{inc} at each step using the design tuning variables.

4.6.1 Sensitivity to Sizing

The sensitivity to sizing is calculated using the analytical expressions given above for delay and energy. Markovic uses the absolute gradient to determine sensitivity [12]. The absolute gradient or absolute sensitivity is equivalent to using normalized sensitivity for

to compare effectiveness of design tuning variables for small blocks without hierarchy (see Chapter 2). Hence the analytical expressions can also be used to model sensitivity for small designs. However, this method is not scalable to large designs that include multiple levels of hierarchy and thousands of tuning variables. However, it is still useful to review the analytical expressions for sensitivity to various tuning variables to understand the dependency on various parameters.

The sensitivity of switching energy to delay and leakage energy to delay (D) is given by the following two equations:

$$\frac{\partial E_{switching}/\partial w_i}{\partial D/\partial w_i} = -\frac{e_i}{\tau_{ref} \cdot (h_{eff,i} - h_{eff,i-1})} \tag{4.29}$$

$$\frac{\partial E_{leakage}/\partial w_i}{\partial D/\partial w_i} = \frac{E_{leakage}}{D} - \frac{D \cdot e_{leakage,i}}{\tau_{ref} \cdot (h_{eff,i} - h_{eff,i-1})} \tag{4.30}$$

In the above, e_i is the switching energy due to capacitances at stage i, τ_{ref} is a process independent time constant [12], $h_{eff,i}$ is the effective fanout of stage i, and $e_{leakage,i}$ is the leakage energy of gate i. When gate size is decreased, it decreases leakage current but also increases cycle time, D, which has the opposite effect of increasing leakage energy. The optimal design is reached when the sensitivity to sizing, supply voltage, and threshold are balanced as shown in Chapter 2.

It should be noted that Equations (4.29) and (4.29) calculate the derivative over every gate in the circuit which can be time consuming. Also, it would be difficult to use these expressions when calculating aggregate sensitivity for a large block that contains hierarchy.

4.6.2 Sensitivity to Supply and Threshold Voltage

The scaling of supply voltage and its effect on delay is modeled by Markovic by introducing a supply voltage scaling factor, K_V [12]:

$$K_V = \frac{V_{DD}^{low}}{V_{DD}^{ref}} \cdot \left(\frac{V_{DD}^{ref} - V_{on}}{V_{DD}^{low} - V_{ON}} \right)^{\alpha_d} \tag{4.31}$$

When supply is scaled down, the logical effort and parasitic delay increase. The scaling factor K_V modulates the delay to account for gates operating at lower supply which require increased logical effort to equalize delay across all stages.

The sensitivity to supply voltage adjustment is given by the following two equations:

$$\frac{\partial E_{switching}/\partial V_{DD}}{\partial D/\partial V_{DD}} = -\frac{2E_{switching} \cdot (1 - V_{ON}/V_{DD})}{D \cdot (\alpha_D - 1 + V_{ON}/V_{DD})} \tag{4.32}$$

$$\frac{\partial E_{leakage}/\partial V_{DD}}{\partial D/\partial V_{DD}} = -\frac{E_{leakage}}{D} \cdot \left(\frac{(1 - V_{ON}/V_{DD}) \cdot (1 + \gamma \cdot V_{DD}/V_0)}{\alpha_d - 1 + V_{ON}/V_{DD}} - 1 \right) \tag{4.33}$$

As supply voltage decreases from the technology reference supply, delay increases. As delay increases leakage energy also increases. However, with a decrease in supply, leakage power tends to decrease. Thus, supply scaling affects leakage energy in two different ways. Overall, with supply reduction, leakage energy tends to decrease which accounts for the negative sensitivity in Equation (4.33).

Switching energy is not a function of threshold voltage as was seen in Section 4.1. Hence, scaling threshold voltage only impacts delay and leakage energy. The sensitivity to threshold voltage adjustment is given by the following equation:

$$\frac{\partial E_{leakage}/\partial(\Delta V_{TH})}{\partial D/\partial(\Delta V_{TH})} = -\frac{E_{leakage}}{D} \cdot \left(\frac{V_{DD} - V_{ON} - \Delta V_{TH}}{\alpha_D \cdot V_0} - 1 \right) \tag{4.34}$$

The threshold voltage can be lowered in conjunction with reduced supply to minimize leakage energy and reduce switching energy. An optimum operating point is reached with the sensitivity to supply voltage scaling and sensitivity to threshold voltage scaling are balanced.

Energy was modeled using two independent components, and sensitivity models also use two components to account for switching and leakage energy. Design composition when using the models for energy, delay, and sensitivity must also carefully account for the two dominant components of energy: switching energy and leakage energy.

4.7 Design Composition

In Chapter 3, composition is used to build optimal aggregate sensitivity for large designs and in addition, energy and delay constraints are created in terms of lower level building blocks. This sections highlights the necessity to take into account accurate estimation of activity factors when forming energy models for high level blocks. In addition, leakage

energy must be estimated independently of dynamic energy. The two are then accumulated to form the total energy as described in previous sections. This section highlights the issues using the models developed earlier.

4.7.1 Delay

When composing small blocks to form larger blocks, delay is simply accumulated if the blocks are composed in series, unless registers are inserted. In Table 3.1 in Chapter 3, the total delay of an N-tap transpose filter is given by the delay of the multiply-accumulate block whereas the delay of an N-tap transverse filter is given the delay of the multiply block and $(N-1)$ add blocks. In the transpose architecture, each multiply-accumulate block is followed by a register and these are connected in series. The longest path from register to register is through a single multiply-accumulate block. In the transverse architecture, the longest path is through a series connection of one multiply and $(N-1)$ add blocks, hence the delay is much longer in the transverse architecture. However, the total energy per cycle is different, giving the transverse architecture lower energy per cycle.

In some cases, the delays of a series of connected blocks do not add linearly. This can be seen in the case of N cascaded ripple carry adder blocks. The critical path delay of the N adder system is not $N \cdot D_{adder}$ in this case. An example of two cascaded 4-bit ripple carry adders is shown in Figure 4.5. In the figure, the worst case critical path runs through 5 1-bit full-adders, or one 4-bit ripple carry adder and one 1-bit full-adder. When composing systems, the system delay model must include enough information to take into account this non-linear addition of delay. Optimal aggregate sensitivity depends on the total delay of a system because the weight depends on contribution of delay of each component to the total delay.

Different delay profiles were given in Table 3.1 for different filter architectures and it was shown that the constraint graph representation of the system highlights the differences in critical path delay for the two filter examples in Figure 3.8. The graph uses nodes to represent ports and edges to represent point-to-point connections between modules. The

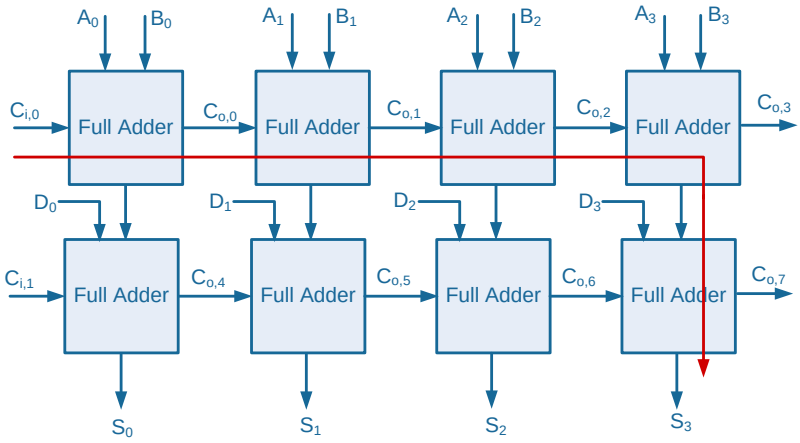


Figure 4.5. Two cascaded 4-bit ripple carry adders

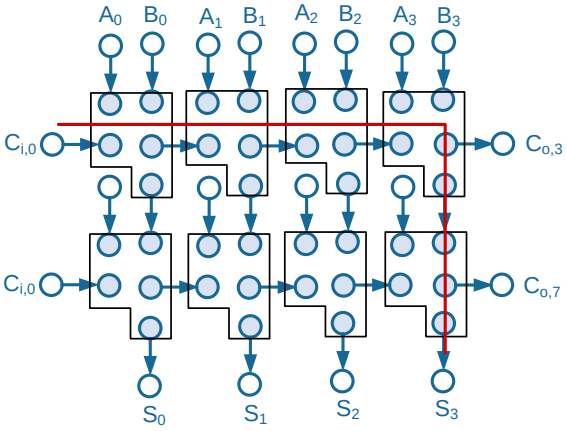


Figure 4.6. Constraint graph for two cascaded 4-bit ripple carry adders

constraint graph for the ripple carry adder example in Figure 4.5 is given in Figure 4.6. In Figure 4.6, it is clear that once the outputs of the first four full-adders are available, the full-adder blocks of the second ripple carry adder can proceed with their computation. The constraint graph representation clearly shows this parallel computation, resulting the red highlighted critical path.

4.7.2 Energy

Since energy per cycle is a function of total power and cycle time, activity factors and how long a block is idle during the entire cycle must be accounted for when estimating energy. The activity factors for small blocks can be estimated via vector simulation. This is done by the convex optimizer used at the leaf cell level. When blocks are composed, the activity factors for the blocks must scaled depending on the total cycle time for the particular architecture. The percentage contribution to total delay can be used as a scaling factor. That is, if a block's estimated activity is only 33% of the its total delay, t_{block}, then for a total system delay of $3 \cdot t_{block}$, the activity factor would be adjusted by $\frac{1}{3}$. This is only a rough method of scaling activity factors and may not be entirely accurate. However, when performing high-level rapid prototyping, and high-level energy-efficiency curves need to be generated early on in the design stage, this type of estimation is adequate.

The leakage energy for a block must also be adjusted based on its contribution to total system delay. If a block is idle for a significant amount of total system delay (say 90%), but only idle for 10% of block delay, then it will contribute more leakage energy to the entire system. Again, a leakage scaling factor can be assigned to the block based on its total contribution to system delay when modeling of system leakage energy in terms of lower level building blocks. This is a crude means of modeling but it is adequate for early stages of design where this design methodology is most effective.

4.8 Summary

This chapter summarized well-known models for energy, delay, and interconnect. They are used to estimate sensitivity and in the implementation of the convex circuit optimizer used at the leaf cell level. The delay and energy model used in the convex optimizer differ from the models used to develop a first-order model for sensitivity to sizing in Chapter 5. Energy and delay models using the Alpha-power Law are used to create analytical models for absolute sensitivity.

Design composition described in Chapter 3 requires careful modeling of energy for large systems in terms of lower level building blocks due to activity factor estimation and leakage energy that changes with composition.

Chapter 5

Sensitivity to Sizing

Independent of the choice of logic family or topology, optimized transistor sizing will play an important role in reduced power consumption.
– A. Chandrakasan et. al., JSSC, April 1992

5.1 Introduction

Circuit sizing plays an important role in the power-performance optimization of digital systems. Switching energy per transition is linearly proportional to the effective switching capacitance of a circuit, implying that the minimum energy solution, ignoring short-circuit currents, is one where gates are minimally sized. The minimum delay solution based on the method of logical effort advocates that gates should be sized such that each stage bears the same gate effort. These two solutions are disjoint, so arriving at a power-performance optimal circuit sizing solution is a non-trivial task. It requires a tradeoff analysis or equivalently, generation of a Pareto optimal solution set. As mentioned earlier in Chapter 2, computation of the Pareto optimal set is often NP-hard so various heuristics and simplifications have been employed in the past to arrive at reasonable solution sets for a particular sizing problem.

5.1.1 Constrained Circuit Sizing

The constrained circuit sizing problem has been the subject of many optimization papers, starting from the 1977 IBM paper by Ruehli et. al. [61]. Authors in [61] formulate the problem as a power minimization subject to timing constraints. Their models for delay and power result in singularities and discontinuities in the objective function. The discontinuities are resolved by a mapping of variables which results in a posynomial delay constraint. Since their gate delay model has singularities, the authors replace the model with a quadratic function for large gate widths. Gradient-based optimization is then used to iterate to a minimum power solution (i.e. gradients are zero at the minimum). However, their solution method is plagued by slow convergence and does not scale well to large designs.

In 1985, Fishburn and Dunlop formulated the circuit sizing problem using RC equivalent circuits to model transistors and used the Elmore delay model for gates [21]. The resulting formulation is shown to be posynomial in transistor and wire widths, and thus can be converted into a convex problem by a simple mapping of variables. Fishburn and Dunlop first run a static timer to estimate arrival and required times at each node in the circuit. Then, sensitivities for each transistor are calculated, and a greedy sensitivity-based sizing approach is employed iteratively until a fixed point is reached at the global minimum.

Soon after Fishburn and Dunlop showed that the circuit sizing problem could be formulated as a posynomial in transistor and wire sizes, Sapatnekar et. al. proposed an exact solution to the convex sizing problem in [34]. In 1999, Conn et. al. [22] moved away from the inaccuracies of approximating the logic gate as an RC circuit. Instead, they combined multiple methods from previous works to optimize transistor sizes. A static-timing based formulation was implemented with the objective to minimize the maximum of all path delays subject to area and transistor width constraints. Rise and fall times are accounted for along with timing slew. A non-linear optimizer based on augmented Lagrangian methods is used, and fast transient simulation was employed to compute sensitivity information. Conn et. al. base their sensitivity computation on adjoint and direct gradient computation

methods. The advantage of their optimization is that any type of custom circuitry can be accommodated.

In more recent convex formulations [36, 37], geometric programming is used to efficiently solve the convex circuit sizing problem. Boyd et. al. in [36] combine geometric programming, dynamic programming, and static timing analysis to solve very large circuit and wire sizing problems efficiently. In [37], on-the-fly static timing is combined with geometric programming and tabulated delay models to solve a similarly formulated problem. The advantage of the work in [37] is that once the delay models for gates are created using circuit simulation, any type of custom circuitry can be accommodated – from static CMOS gates to dynamic logic.

In each of the works mentioned above, the constrained gate sizing problem for a particular path in a combinational circuit is generally formulated in Equation (5.1). Additional constraints can be included for better accuracy.

$$\text{minimize } D_{path}(W_i) \text{ subject to } \begin{cases} E \leq E_{max} & \\ C_{in} \leq C_{in,max} & \\ W_i \geq 1 & i = 1 \ldots N \\ t_{slope,j} \leq t_{slope,max} & \forall j \in FI(i) \end{cases} \tag{5.1}$$

The objective in Equation (5.1) minimizes the total delay of a given path, D_{path}, which is a function of the relative gate sizes, W_i, and the input signal slopes, $t_{slope,j}$ arising from the fan-in of gate i. The energy is given by E which includes both switching and leakage energy. The energy budget is given by E_{max}. The input capacitance of the path, C_{in} is limited to be less than the maximum for the design, $C_{in,max}$. The total number of gates along the given path is N. A combinational circuit consists of many paths, hence the total circuit delay is given by the maximum of all path delays: $D_{total} = \max\,(D_{path_j},\ j = 1 \ldots M)$. Thus, the overall objective for the circuit sizing problem is to minimize D_{total} subject to the constraints shown in Equation (5.1). A recursive formulation based on dynamic programming can be employed to find the delay of a circuit without enumerating all of its paths [36]. The constrained sizing optimization in Equation (5.1) can also be modeled as an energy minimization subject to delay constraints [17] (see Chapter 4, Equation (4.13)).

When circuit sizing is combined with other tuning variables such as supply voltage and threshold voltage, it can be shown that significant savings in energy dissipation are obtained while maintaining performance [11, 12, 14, 62, 63, 64, 65]. The relative effectiveness of tuning one variable over the other to maximize energy-efficiency is captured by sensitivities to individual tuning parameters. If sensitivity information is used as an optimization guide, it allows systematic traversal of the tradeoff space, leading to a quick discovery of the optimal system design as was described in Chapter 3.

5.1.2 Chapter Overview

This chapter introduces an alternative method for estimating or calculating sensitivity to sizing. As highlighted in Chapter 3, sensitivity is a necessary ingredient to hierarchical power-performance optimization. Once sensitivities to tuning variables such as gate size are available, it can be used as a guide for optimization. A simple first order model for sensitivity to sizing is presented. It is based on physical parameters of a circuit that are readily available to designers through either CAD tools or via simple calculations. Specifically, the components of the model include total gate capacitance, total wire capacitance, and input capacitance. The model obviates the need to calculate derivatives or generate large numbers of energy-efficiency curves.

The first section reviews previous attempts to calculate and use sensitivity in the context of the constrained circuit sizing problem. The circuit sizing tool described at the end of the section is used in this work to develop models for sensitivity. Section 5.3 begins with an analytical approach and then moves to a numerical approach for developing a model for sensitivity to sizing. Circuit benchmarks, starting with an inverter chain and ending with digital filters, are used to develop the model for sensitivity to sizing. The optimization of small combinational blocks (e.g. inverter chain and adder) is accomplished using a Matlab-based custom circuit optimizer developed in [17]. Next, the model is verified for larger blocks and for synthesis optimized blocks. Section 5.8 presents the model's limitations. The final section summarizes the chapter.

5.2 Gradient-Based Circuit Sizing

In the late 1980s and 1990s, sensitivity analysis was applied to constrained circuit sizing optimization in tools such as TILOS [21] and Einstuner [22]. In 1985, Fishburn and Dunlop applied convex optimization to transistor sizing of a combinational circuit. Given a set of N transistors of size $x_1, x_2, \ldots, x_N$, the problem is to find the optimal size of each transistor given a constraint on either area or delay in the form the required clock period. They termed their CAD tool TILOS; it took as input a transistor connectivity file and an input/output delay file. The output was a transistor connectivity file. Under the hood, TILOS used a static timing analyzer to extract all relevant timing paths. The problem was formulated in three different ways: (1) minimize area, which is correlated to the sum of transistor sizes, subject to a delay constraint; (2) minimize delay subject to an area constraint; (3) minimize the product of area and delay to the power of some integer, which allows designers to put a weighting factor on delay.

In [21], the authors show that the sizing problem is posynomial in transistor and wire widths, and can be converted to a convex optimization problem by a simple mapping of variables. The delay through a single path is modeled as a posynomial, hence making the delay constraint or objective function (depending the optimization) convex. Since the area is simply the sum of the transistor sizes, all three formulations are convex in nature and geometric programming is used to solve the optimization. Fishburn and Dunlop use sensitivity information to decide which transistor must be increased in size: the one with the largest sensitivity to increasing performance is selected for up-sizing. The sensitivity is calculated on the fly by fixing all other transistor sizes and increasing the size of one critical transistor. The resulting sensitivity is given as a function of resistance and capacitance of a unit-sized FET, the resistance and capacitance of the driven RC chain, and the size of the selected transistor [21]. For efficiency purposes, not all paths are enumerated, neither are all sensitivities stored. Recently, the formulation that Fishburn and Dunlop proposed has been used as a foundation for further work on gate sizing by other researchers [34, 33, 37]. Fishburn and Dunlop use a heuristic method to solve the optimization whereas Sapatnekar

et. al. [34] employ exact methods. The drawback of methods in [21] and [34] is that they both suffer from the inaccuracy of approximating a logic gate by an RC circuit.

Conn et. al. in [22] alleviated this problem by using simulation-based static timing analysis, where the delay across a single path is the maximum arrival time at the output of a gate. The maximum is transformed into a continuous differentiable objective function by the addition of an auxiliary variable. They take into consideration rising and falling arrival times, and rising and falling slew. Their approach relies on nonlinear optimization and incremental time-domain gradient computation to optimally size a circuit to minimize delay under area constraints. Unfortunately, their more accurate modeling of delay results in a non-convex optimization problem, and has the danger of falling into a local minimum. A simulator is used to evaluate path delays and updates the rising and falling arrival times and slews which are then passed to the nonlinear optimizer. Gradients are also computed via the circuit simulator using adjoint methods. Sensitivity to transistor sizing (gradients) are used in the nonlinear optimizer but the details are not clarified by the authors. The authors simply state that the nonlinear optimizer uses a Lagrangian merit function and a penalty term consisting of a weighted sum-of-squares of the constraints [22]. In 2005, the nonlinear optimizer was updated to use interior point iterative methods to arrive a final solution [66].

More recently, a similar static-timing-based optimizer using a posynomial formulation based on logical effort [43] has been implemented in Matlab and is used to minimize delay in custom datapath circuits under given energy constraints [67, 37, 17]. The tool can be extended to generate energy-delay tradeoff curves to help designers analyze various architecture and implementation choices. The determination of sensitivities from the generated energy-efficient curves is left to the designer. Similar geometric programming based formulations have also appeared in [33] and are solved by optimizers using interior-point methods. The Matlab tool described in [17] is used in this work to optimize small combinational circuits and develop models for sensitivity to sizing. Models for energy, delay and area used in the optimization are described in Chapter 4.

In each of the above optimization methods, gradients or sensitivity analysis is necessary

in determining the direction of the next iteration (i.e. which transistor, gate, or path is selected for resizing). In the design methodology developed here, hierarchical optimization uses sensitivity as a guide in determining system optimality and tradeoffs. The next section describes how sensitivity to sizing may estimated by simple circuit properties such as gate capacitance and wire capacitance.

5.3 Gate Capacitance, Wire Capacitance, and Sensitivity

Device sizing is characterized by the total gate capacitance to wire capacitance ratio (C_{gate}/C_{wire}), assuming that the path fanout (C_{load}/C_{in}) is a simple function of C_{in} [16]. This is useful since C_{gate}/C_{wire} can be easily estimated after the technology mapping stage of synthesis. An accurate value is available to the designer after the design has been initially placed and routed. After synthesis and during the floor-planning stage, a reasonable estimate of total wiring capacitance is usually available along with sizes of individual gates which will remain relatively constant during place and route. As C_{gate}, C_{wire}, C_{in}, and C_{load} are physical parameters of any circuit that are readily available through tools or design calculations, a model for sensitivity to sizing that uses these parameters can be extremely useful in determining the optimality of a design. The remainder of this chapter explores the relationship between sensitivity to sizing, $S(W)$, and the ratio of total gate capacitance and total wire capacitance via optimization of benchmarks ranging from a simple inverter chain to digital filters.

Interconnect introduces a complication to the logical effort model for delay [43]. Since the wire capacitance remains fixed, the effort required at the gate driving the wire will not equal the effort at the gate following the wire. Hence, for paths that include wires, the efforts across the stages in the path are not equal. The gate driving the wire will have a higher effort than the gate at the end of the wire [43].

The effect of interconnect on path efforts is modeled by Sutherland et. al. in [43] by a branching effort at the wire driving a gate; it is given as: $(C_{gate} + C_{wire})/C_{gate}$. Since this branching effort changes whenever gates are resized, an approximation of the branching

effort is often required to optimally size a path for minimum delay [43]. Iteration is used to determine the optimal number of stages as the branching effort at the wire is initially unknown.

Short wires in a design are usually treated as parasitic capacitance. A path containing long wires is split into two parts: the first part drives the wire and the second receives input from the wire. Each part can be designed independently, lumping the wire capacitance as either a load or as an input capacitance. When wires become very long, the resistance of a wire begins to impact delay and energy. Since wire delay scales quadratically with wire length [6, 10, 43], long wires are usually broken into smaller sections by inserting repeaters. When repeaters are inserted optimally, the delay along a wire scales linearly with wire length [6].

Medium wires with capacitance that is comparable with gate capacitance pose a difficult design problem. The wire branching effort becomes a strong function of both the gates and the wire. In this case, wire capacitance cannot be treated as parasitic capacitance, nor can it be lumped as load or input capacitance. Sizing a path with medium wires requires a numerical solution to a polynomial function [43]. In newer technologies, this is even a bigger issue as wire resistance starts playing a role in increased delay across medium wires in addition to long wires.

In the following subsection, the logical effort method of sizing gates is augmented to include interconnect. It shows the dependence of sensitivity to sizing on wire, gate, and input capacitance of a circuit. The simple two-stage inverter chain with interconnect is used as a starting point for the analytical derivation.

5.3.1 Analytical Derivation of Sensitivity to Sizing

The method of logical effort for optimizing the sizing of logic gates (for minimum delay) along a path without wires is based on equal stage efforts, $g_i h_i = g_j h_j$, where g_i is the logical effort of gate i and h_i is the electrical effort of gate/stage i [43]. However, when interconnect is present in a path, stage efforts are not necessarily equal. Three distinct

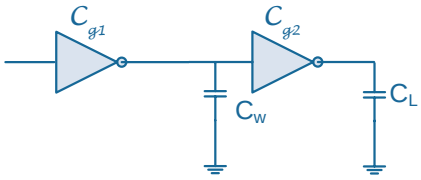


Figure 5.1. Two-stage inverter chain with wire side load

methods of sizing a simple two-stage inverter chain (shown in Figure 5.1) in the presence of interconnect have been published [10, 43, 68, 69, 70]. The three methods yield different solutions.

Interconnect as a Side Load - Short Interconnect

Sutherland et. al. [43] and Horowitz [68] state that the minimum delay sizing problem that includes side loads either from wire capacitance or loading of non-critical gates is difficult to solve exactly. However, it is easily solved approximately. In both solutions, the approximate minimum delay sizing solution neglects wire resistance.

The logical effort solution from [43] uses branching effort at the output of the driving inverter to model the effect of interconnect. The branching effort is given as $\frac{\text{on path capacitance + off path capacitance}}{\text{on path capacitance}} = \frac{C_{gate}+C_{wire}}{C_{gate}}$. This ratio is used to gauge the difference between gate capacitance along a path and wire capacitance which stays relatively fixed. If the contribution of wire capacitance to path delay is small and pat delay is dominated by gate capacitance, then the ratio is close to unity. In this case, it is reasonable to treat wires as additional parasitic capacitance when minimizing for delay. The total parasitic capacitance includes parasitic wire capacitance and parasitic diffusion capacitance [43]. Stage efforts are equalized and the best stage effort is slightly over four for paths with reasonably short wires [43]. Horowitz in [68] gives a similar solution. The delay equation for the two-stage inverter chain with interconnect that acts as parasitic capacitance is given by:

$$D = \frac{C_{g_2}}{C_{g_1}} + \frac{C_L}{C_{g_2}} + 2 + p_{wire} \tag{5.2}$$

Minimizing Equation (5.2) with respect to the electrical effort gives the same solution as the minimum delay solution without interconnect: equal stage efforts, $\frac{C_{g2}}{C_{g_1}} = \frac{C_L}{C_{g_2}}$. The size of the second gate is then given by $C_{g_2} = \sqrt{C_L C_{g_1}}$ and the stage effort is given by: $f_i = \sqrt{\frac{C_L}{C_{g_1}}}$. When C_w is small compared to C_{g_2}, its effect on the path effort is small [43, 68]. Equation (5.2) is a good approximation only if the wire capacitance is very small relative to gate sizes and thus, can be treated as parasitic capacitance.

Interconnect as A Side Load – Medium and Long Interconnect

In the case of very long wires, a simple approximation is to split the chain into two pieces: the first piece drives the wire and treats it as an output load [43, 68, 69]. This approach is detailed in [69]. The stage effort of the gate driving the wire is given by $f_1 = \frac{C_{g_2}+C_w}{C_{g_1}}$ and the stage effort of the gate after the wire is given by $f_2 = \frac{C_L}{C_{g_2}}$. In [69], the authors show that for an inverter chain with n inverters preceding the wire and m inverters after the wire, the optimal delay cannot be solved analytically but bounds on the optimal stage efforts are presented by assuming equal stage efforts along gates preceding the wire and equal stage efforts among gates after the wire. However, this is still an approximation as the effort of the gate driving the wire will not necessarily be equal to the gates preceding the gate driving the wire [43, 68]. The main issue with this approximation for long wires is that it does not include branching effort due to the wire. It must be included at the node driving the wire as wire capacitance affects the stage effort of the gate following the interconnect as well as the gate driving the wire. Current will be split amongst the wire capacitance side load and the path following the interconnect. The proper way to model delay for medium and long wires with side load capacitance is as follows.

If C_w is comparable to the gate capacitance, C_{g_2}, and load capacitance C_L, then the branching effort for wires $(C_{gate}+C_{wire})/C_{gate}$ is a strong function of both wire capacitance and gate capacitance, and must be included in the delay equation as shown in Equation (5.3).

$$\begin{aligned} D &= g_1 h_1 b_1 + p_1 + g_2 h_2 b_2 + p_2 \\ D &= \frac{(C_{g_2}+C_w)}{C_{g_1}} + \frac{C_L}{C_{g_2}} \cdot \frac{(C_w+C_{g_2})}{C_{g_2}} + 2 \end{aligned} \tag{5.3}$$

Minimizing Equation (5.3) with respect to gate size results in the following optimal sizing condition:

$$C_{g_1} = \frac{C_{g_2}^2}{C_L} \cdot \left(\frac{C_{g_2}}{C_{g_2} + 2C_w} \right) \tag{5.4}$$

When wire capacitance is zero, Equation (5.4) reduces to the optimality condition for a two-stage inverter chain without interconnect, and results in $C_{g_2} = \sqrt{C_{g_1} C_L}$. The optimal sizing for C_{g_2} in the presence of interconnect is given by a solution to the following cubic equation:

$$C_{g_2}^3 - C_{g_1} C_L C_{g_2} - 2C_L C_w C_{g_1} = 0 \tag{5.5}$$

The size for the gate following the interconnect is given by:

$$C_{g_2} = \sqrt[3]{C_L C_w C_{g_1} + \sqrt{(C_L C_w C_{g_1})^2 + \frac{(C_{g_1} C_L)^3}{27}}} - \sqrt[3]{-C_L C_w C_{g_1} + \sqrt{(C_L C_w C_{g_1})^2 + \frac{(C_{g_1} C_L)^3}{27}}} \tag{5.6}$$

Note that the two stage efforts are not equal. The driving gate will require higher effort as interconnect capacitance increases relative to gate capacitance. As interconnect capacitance increases relative to the load capacitance, the stage effort of the second gate must also increase. The dependence on C_w is shown in Equations (5.4) and (5.6). In newer technologies, delay due to wire capacitance and wire resistance are contributing a significant amount to the cycle time of a system [6], hence models for delay and energy must include their effect for accurate architecture tradeoff analysis.

Wire Resistance

The delay dependence on the length of the wire is seen by representing the interconnect along a path by the Elmore π-delay model for wire segment delay [10]. This is shown in Figure 5.2. Based on the Elmore delay model (which includes wire resistance), the total delay long the two-stage chain with interconnect is then given by [10]:

$$D = 0.69R_1(C_{p_1} + C_{g_2}) + 0.69(R_1 c_w + r_w C_{g_2})L + 0.38 r_w c_w L^2 \tag{5.7}$$

In Equation (5.7), L is the length of the wire, r_w and c_w are the wire resistance and

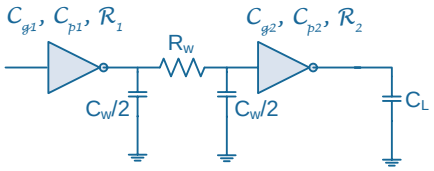


Figure 5.2. Two-stage inverter chain with Elmore wire segment delay

capacitance per unit length. Equation (5.7) clearly shows the dependence of delay on wire length. If the wire is very long, then the quadratic term becomes quite large. In order to mitigate this, repeaters are usually inserted along a long wire so that the dependence of delay on wire length becomes linear [6].

The previous minimum delay sizing solutions neglected wire resistance. In the following, wire resistance is included using the Elmore delay model. Authors in [70] use the π-delay model for wire segment delay and the method of logical effort to arrive at a result that shows how stage efforts change in the presence of interconnect. However, their derivation neglects branching efforts due interconnect. The branching factor due to interconnect is augmented to the equation for delay given in [70]. Equation (5.8) derives the delay model for the two-stage inverter chain with interconnect in the presence of wire capacitance and wire resistance.

$$\begin{aligned} D &= \textstyle\sum_{i=1}^{N} g_i \cdot (h_i + h_{w_i}) + g_{i+1}h_{i+1} \cdot \left(\frac{C_{g_{i+1}}+C_{w_i}}{C_{g_{i+1}}}\right) + (p_i + p_{w_i}) \\ D &= h_1 + h_{w_1} + p_1 + p_{w_1} + h_2\left(\frac{C_{g_2}+C_{w_1}}{C_{g_2}}\right) + p_2 \\ D &= h_1 + \frac{C_w}{C_{g_1}} + \frac{R_w \cdot (0.5C_w + C_{g_2})}{R_o C_o} + \frac{C_L}{h_1 C_{g_1}} \cdot \left(\frac{h_1 C_{g_1} + C_w}{h_1 C_{g_1}}\right) + 2 \end{aligned} \tag{5.8}$$

In Equation (5.8), $h_1 = \frac{C_{g_2}}{C_{g_1}}$ represents the electrical effort for stage 1; C_{g_i} is the input capacitance of gate i; R_w and C_w is the wire resistance and capacitance of wire segment; h_{w_i} is termed the *capacitive interconnect effort* which is $\frac{C_{w_i}}{C_{g_i}}$; $R_o C_o$ is the delay of a minimum-sized inverter. If the wire is short, then wire resistance can be neglected and the resulting equation is given by:

$$D = h_1 + \frac{C_w}{C_{g_1}} + \frac{C_L}{h_1 C_{g_1}} \cdot \left(\frac{h_1 C_{g_1} + C_w}{h_1 C_{g_1}}\right) + 2 \tag{5.9}$$

When the two-stage inverter chain with wires is optimized for delay given in Equation (5.8) (derivative of delay with respect to gate size is set to zero), the resulting optimality condition is given by:

$$\left(1 + \frac{R_w C_{g_1}}{R_o C_o}\right) \cdot h_1 = h_2 \cdot \left(\frac{C_{g_2} + 2C_w}{C_{g_2}}\right) \tag{5.10}$$

The two stage efforts are different due to the wire resistance and wire capacitance. If wire resistance is not significant (as for short wires), then the optimality condition reduces to the one in Equation (5.4). When interconnect is not present in the path, then the optimality condition reduces to the one of equal stage efforts. The above analysis is extended for general gates along a path in [70] and here it is augmented with appropriate branching factors. The derivation is not presented here, but the resulting optimality condition for minimum delay is given by:

$$\left(g_i + \frac{R_{w_i} C_{g_i}}{R_o C_o}\right) \cdot h_i = g_{i+1} \cdot \left(h_{i+1} \cdot \left(\frac{C_{g_{i+1}} + 2C_{w_i}}{C_{g_{i+1}}}\right) + \frac{C_{w_{i+1}}}{C_{g_i}}\right) \tag{5.11}$$

In Equation (5.11), the stage efforts are modulated by wire resistance and wire capacitance. The stage effort for the gate driving the wire will increase as wire resistance increases, and effort of the stage following the wire will increase as wire capacitance increases relative to the size of the stage. In the absence of interconnect, the optimality condition for minimum delay reduces to that of equal stage efforts.

Constrained Delay Minimization

Sensitivity analysis is based on constrained delay minimization or constrained energy minimization as explained in Chapter 2.

Energy is given by switching and static energy as summarized in Chapter 4. If it is assumed static power can be neglected for the two-stage inverter chain example, then the energy is approximately given by the switching energy which is proportional to the switched capacitance and square of the supply voltage:

$$E \approx \alpha \cdot (C_w + C_{g_2}) \cdot V_{DD}^2 \tag{5.12}$$

The general constrained circuit sizing problem given in Equation (5.1) cannot be solved analytically. A numerical optimizer is required for a complete solution [48]. The main feature of the optimal solution is that stage efforts increase along the chain towards the load capacitance. In the presence of interconnect, the constrained delay minimization problem is further complicated.

In the case of the two-stage inverter chain example in Figure 5.2, the constrained delay minimization can be solved analytically using Lagrangian methods. Since the Lagrange multiplier provides a lower bound on the sensitivity, the resulting optimal Lagrange multiplier can be used to understand the dependency of sensitivity on gate capacitance and wire capacitance. The Lagrangian for the problem neglecting wire resistance is given by:

$$L(C_{g_2}, \lambda) = \frac{C_{g_2}}{C_{g_1}} + \frac{C_w}{C_{g_1}} + \frac{C_L}{C_{g_2}} \cdot \left(\frac{C_{g_2} + C_w}{C_{g_2}} \right) + 2 + \lambda \cdot (\alpha \cdot (C_w + C_{g_2}) \cdot V_{DD}^2 - E_{max}) \quad (5.13)$$

By setting $\frac{\partial L}{\partial C_{g_2}} = 0$, the Lagrange multiplier is found to be:

$$\lambda = \frac{1}{\alpha \cdot V_{DD}^2} \cdot \left[\frac{C_L}{C_{g_2}^2} \left(\frac{C_{g_2} + 2C_w}{C_{g_2}} \right) - \frac{1}{C_{g_1}} \right] \quad (5.14)$$

In Equation (5.14), λ is a function of the branching effort, and the inverse of the input capacitance to C_{g_2}. When wire resistance is included, an additional term $\frac{1}{\alpha \cdot V_{DD}^2} \cdot \frac{R_w}{R_o C_o}$ is subtracted from the Equation (5.14). As the branching factor ratio becomes a strong function of wire resistance, the sensitivity to sizing will increase for a given set of gate capacitances. As the wire length increases, wire resistance will increase as well causing a decrease in sensitivity to sizing for the second stage. As the capacitance of the gate driving the gate increases, the sensitivity to sizing will increase slightly. Equation (5.14) indicates that sensitivity to sizing is a function of wire branching effort, input capacitance, wire capacitance, and wire resistance.

Constructing an analytical equation in terms of C_{gate}/C_{wire} for the above case is a long exercise and does not yield an interesting result as most paths in circuits are longer than two inverter stages, contain diverse gates, and include interconnect at multiple nodes along a path. For diverse and longer paths, it is much more convenient to use a numerical optimization to develop a model for sensitivity. The next subsection describes the experimental setup.

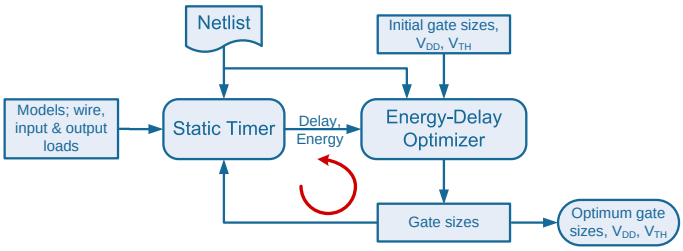


Figure 5.3. Convex model based optimizer built in Matlab

5.3.2 Numerical Approach to Modeling Sensitivity to Sizing

The model for sensitivity to sizing in terms of total gate capacitance, input capacitance, and wire capacitance is developed through the use of a custom circuit optimizer which is described in detail in [17]. A simplified diagram of the optimizer is shown in Figure 5.3. The optimizer is based on convex models for energy and delay as described in Chapter 4. The netlists for the various benchmarks are described in a format similar to HSPICE, tabulated delay models for each gate and technology are created using circuit simulation. The drawbacks of using tabulated models are described in Chapter 4. The netlist and models are input to the optimizer which produces the optimal gate sizes for minimum delay subject to an energy constraint. Wire capacitance is accurately measured through one pass of place and route, and then fed back into the optimizer as input. The optimizer neglects wire resistance in the optimization [17]. At the completion of the optimization, after the optimal gate sizes are available, Matlab scripts are used to calculate the sensitivity to sizing for each point on the energy-efficiency curve for different input capacitances. Then sensitivity to sizing is plotted against the ratio of total gate capacitance and total wire capacitance for a given input capacitance and load capacitance.

The benchmark studies presented in the following sections show that there is a simple first-order linear relationship between sensitivity to sizing and the pair $(C_{gate}/C_{wire}, C_{in})$. Opportunities for improvement in the energy-efficiency of a given design through gate sizing at different levels of design abstraction can be quickly identified using this model for sensitivity. The simplicity of the approach obviates the need to calculate the energy-efficient

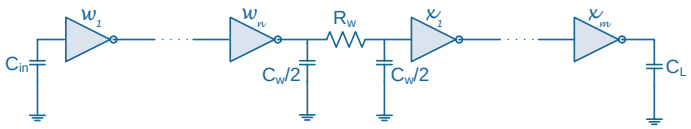


Figure 5.4. Inverter chain with wire capacitance and resistance

curves in the energy-delay coordinate space and lends itself well to an automated sizing optimization for an energy-constrained circuit within a synthesis-based design flow for ASICs.

5.4 Inverter Chain

The first simple benchmark is an inverter chain consisting of 20 inverters, with a wire capacitance and resistance placed in the middle of the chain, after the tenth inverter, as shown in Figure 5.4. The inverter chain is implemented in a 90nm standard CMOS process. The 90nm CMOS inverter has a 5.6ps/fanout delay and 7.1ps self-loaded delay at 1.0V supply. The intrinsic capacitance and resistance are 1.6fF and 11kΩ, respectively. The gate and diffusion capacitance are 0.73fF and 0.93fF, respectively. Wires in the 90nm process have a sheet resistance of 0.07$\Omega/\square$ resulting in a wire resistance of 0.01Ω/μm. The total wire capacitance per length is given as 0.35fF/μm. The wire capacitance stays fixed at 100fF during each optimization.

Energy-efficiency curves are generated for input capacitances ranging from 2fF to 30fF. The total load at the end of the chain is set to be four times that of the input capacitance. The energy-efficiency curves are shown in Figure 5.5 for the different values of input capacitance. The minimum delay across the chain is 0.25ps. The lowest curve in the plot is for $C_{in} = 2fF$ and the highest curve on the graph is for $C_{in} = 30fF$.

Matlab scripts are used calculate the total wire capacitance, total gate capacitance, and sensitivity for a given input capacitance. The sensitivity is then plotted against the ratio of total gate capacitance to total wire capacitance. Figures 5.6 shows a subset of the results of the optimization for the inverter chain with a 100fF wire capacitance and a fixed fanout

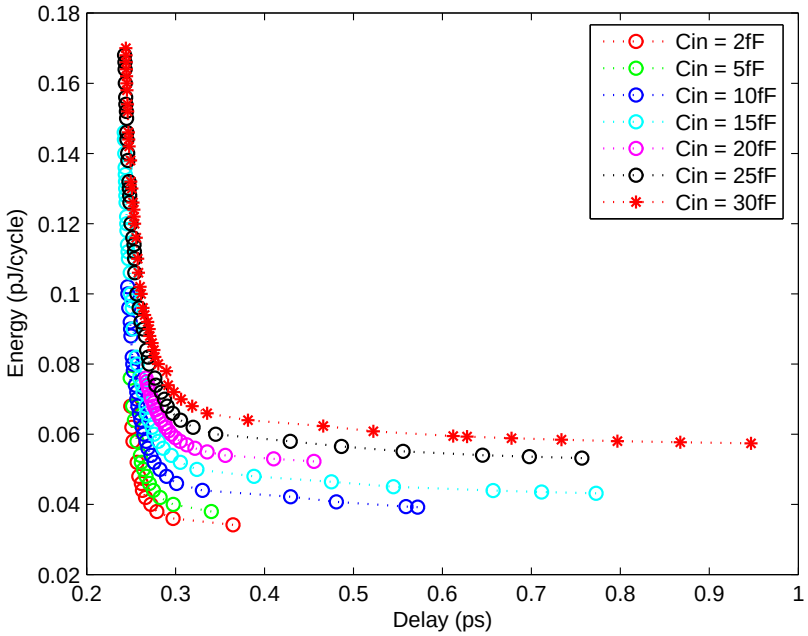


Figure 5.5. Inverter energy-efficiency curves for fixed wire capacitance of 100fF

relationship (i.e. $C_{load} = 4 \cdot C_{in}$). The plots show sensitivity to sizing plotted against the ratio of total gate capacitance to total wire capacitance for C_{in} = 5fF, 10fF, 15fF, 20fF, 25fF, and 30fF. As illustrated in each of the plots, there is a clear linear relationship between C_{gate}/C_{wire} and sensitivity to sizing. The x-intercept gives the minimum energy point where gates are minimally sized. The dispersion around the linear function for $C_{in} = 25fF$ is due to the inability of the custom optimizer to converge to an optimal point for sensitivities higher than 5. This is case is special because the wire capacitance is exactly to the load capacitance which causes problems for the optimizer. This phenomena was also seen for other cases where wire capacitance was exactly the same as load capacitance.

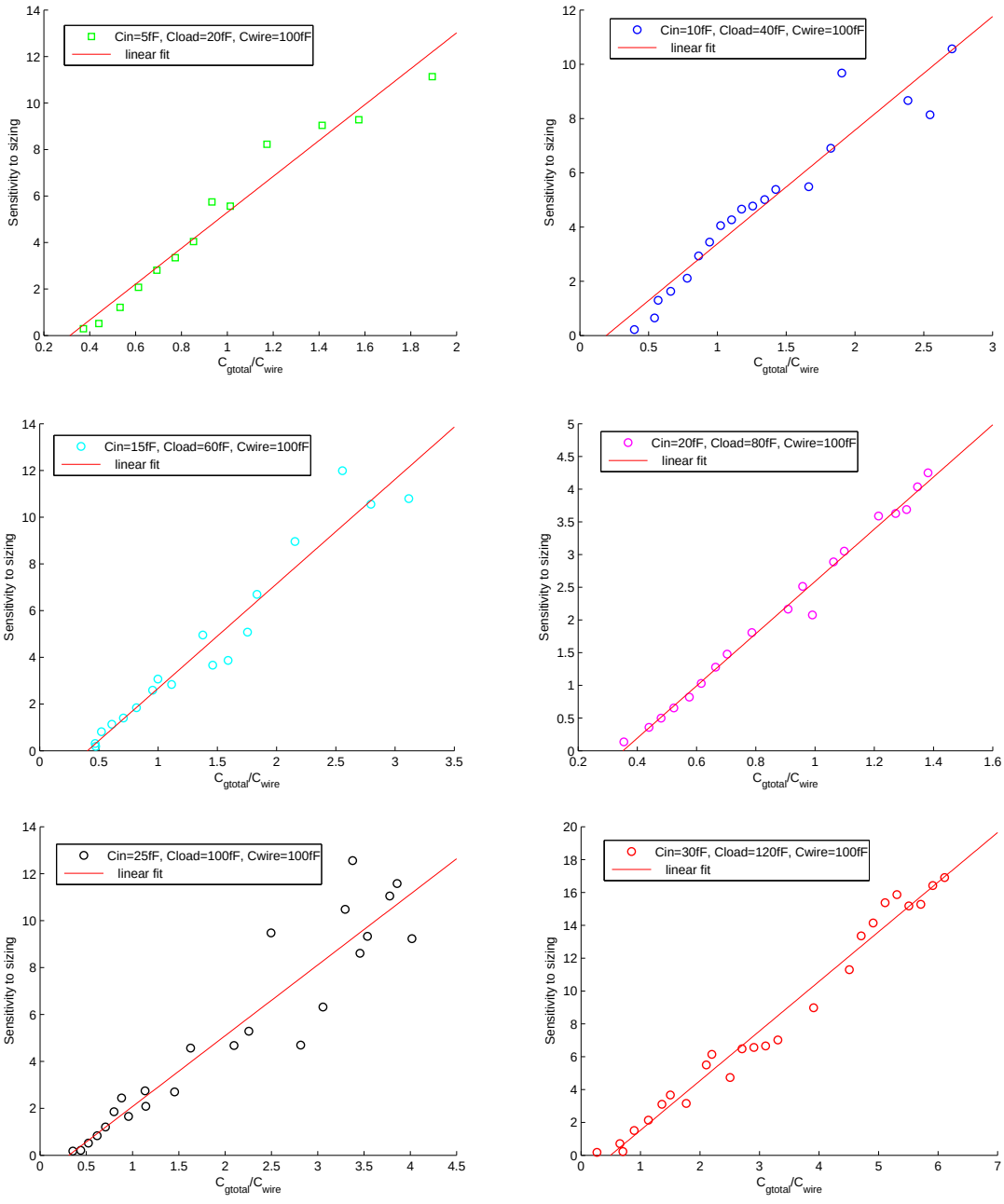


Figure 5.6. Sensitivity versus C_{gate}/C_{wire}

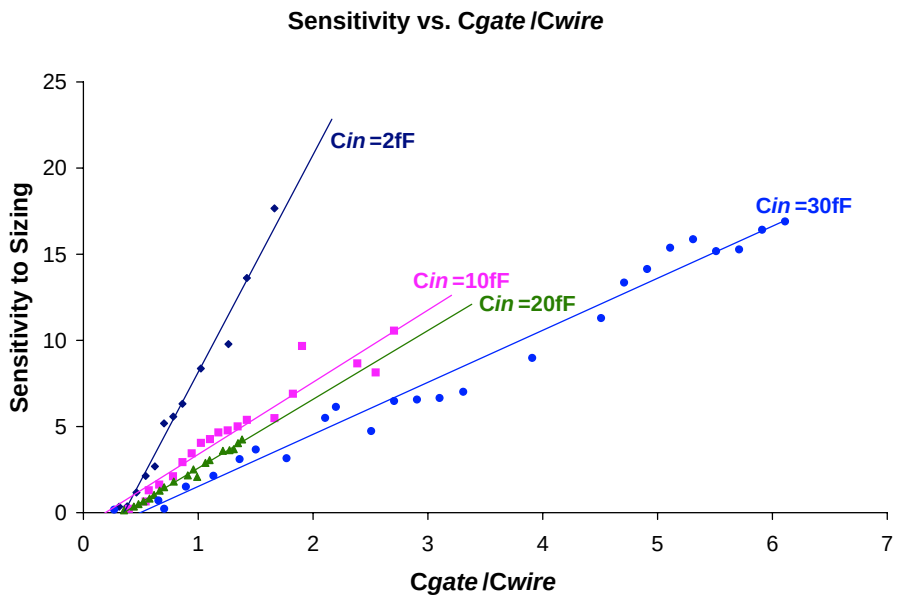


Figure 5.7. Sensitivity versus C_{gate}/C_{wire}

Figure 5.7 shows the curves for $C_{in} = 2fF$, $C_{in} = 10fF$, $C_{in} = 20fF$, and $C_{in} = 30fF$ plotted on the same axis which highlights the changes in the slopes of the curves as input capacitance is increased. As C_{in} increases (causing C_{load} to increase due to the fixed fanout relationship), gates are sized up. When total gate capacitance increases in relation to the total fixed wire capacitance, the ratio $(C_{gate}+C_{wire})/C_{gate}$ starts approaching unity. Hence, for a fixed delay target, the amount of additional energy required to compensate for loss in delay in the wire is reduced. This is shown by the smaller slope of the sensitivity versus C_{gate}/C_{wire} plot for increasing C_{in} in Figure 5.7. Less additional energy is required for a percentage increase in performance when the input and load capacitance increase relative to the fixed wire capacitance.

When input capacitance is left to vary up to a maximum value and the load capacitance remains fixed, a very clear linear relationship between sensitivity to sizing and C_{gate}/C_{wire} emerges as shown in Figure 5.8, for different fixed wire capacitances. As wire capacitance increases, the slope of the linear curve also increases. The gate driving the wire must work harder to compensate for the increased delay across the wire to meet the target delay constraint. The driving inverter must then be up-sized causing an up-size of the

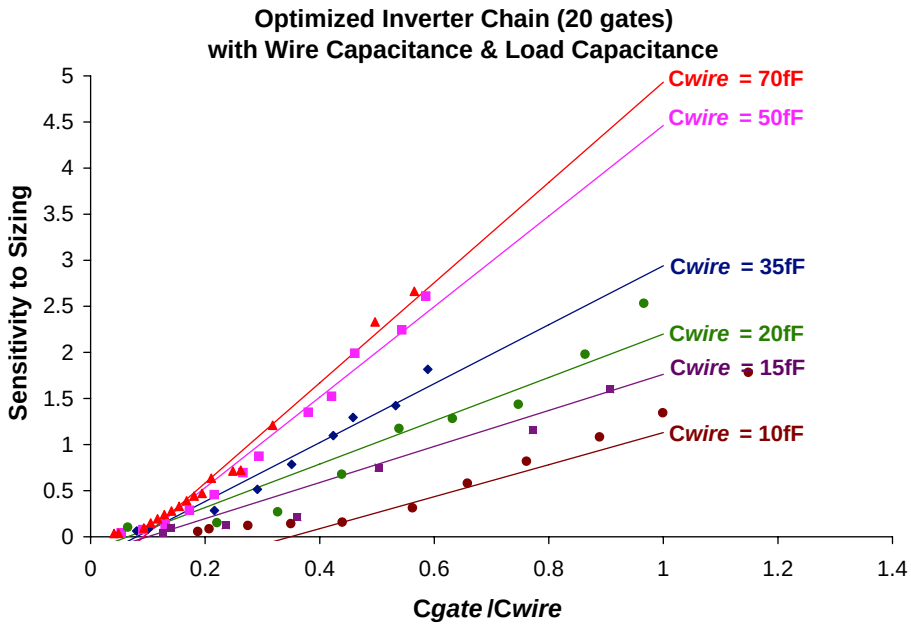


Figure 5.8. Sensitivity versus C_{gate}/C_{wire}: fixed C_{wire}, varying C_{in}

other inverters preceding the driving inverter resulting in an increase in energy. When wire capacitance increases, more energy must be expended for each percentage increase in performance.

Figure 5.9 shows the energy-efficiency curves corresponding to the inverter chain optimization for different wire capacitances. The minimum energy point and the energy-delay-squared point (ED^2) on the knee of the curve are highlighted to show that only these two points are required to create a linear model for sensitivity in terms of C_{gate}/C_{wire}. If a designer knows the minimally sized solution for a particular design and can calculate the total gate and wire capacitance for the EDP or ED^2 points, then sensitivity to sizing can be approximated without having to calculate derivatives or without having to generate multiple points on the energy-efficiency curve.

There is another way to model the sensitivity. The x-intercept of the graphs in Figure 5.7 and Figure 5.8 is the minimum energy point where the gates are minimally sized. The slope is based on the ratio $(C_{gate} + C_{wire})/C_{gate}$, C_{wire}, and C_{in}. As observed in the two graphs, the slope increases as wire capacitance increases but decreases as C_{in} increases. Using these

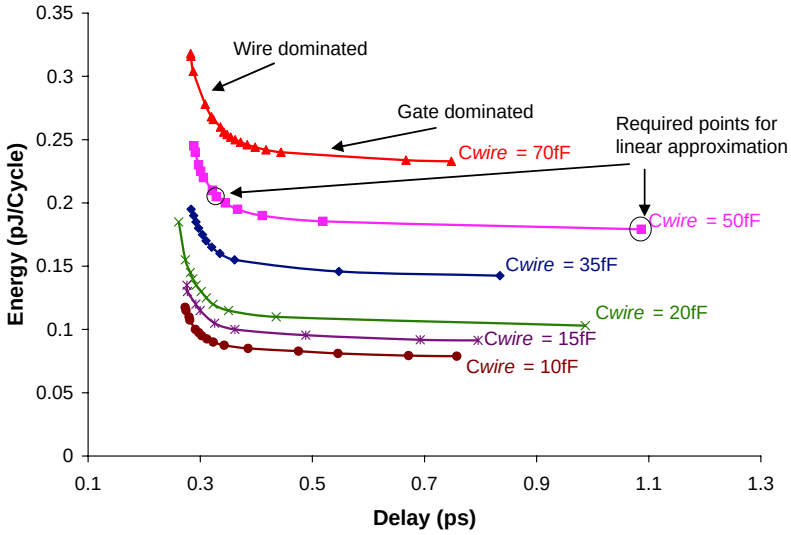


Figure 5.9. Inverter energy versus delay for varying C_{in}

facts and the analysis of the two-stage inverter chain which shows that sensitivity is a strong function of branching factors due to interconnect, the slope is approximated by the following equation:

$$slope = \frac{C_{wire}}{C_{in}} \cdot \frac{C_{gate}}{(C_{gate} + C_{wire})} \tag{5.15}$$

The first part of the model is based on the observation of how the slope varies in relation to changes in C_{wire} and C_{in}. The second term is based on the inverse of the branching factor given in [43], but the values are based on total wire capacitance and total gate capacitance. This is opposed to the product of branching factors. When equalizing stage efforts, the product is used. However, stage efforts will not be equal across gates in the chain. If a general path in a circuit is considered where there may be interconnect at multiple nodes in the path, the path delay equation in (5.11) shows that each node where the wire is located contributes a term to the summation. When the partial derivative of delay is taken with respect to each gate size (i.e. $C_{g_2}, C_{g_3}, \ldots, C_{g_n}$), each partial derivative contributes a wire branching factor term to the sensitivity and hence to the slope of the sensitivity versus C_{gate}/C_{wire} graph. A good estimate of the total contribution is to form

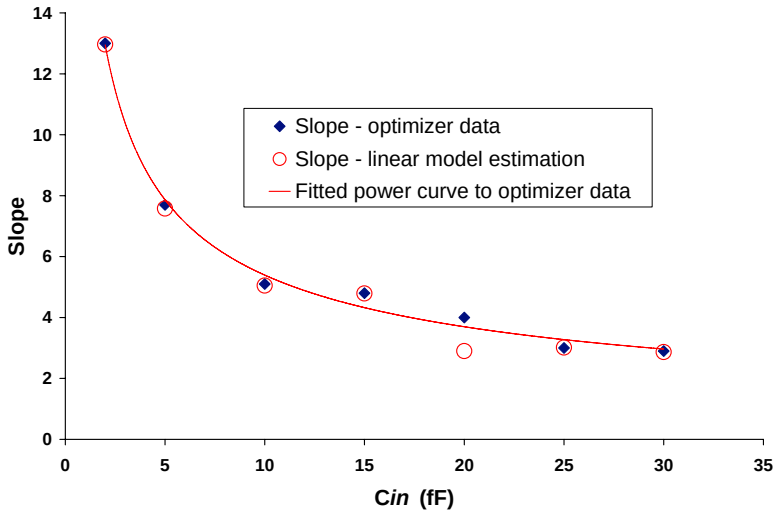


Figure 5.10. Inverter: slope of C_{gate}/C_{wire} versus C_{in}

the ratio $(C_{total\_gate} + C_{total\_wire})/C_{total\_gate}$ along a path. Intuitively, the ratio is accounting for current split among all the interconnect and all the gates along a path. The inverse is used because as C_{gate} increases relative to the fixed wire capacitance, less additional energy is required to compensate for delay across the wire, hence lowering the slope. The solid line in Figure 5.7 plots the model for sensitivity as a linear function of C_{gate}, C_{wire}, and C_{in}. It is not feasible to extract exact values of C_{gate}, the term $C_{gate}/(C_{gate} + C_{wire}$ to form a reasonable model for the slope. Since C_{wire} is fixed, C_{gate} needs be approximated in some manner. The value chosen for C_{gate} will depend on whether the design is inherently gate-limited or wire-limited.

As one can see, there is very good agreement between the optimized points and the model. If the slopes derived from the optimized data are plotted as a function of C_{in}, and the model described in Equation (5.15) is used to estimate the slope, then good agreement is shown in Figure 5.10.

The choice of the branching factor value depends on the relative difference between C_{in}, C_{load}, and C_{wire}. This is because short, medium and long wires must be treated differently. If $(C_{in} + C_{load}) \leq \frac{1}{4} \cdot C_{wire}$, then the minimum energy point is chosen for the C_{gate} value. This for wire-limited designs. If $(C_{in} + C_{load}) > \frac{1}{4} \cdot C_{wire}$ and $(C_{in} + C_{load}) < \frac{1}{2} \cdot C_{wire}$, then

the ED^2 point can be chosen for C_{gate}. This value is appropriate for designs where total wire capacitance is similar in size to total gate capacitance. The ED^2 point was chosen after experimentation with different values obtained from various points on the energy-efficiency curves. The points along the knee of the curve provided the best fit for designs where wire capacitance was comparable to total gate capacitance. If $(C_{in} + C_{load}) \geq \frac{1}{2} \cdot C_{wire}$, the minimum delay point is chosen for the value of C_{gate}. This value is suited to designs that are gate-limited rather than wire-limited.

For designs with small total wire capacitance in relation to total gate capacitance, delay is dominated by gate capacitance. If the minimum delay point is used to approximate the branching factor, $(C_{gate} + C_{wire})/C_{gate}$, then the approximation puts more weight on the total gate capacitance, which closely ties sensitivity to sizing for delay across gates rather than wires. For designs with wire capacitance approximately equal to gate capacitance, delay is equally partitioned across wires and gates. The branching factor is then approximated by a point on the knee of the energy-efficiency curve (e.g. ED^2 or ED^3 or ED^4 points). For designs with large total wire capacitance in relation to total gate capacitance, wire capacitance and resistance contribute significantly to delay. The branching factor is approximated by the minimum energy point because this puts more weight on wire capacitance.

The above rules for choosing the value for C_{gate} which helps determine the slope for the sensitivity model were based on a number of different experiments where wire capacitance, input capacitance, and load capacitance were varied. Figure 5.11 shows the resulting linear model for sensitivity to sizing for each of the input capacitances. The rules given in the previous paragraph were used to generate the curves. It should be noted that since the custom optimizer neglects wire resistance in the optimization, it is also missing from the model given in Equation (5.15). The impact of wire resistance on sensitivity and architecture optimization was detailed in the previous chapter.

The inverter chain is a limited example as it contains no branching or reconvergent paths as is the case with most realistic circuits. Larger blocks must be investigated to ensure that the linear model holds across multiple types of circuits and across layers of hierarchy. Next,

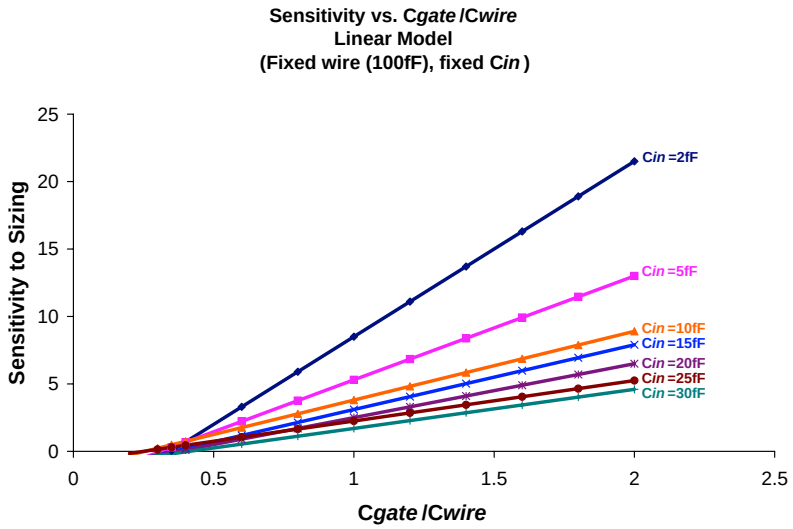


Figure 5.11. Inverter: linear model for sensitivity to sizing

a 64-bit adder is optimized using both a custom circuit optimizer and a synthesis-based optimizer. The adder is then used as a component in an integer execution unit.

5.5 64-bit Ling Adder

The adder study focuses on two levels of hierarchy. The bottom level is a transistor-level implementation of a 64-bit sparse radix-4 Ling carry-look-ahead adder. The sum-precompute path uses a static CMOS implementation whereas the carry-look-ahead tree is implemented using footless domino. The generic carry-select architecture is depicted in Figure 5.12. The higher level of hierarchy is an integer execution unit (IEU) that is built from six 64-bit adders, cache, register files, and muxes. The optimization of the IEU is described in Section 5.6.

The adder gate sizes are optimized for a range of input capacitances for minimum delay under fixed energy constraints using the optimization tool described in [17]. The total wire capacitance is calculated from extracted wire capacitances after layout. The total input and output capacitance for the extracted design were set at 27fF. The energy-efficient curves for the adder are generated for input capacitance varying from 13.5fF to 108fF and the total

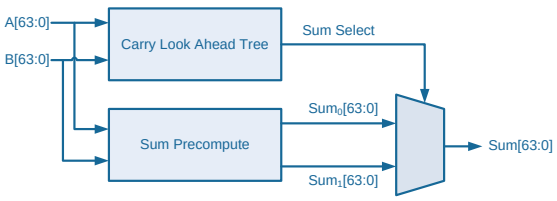


Figure 5.12. Ling Adder

output capacitance, C_{out}, has a simple relationship with C_{in} for all optimizations:

$$\begin{aligned} C_{out} &= 6 \cdot C_{in} + 2 \cdot C_{RF} + C_{cache} + C_{bus} \\ C_{out} &\simeq 9 \cdot C_{in} + +C_{bus} \end{aligned} \tag{5.16}$$

The output capacitance equation is based on the fact the adder is used in an integer execution unit that has six 64-bit adders, two register files, and a cache sitting on a bus, which is similar to the Itanium design published in [71]. The load capacitance of the cache and register files is approximately the same as the load of an adder. Initially, the total wire capacitance was kept at the same value as the extracted C_{wire} for $C_{in} = 27fF$ since it was assumed that for a regular block like an adder, the total wiring capacitance does not vary too much with varying input capacitance relative to total device capacitance. However, it was found that for large input capacitances, the linear relationship broke down due to inaccurate wire estimation. The larger load and input capacitance results in larger area for the adder overall, which results in increased routing overhead. As long as the wiring overhead is appropriately accounted for, the linear relationship holds.

The energy-efficiency curves for the Ling adder are shown in Figure 5.13. The minimum achievable delay for the 64-bit adder is shown to be about 230ps; increasing the input capacitance beyond 54fF results in little change in the minimum achievable delay.

In Figure 5.14, the plot for the 64-bit Ling adder shows a linear relationship between sensitivity and C_{gate}/C_{wire} for all input loading conditions. The minimum energy solution is given by the x-intercept as was the case for the inverter chain example. The slopes are plotted against C_{in} in Figure 5.15. A profile similar to the inverter chain results. The branching factor is estimated using the minimum energy point since the adder design has

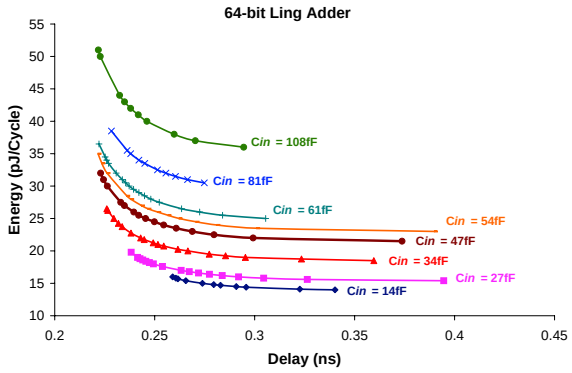


Figure 5.13. Energy-efficiency curves for 64-bit Ling adder

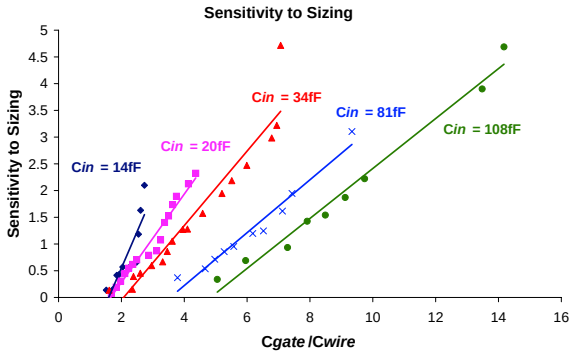


Figure 5.14. Adder sensitivity to sizing versus C_{gate}/C_{wire}

a large output load; that is, $(C_{in\_total} + C_{load}) \leq \frac{1}{4}C_{wire}$. Figure 5.15 shows a very good correspondence between the model for the slope and the measured slopes of each of the sensitivity versus C_{gate}/C_{wire} plots in Figure 5.14.

5.5.1 Synthesis-based Adder Optimization

A static version of a radix-2 full-tree 64-bit Ling Adder is optimized using the custom circuit optimizer and a commercial synthesis tool in the same technology. The comparison is made to ensure that the linear model holds across different types of optimization. Synopsys Design Compiler is used to synthesize the gate-level Verilog model of a static radix-2 full-tree 64-bit Ling adder. A 90nm standard CMOS process with nominal operating conditions

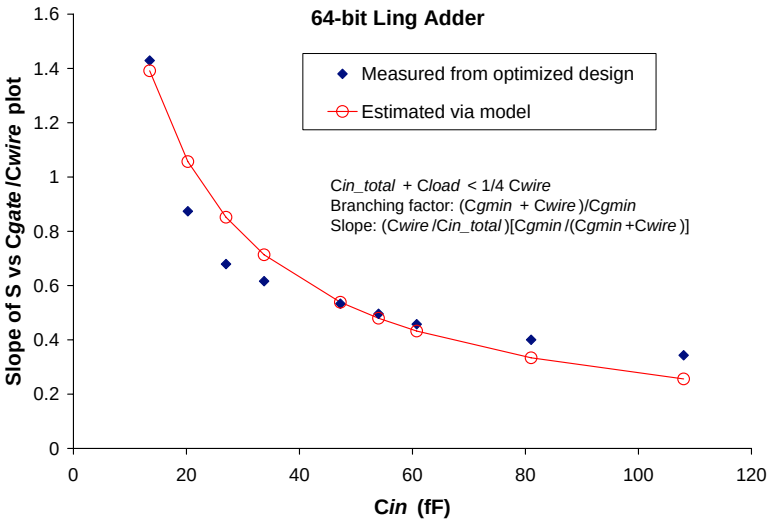


Figure 5.15. 64-bit Ling Adder: slope of C_{gate}/C_{wire} versus C_{in}

was used for mapping the design. Wire load models were created from custom layout, and 50% switching probability was assigned to the inputs. The energy-delay curves for the synthesized version of the adder and the custom circuit optimized version are shown in Figure 5.16.

Sensitivity versus C_{gate}/C_{wire} is plotted using the available estimates of total C_{gate}, C_{wire}, and C_{in} after one pass of place and route. Figure 5.17 shows a significant difference between the synthesized design and the custom design. The plot of sensitivity versus C_{gate}/C_{wire} for $C_{in} = 27fF$ highlights the gap by the difference in slopes. The custom-synthesis gap is due to the fact that synthesized designs require more routing area and gate sizes are quantized. Additionally, the delay constraint dominates in synthesis. Once the delay constraint is met, little effort is spent in optimizing the energy. However, the relationship between sensitivity and C_{gate}/C_{wire} remains linear for a given C_{in}. This linear relationship extends to larger blocks as will be seen for the IEU and digital filters.

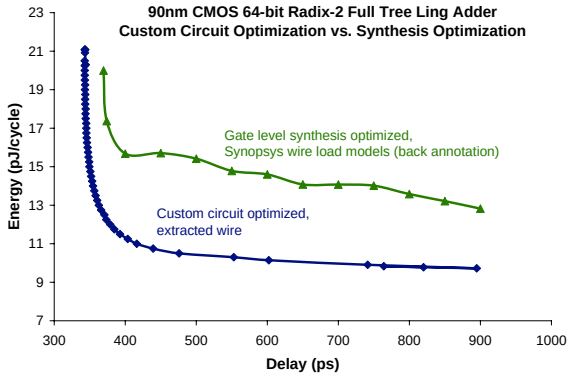


Figure 5.16. Synthesis versus custom optimization: adder energy-efficiency curves

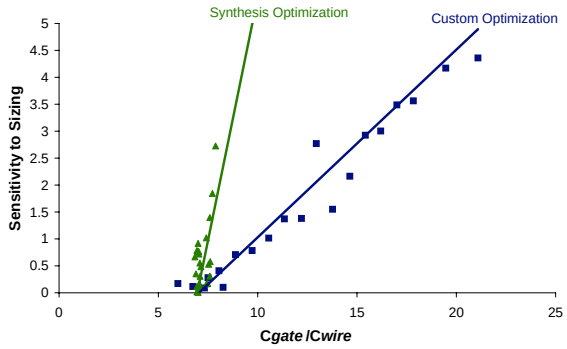


Figure 5.17. Synthesis versus custom optimization: sensitivity to sizing model

5.6 Integer Execution Unit

The adder and inverter chain represent simple lower level leaf cells. In order to show that the linear model for sensitivity to sizing holds for larger blocks, an integer execution unit (IEU) is optimized. The architecture is based on the Itanium-2 design [71] and is illustrated in Figure 5.18. The IEU consists of six ALUs operating in parallel, two register files, a cache, and a loop-back bus. The core of each ALU is the 64-bit Ling adder.

A simple energy and delay model for the IEU is created based on the design information presented in [71].

$$\begin{aligned} D_{IEU} &= 2 \cdot D_{ALU} + D_{BUS} \\ E_{IEU} &= 6 \cdot E_{ALU} + 2 \cdot E_{RF} + E_{CACHE} + E_{BUS} \end{aligned} \tag{5.17}$$

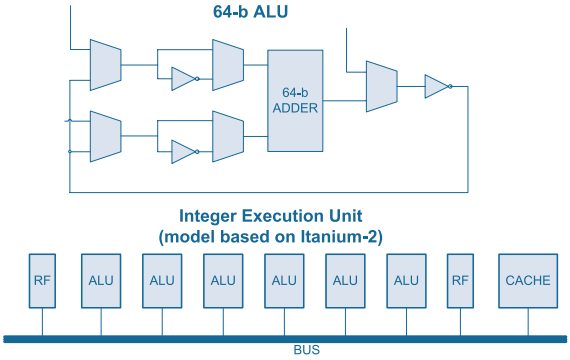


Figure 5.18. Integer execution unit (IEU)

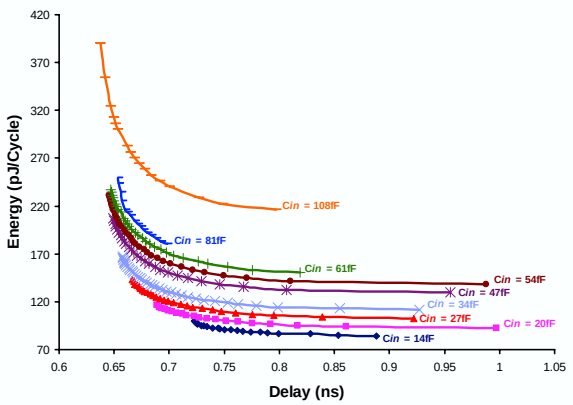


Figure 5.19. IEU energy versus delay

In [71] the authors show that each half clock cycle of the IEU is exactly the delay of one ALU. The amount of energy dissipated in the register files and cache is small compared to the ALU and is lumped together with the energy attributed to the large bus, hence, the model for energy can be simplified to: $E_{IEU} = 6 \cdot E_{ALU} + E_{BUS}$. Based on this simplified model, energy-efficient curves for the same range of input capacitances as used in the adder optimization are generated and plotted. These are shown in Figure 5.19.

In order to gauge the effect of wire scaling, two extreme models are used for calculating total C_{gate}/C_{wire} for the IEU. The first only includes wiring overhead for the six adder blocks and the bus, and neglects all additional wiring overhead associated with the cache

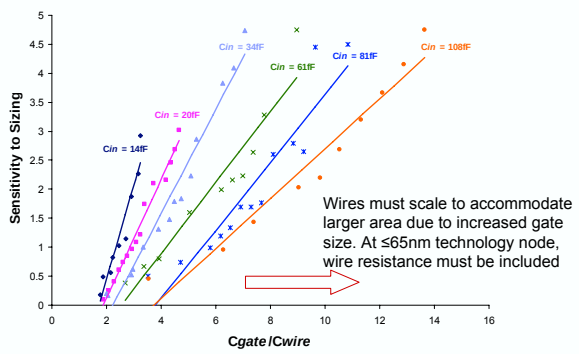


Figure 5.20. IEU sensitivity to sizing versus C_{gate}/C_{wire}

and register files. The second model includes the total wiring overhead of the peripheral circuitry as $1.2C_{wire}$ and is included in the calculation of C_{gate}/C_{wire}.

Figure 5.20 shows sensitivity versus C_{gate}/C_{wire}, where wiring overhead for the peripheral circuitry in the IEU is included in the calculation of the total wire capacitance. The linear relationship continues to hold for larger blocks composed of smaller building blocks such as adders.

When C_{gate}/C_{wire} is plotted against C_{in} for sensitivity of 2 (at the knee of the energy-efficiency), the effect of inaccurate wire estimation is shown in Figure 5.21. The curve becomes polynomial in nature rather than linear. However, when the correct total wire capacitance is used, the slope of the curve closely matches the slope of the curve for the adder

The plot of C_{gate}/C_{wire} versus C_{in} for sensitivity of 2 shown in Figure 5.21 verifies that for an optimum design the sensitivity of both blocks is equal at only one particular C_{gate}/C_{wire} and C_{in}. Figure 5.21 also shows that when wires scale with the design, an increase in C_{in} causes a linear rise in C_{gate}/C_{wire} for both the adder and IEU; when wires remain fixed and do not scale with the design, there is no correlation between the adder and IEU in how C_{gate}/C_{wire} scales with increasing C_{in}. The light blue curve with triangular points is for an IEU model that neglects to take into account overhead of wiring for the register files, cache and multiplexers.

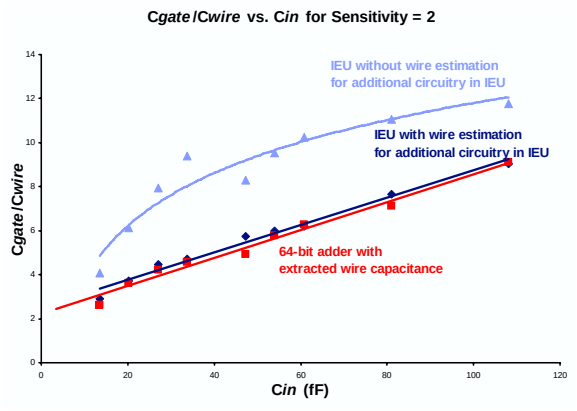


Figure 5.21. Effect of inaccurate wire estimation: C_{gate}/C_{wire} vs. C_{in} for sensitivity of 2

5.7 Finite Impulse Response Filter

In order to further illustrate the linear relationship between sensitivity and C_{gate}/C_{wire}, transpose and transverse 32-tap filters are synthesized using high-level RTL code using the tools and methodology described in Section 5.5.1. The resulting sensitivity curves from the optimized filters using synthesis tools are shown in Figure 5.22. The relationship remains linear, according to the same type of model developed for the IEU, adder, and inverter chain. The x-intercept represents the minimally sized solution (minimum energy design point), and the slope is estimated using Equation 5.15. The branching factor is approximated by the minimum delay point as the filters represent systems which are gate-limited rather than wire-limited due to large number of multipliers, registers, and adders. The transverse filter has less routing overhead than the transpose filter which is shown by the different slopes and x-intercepts.

5.8 Model Limitations

The results in the previous sections highlight a fast approach to estimating sensitivity to sizing without the need to calculate derivatives or generate large numbers of energy-efficiency curves. The model is a first order linear model that can be used at early stages of design. It is useful to accurately model small building blocks of a design such as adders,

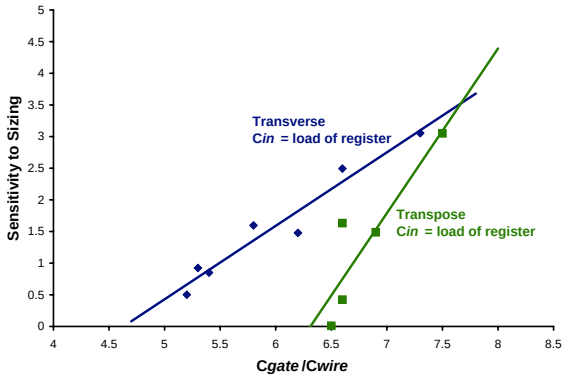


Figure 5.22. FIR sensitivity to sizing versus C_{gate}/C_{wire}

multiplier, registers, and muxes. Once these models are available, then composition rules can be used to generate energy-efficiency curves or sensitivity to sizing for larger systems using the building blocks as components. The composition method of generating system level energy-efficiency curves is described in Chapter 3.

There are some limitations to using a model for sensitivity. First, it is an approximation based on how well the optimizer behaves on smaller blocks. The branching factor used in estimating the slope is also an approximation. Sometimes, it is not clear whether a design will be wire-limited, gate-limited or the total wire capacitance will be on the same order as total gate capacitance. Hence picking an appropriate value for branching factor may be difficult if an initial synthesis with wire estimation has not yet taken place.

5.9 Summary

This chapter built a first-order linear model for sensitivity to sizing using numerical optimization of various types of benchmark circuits. The first step to developing the model was to analytically solve an optimization of a two-stage inverter chain in the presence of interconnect. The results hinted that sensitivity to sizing is characterized by total gate capacitance, total wire capacitance, and input capacitance. The benchmark optimization results for inverter chain and 64-bit adder show that in regions of the energy-delay space where the sensitivity lies below 5, the relationship between sensitivity and device-to-wire capaci-

tance ratio is linear. At higher levels of design abstraction where the adder is a sub-block, the relationship between sensitivity and device-to-wire capacitance ratio remains linear as long as wires scale with devices. If wires do not scale at the same rate as device size (or total gate area), then the relationship becomes increasingly polynomial due to non-optimal gate sizing. The linear model can be exploited to easily quantify the power-performance tradeoff due to sizing in the energy-delay space without having to directly calculate sensitivities from energy-efficient curves. It provides a mechanism to automatically uncover opportunities for improving the energy-efficiency of a digital circuit in synthesis-based design environment. The model is used in the hierarchical power-performance optimization methodology described in Chapter 3.

Chapter 6

Architecture Optimization of Multi-Standard Radio FIR

Data-rate and mobility tradeoffs and different standards like 2G, 3G, Bluetooth, WLAN, GPS and digital video broadcast are leading to multi-mode requirements; and issues relating to co-existence and inter-working of these different technologies must be solved. Furthermore, secure data transfer and encryption are vital...Together, these issues lead to challenging architectural requirements such as reconfigurability and programmability...
– H. Eul (Infineon), ISSCC, February 2006

The accelerated deployment of multi-mode, multi-standard wireless systems is resulting in an exponential increase in algorithmic complexity that is outpacing the scaling benefits of Moore's Law [8]. Multi-mode, multi-standard wireless communication demands extremely high levels of functionality and flexibility which cannot be simply obtained via technology scaling at little or no area or energy cost. It is necessary to design energy-efficient algorithms and architectures that consume the least power at the required performance. For example, a straight-forward implementation of multi-mode operation requires several transmit and receive chains – one for each radio; this is a costly system in terms of energy and area cost. Ideally, the most efficient design is one where a single transceiver chain is shared among multiple modes and multiple standards. Numerous architectures can be conceived which achieve the required throughputs. However, wireless transceivers have tight power

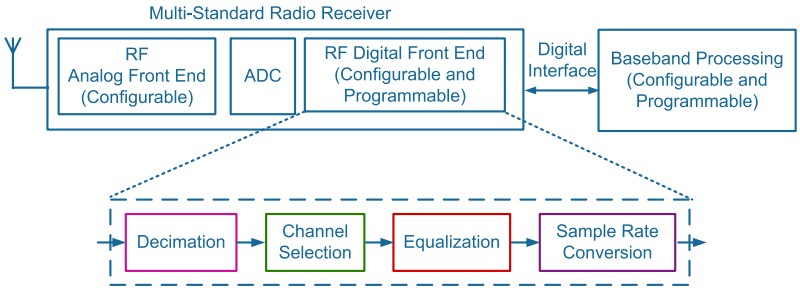


Figure 6.1. Generic RF front-end architecture for multi-standard radio

and area constraints due to battery and size restrictions; additionally, consumers continue to demand increasing amounts of functionality, low-cost, and highly reliable systems. The design tradeoff space is very large and design constraints are numerous; thus significant effort is required to select the optimal architecture which will result in the lowest power consumption for the required performance.

Future multi-mode, multi-standard wireless receivers will continue to move boundary between analog and digital signal processing for increased flexibility [72]. The general receiver architecture consists of analog hardware from the antenna to the analog-to-digital converter (ADC). Since analog blocks are difficult to design for reconfiguration, recent proposals for new architectures [72] recommend that some analog signal processing tasks be relegated to the digital domain for flexibility. Consequently, the requirements for flexibility in the analog front-end are reduced. The generic flexible receiver architecture for multi-standard radio is shown in Figure 6.1. The traditional analog interface between the RF front-end and baseband processing block is now replaced by a digital one.

Various signal processing tasks must now be constructed in the digital domain to support the new digital interface between the RF front-end and the baseband processing. These include: signal detection, channel selection, decimation, sampling rate conversion, and equalization. Finite impulse response (FIR) filters are essential blocks that help implement all of these tasks which are required in the digital front-end of flexible radio receivers (and transmitters). Each of the signal processing tasks and standards require varying number of taps, coefficient and input word lengths, and throughput rates. The flexible filters

STANDARD			FILTER REQUIREMENTS	
	Max. Throughput		No. Taps	Word Length (bits)
3G				
WCDMA-UMTS	16-32 MSample/s	1.92 Mbit/s (5MHz)	8-64	6-8
WLAN				
802.11g	40-80 MSample/s	54 Mbit/s	8-64	10-12
802.11n	40-160 MSample/s	100-200 Mbit/s (40 MHz)	8-64	10-12
DIGITAL VIDEO BROADCAST				
DVB-T/H	20-25 MSample/s	4-30 Mbit/s (5-8 MHz)	32-64	10-12
ATSC Resample Filter	15-25 MSample/s	20 Mbit/s	32-64	10-12

Table 6.1. Flexible Filter Requirements

must consume very little power but support low to high throughput rates, and varying number of bits in the word length of the coefficient and input stream. Each standard dictates different requirements for the FIR filter. Some of these are shown in Table 6.1 [73].

This chapter describes the various filter choices, design tradeoff space, the final design and final implementation of a flexible FIR filter for use in a multi-standard, multi-mode radio receiver. The FIR kernel can be used within any channelizer or synchronization block of a wireless receiver in a baseband processor. The design methodology presented in previous chapters is exemplified in the architecture selection and design of the FIR filter. The content of the chapter references work published in [74] and some material from M. Ler's Master's thesis [75]. The chapter concludes by presenting measured results from the fabricated distributed arithmetic FIR filter.

6.1 Digital Front-End FIR Requirements

The designed filter supports the Advanced Television Systems Committee (ATSC) standard, Digital Video Broadcasting (DVB) standard for both terrestrial (DVB-T) and handheld receivers (DVB-H), 3G cellular networks (WCDMA-UMTS), and wireless local area network (WLAN) standards (IEEE 802.11g/n) as given in Table 6.1. The goal of the design is to limit power consumption to two to four times as much power as a filter dedicated to a single standard, while maintaining flexibility and performance.

The specifications given in Table 6.1 are the result of discussions with researchers at Intel based on the implementation of their flexible radio system [73]. As seen from Table 6.1,

throughput requirements range from 16 MSample/s for WCDMA to 160 MSample/s for 802.11n. The range of input word length varies from 6 bits to 12 bits, and the number of taps vary from 8 to 64 [73]. The number of taps for DVB-T/H depends on the tuner architecture; typically, less than 32-taps are required for standard architectures with SAW (surface acoustic wave) devices, but for non-standard architectures (without SAW devices) that may require increased oversampling, 32 to 64-taps are needed [73]. WLAN systems typically require 32-taps for decimation functions; however, flexibility in the number of taps can facilitate reuse for other tasks such as automatic gain control (AGC) and resampling. AGC requires flexibility as WLAN switches between antenna. The ATSC resample filter for digital video broadcast has slightly different range of throughput requirements [73].

6.2 Design Tradeoff Space

An N-tap finite impulse response digital filter is described by the following equation:

$$y[n] = \sum_{k=0}^{N-1} a_k x[n-k] \tag{6.1}$$

There are numerous architectures available for implementing the filter described in Equation (6.1); however, a systematic methodology is required to discover the most energy-efficient design that supports flexibility. The methodology described in Chapters 2 and 3 is used to uncover and evaluate the digital FIR design space, so that the power-performance optimal FIR architecture is chosen for the given underlying technology.

There are four separate design abstraction layers where power-performance-flexibility tradeoffs must be considered: architecture level, micro-architecture level, logic or arithmetic level, and circuit level. Constraints for each layer must be propagated to the other layers to ensure an optimal tradeoff between power, performance, and area. The cost of flexibility is measured as the additional power and area required to support flexibility in terms of tap programmability, and in terms of programmability of input and coefficient word length. The authors in [76] present a detailed analysis of the various architecture, arithmetic level,

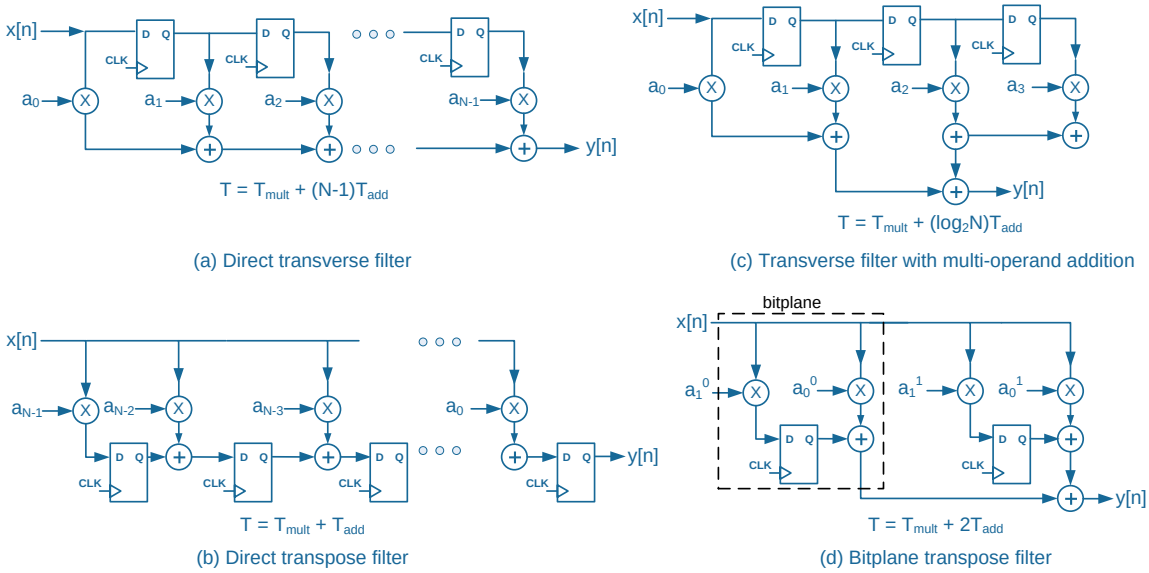


Figure 6.2. A selection of filter architecture examples

and logic level choices available for a conventional filter design. The following subsections highlight the vastness of the filter design tradeoff space.

6.2.1 Architecture Tradeoffs

At the architecture level, a designer can choose to either time-multiplex or fold or parallelize (e.g. multiplex in the space domain) [77]. Time-multiplexing is attractive if the specified throughput constraint is low or medium and power or area constraints are tight. Multiplexing in space is generally used if high-performance is of major concern to the designer. Parallelization of a design is not so straightforward due to the serial nature of the input, so a parallel filter is constructed by splitting the impulse response into multiple phases (i.e. poly-phases) [77, 76].

At this abstraction layer, the designer also has the option of choosing between different filter structures: direct transversal filter, transposed direct form, multi-operand addition where addition forms a tree, or using a distributed arithmetic structure that eliminates multiplication. Each of these structures are shown in Figure 6.2 and Figure 6.4.

Conventional FIR Filters

A one-to-one mapping of Equation (6.1) leads to the direct form shown in Figure 6.2(a). Unfortunately, this form does not yield high performance as the critical path consists of one multiplication followed by N-1 additions. Pipelining a transverse filter can improve performance. A more efficient design is the transpose form shown in Figure 6.2(b). The critical path in this structure is one multiplication followed by one addition. However, this design exhibits an overly large input capacitance due to the multiplications at the input unless buffering is employed. Thon in his ISSCC 1995 presentation described an 8-tap programmable transposed filter for disk-drive read channels; it was fabricated in 0.8μm, 3.7V CMOS and consumed 150mW of power at 240Mb/s throughput [78]. Coefficient word length and tap programmability is supported and the filter incurs a small 3-cycle latency penalty that includes input and output latches [78]. An example of a transpose filter design using space multiplexing is described in [79]; the authors report a 550 Mb/s, 36mW (in a 0.21μm CMOS process) 8-tap transpose filter using Booth-encoded data. The drawback of this approach is increased area due to additional hardware, and increased power consumption.

An interleaved, bit-plane approach shown in Figure 6.2(c) is used to reduce area and power costs and maintain performance. The main idea behind the interleaved approach is to compute and accumulate the partial products associated with the filter coefficients simultaneously, which reduces routing complexity dramatically [77]. If the coefficients are interleaved so that their partial products are computed in different rows, then this leads to a bit-plane architecture as shown in Figure 6.2(c). Folding or multiplexing in time may be applied to a bit-plane architecture that increases speed and enables systematic synthesis. The authors in [80] describe this technique in detail for a transpose direct form filter. Authors in [81] present a five-tap programmable FIR filter that uses two interleaves to achieve high throughput and simplify clock and signal distribution at the physical design level; the cost is increased area. They also employ pipelining, Booth-recoded partial products, and multi-operand addition. The pipelining of the carry-save partial-product summation array adds an additional cycle of latency [81].

Distributed Arithmetic Digital Filters

Conventional filter architectures are based on a straightforward mapping of the FIR filter algorithm to hardware with a multiply-accumulate function as the core processing element. A distributed arithmetic filter eliminates explicit multiplication by reordering and mixing the multiplication terms of the filter [82, 83]. This architecture eliminates the need for multipliers which are costly in terms of computation and power. The output of the N-tap, W-bit input word FIR filter can be represented by Equation (6.2) which can be rewritten as in Equation (6.3).

$$y[n] = \sum_{k=0}^{N-1} a_k \left(-x_{(W-1)}[n-k] + \sum_{j=1}^{W-1} x_{(W-1-j)}[n-k] \cdot 2^{-j} \right) \tag{6.2}$$

$$y[n] = -\sum_{k=0}^{N-1} a_k x_{(W-1)}[n-k] + \sum_{j=1}^{W-1} \left(\sum_{k=0}^{N-1} a_k x_{(W-1-j)}[n-k] \right) \cdot 2^{-j} \tag{6.3}$$

The inner product sums in Equation (6.3), namely $a_k x_{(W-1-i)}[n-k]$ where $i = 0 \ldots (W-1)$, are simply the coefficients of the N-tap filter weighted by the $(W-1-i)$ bit of each $x[n-k]$ input word. Since there can only be 2^N such inner sums for N coefficients, these inner sums can be precomputed and stored in a look-up table (LUT). Equation (6.3) can be rewritten as:

$$y[n] = ((\ldots((0 + A_0) \cdot 2^{-1} + A_1) \cdot 2^{-1} + \ldots + A_{(W-3)}) \cdot 2 - 1 + A_{(W-2)}) \cdot 2^{-1} - A_{(W-1)} \tag{6.4}$$

where $A_{(W-1-i)} = \sum_{k=0}^{N-1} a_k x_{(W-1-i)}[n-k]$.

In a serial implementation, at the i-th clock cycle (where $i = 0 \ldots (W-1)$),the i-th bit of the input signals $x[n-k]$ where $k = 0 \ldots (N-1)$ form an N-bit address which is used to retrieve the inner product sum $A_{(W-1-i)}$. Each inner product sum is accumulated with the previous partial sum and then shifted to the right by one bit as in Equation (6.4). After $(W-1)$ clock cycles, the precomputed inner sum corresponding to the sign bit of the input stream (i.e. the MSB) is subtracted from the partial accumulated sum and shifted to the right by one bit to produce the final output of the N-tap FIR filter. Figure 6.3 shows a pictorial representation of the steps just described.

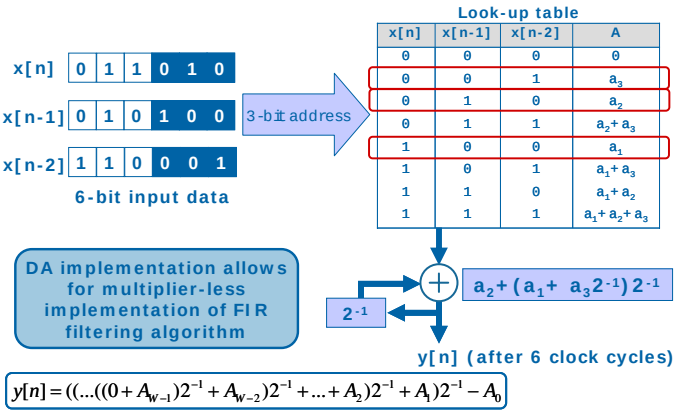


Figure 6.3. 3-tap, 6-bit input word distributed arithmetic FIR example

Figure 6.4 shows a bit serial block level implementation of an N-tap distributed arithmetic filter. A parallel implementation is possible by duplicating the the look-up tables for each bit of the input word stream. For example, if there are 12-bits in the input word stream then there would be 12 replicas of the look-up tables, each indexed by the same N-bit address. The bit-serial implementation takes W clock cycles to produce the final result, whereas in a parallel implementation, a single clock cycle is all that is required. A parallel implementation of a distributed arithmetic FIR is shown in Figure 6.5.

The disadvantage of using the distributed arithmetic FIR filter is that it consumes a lot of memory: memory requirements grow exponentially with increasing number of taps. For example, for a 64-tap filter, the look-up table size required (for processing a single bit) would be 2^{64}! Section 6.2.3 describes how this problem may be mitigated.

Previous implementations of distributed arithmetic FIR filters [83, 84, 85] have focused on using this structure to achieve high performance for low-order filters. For example, in [84], 8-tap and 10-tap filters are implemented in a 0.5μm BiCMOS process which achieves over 200MHz operation at over 300mW. A 10-tap digital FIR in 0.18μm domino CMOS is demonstrated in [83]. This filter performs at 2.3GSample/s consuming 1.2W of power. In a revised design, Tierno et. al. published a variable-latency distributed arithmetic 6-bit, 10-tap distributed arithmetic filter also in 0.18μm domino CMOS which achieved 1.3GSample/s

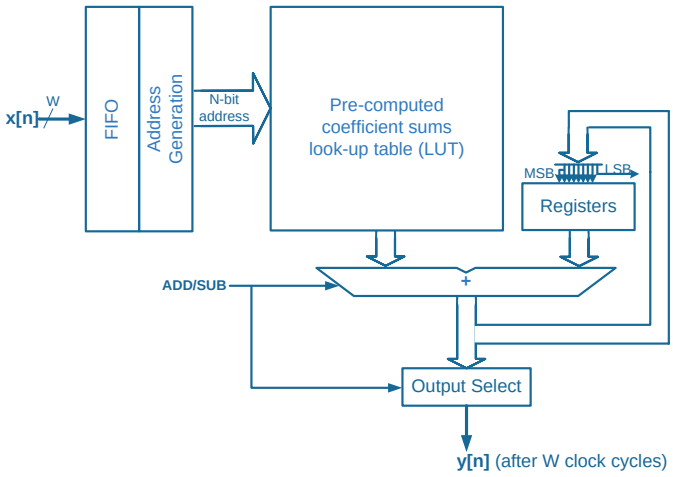


Figure 6.4. Bit serial implementation of distributed arithmetic FIR

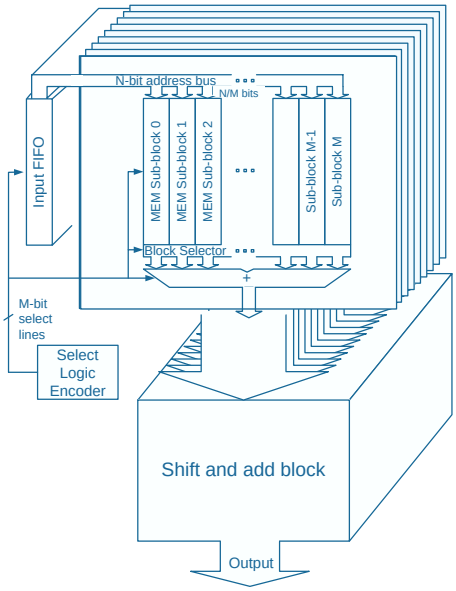


Figure 6.5. Bit parallel implementation of distributed arithmetic FIR

AUTHORS	FILTER TYPE	TECHNOLOGY	THROUGHPUT	POWER
Thon et. al. ISSCC 1995	8-tap transpose programmable in taps and coefficient word length Booth-encoded coefficients	0.8μm, 3.7V CMOS	240Mb/s	150mW
Pearson et. al. JSSC December 1995	6-bit, 8-tap and 10-tap distributed arithmetic	0.5μm BiCMOS 3.7V (8-tap) 3.3V (10-tap)	250MSample/s (8-tap) 270MSample/s (10-tap)	340mW (8-tap) 780mW (10-tap)
Moloney et. al. JSSC July 1998	Programmable interleaved 5-tap transverse with Booth recoded partial products	0.7μm BiCMOS, 5V	200MSample/s	165mW
Staszewski et. al. JSSC August 2000	8-tap parallel transpose with Booth-encoded input data	0.21μm CMOS, 1.8V	550Mb/s	36mW
Rylov et. al. ISSCC 2001	6-bit, 10-tap distributed arithmetic filter with domino logic adders	0.18μm domino CMOS, 1.8V	2.3Gb/s	1.2W
Tierno et. al. ISSCC 2002	6-bit, 10-tap distributed arithmetic filter with dynamic logic datapath with variable latency	0.18μm domino CMOS, 1.8V	1.3Gb/s at 2.1V	450mW
Kim et. al. TVLSI 2003	10-bit, 32-tap distributed arithmetic filter	0.6μm CMOS, 3.3V	20MSample/s	75mW

Table 6.2. Summary of referenced FIR filters

at 2.1V and dissipated 450mW [85]. The filter uses independent precharge and compute signals per domino stage and these signals are controlled by by self-timed control circuits. Latches at the input and output of the datapath resynchronize the data to the clock [85]. In 2003, Kim et. al. demonstrated a 32-tap distributed arithmetic filter in 0.6μm 3.3V CMOS which achieves 20MHz operation and consumes 75mW of power [82].

A summary of the referenced conventional and distributed arithmetic filters are given in Table 6.2. The goal in this thesis is to evaluate the energy-efficiency of using a distributed arithmetic structure compared to conventional FIR architectures for a multi-standard radio receiver using the design methodology described in earlier chapters. The systematic tradeoff analysis clarifies for the designer which filter structure is best suited for use in a multi-mode, multi-standard radio receiver.

6.2.2 Micro-Architecture Tradeoffs

Pipelining and retiming are common techniques used at the micro-architecture level to improve performance or reduce power consumption. Pipelining over N stages for a N-tap filter can attain a speedup of N over sequential processing. This can easily be seen if applied to an N-tap transverse filter implemented in its direct form. The latency of its design is only changed if the critical paths are of unequal length. Pipelining the N-tap transverse filter can result in the same performance in terms of delay as a transpose filter.

Parallel processing is a dual of pipelining: here multiple outputs are computed in a single cycle. For a parallel implementation of a digital filter, the single-input, single-output natural structure must first be converted into a multiple-input, multiple-output structure. Pipelining can be combined with parallelism to further increase the speed of a pipelined architecture [77].

Retiming can be used to change the position of sequential elements without affecting the functionality of the filter. It is a useful technique to reduce the number of registers in a design (and hence power consumption) while maintaining the required performance. Retiming can also be used to improve the performance of the design by reducing the delay along the critical path. A detailed discussion of retiming filters is provided in [77].

Unfolding and folding are transformation techniques that affect performance and power consumption. Unfolding can uncover hidden concurrencies in a design which can help parallelize a design [77]. Folding is used to decrease the number of adders and multipliers, registers, multiplexers, and wires by systematically determining where multiple operations (e.g. additions) can be time-multiplexed into a single functional unit (e.g. pipelined adder).

6.2.3 Logic and Arithmetic Tradeoffs

At the logic and arithmetic level, decisions on number representation, sign processing, and adder and multiplier architectures can affect the number of partial products, the critical path delay, and power dissipation. For example, two's-complement representation of the input helps eliminate the need for multipliers in a distributed arithmetic architecture.

Recoding of input data and/or coefficients can reduce the number of partial products. Typically, Booth recoding is used to reduce area and power dissipation [76, 79, 81]. Booth encoding of coefficients can save power and area only if the reduction in partial products saves more power and area than the increase in cost required to implement the encoding. An example is given in [86]: here it is shown that Booth recoding of coefficients in a modified bitplane structure can reduce area by approximately 20% and lower power dissipation by about 25%. Booth encoding of data allows reduction of area since resources are shared as

shown and implemented in [79]. In this filter, Staszewski et. al. encoded the data so that the coefficient pre-multiplication could be performed off the critical path; this saving in performance outweighed the penalty due to the overhead of coding the input data [79].

In a distributed arithmetic scheme, memory partitioning and offset binary coding can dramatically reduce the amount of memory required [87]. Memory partitioning subdivides the address space into several clusters and maps addresses to different memory banks which can be independently enabled or disabled [87]. For example, for an N-tap filter, 2^N coefficients need to be stored. Instead of generating N-bit addresses, $\frac{N}{M}$-bit addresses can be used if the memory space is partitioned into M clusters. This reduces the memory required to $2^{N/M} \cdot M$ words instead of 2^N words. The benefit of memory partitioning increases with increased filter order.

Memory code compression can be used to further reduce memory requirements for a distributed arithmetic filter. Offset binary coding represents binary numbers as {-1,1} instead of {0,1} and exploits the identity $x = \frac{1}{2}[x - (-x)]$ to reduce the memory requirements by one half [87]. The two's-complement number $-x$ is given as:

$$-x = -\overline{x}_0 + \sum_{i=1}^{W-1} \overline{x}_i \cdot 2^{-i} + 2^{-(W-1)} \tag{6.5}$$

The over-score indicates the complement of a bit. Then,

$$x = \frac{1}{2}\left[-(x_0 - \overline{x}_0) + \sum_{i=1}^{W-1} (x_i - \overline{x}_i) \cdot 2^{-i} - 2^{-(W-1)}\right] \tag{6.6}$$

Letting $X_i = x_i - \overline{x}_i$ where $i \neq 0$ and $X_0 = -(x_0 - \overline{x}_0)$ and allowing possible values of X_i be ± 1, then (6.6) can be rewritten as:

$$x = \frac{1}{2}\left[\sum_{i=0}^{W-1} (X_i \cdot 2^{-i} - 2^{-(W-1)}\right] \tag{6.7}$$

By substituting (6.7) into (6.4), the result is:

$$y[n] = \sum_{i=0}^{W-1} (A_{(W-1-i)} \cdot 2^{-i} + A(0) \cdot 2^{-(W-1)} \tag{6.8}$$

where $A(0) = \sum_{k=0}^{N-1} \frac{a_k}{2}$. The inner product sum term $A_{(W-1-i)}$ in (6.8) now has only $N-1$ possible values thus reducing the memory requirement to 2^{N-1}. There is a small overhead of initializing the accumulation with the $A(0)$ term.

The use of different adders and multipliers can also affect the power-performance of a design. Carry path reduction via carry-select and carry-lookahead architectures can respectively reduce the carry propagation path from $O(n)$ to $O(\sqrt{n})$ and $O(log(n))$ [10]. Deep pipelining of the carry path can yield a $O(1)$ carry propagation delay albeit with higher synchronization overhead. Carry-save architectures in multipliers are utilized to postpone the carry propagation for several additions but needs a final adder stage for merging the sum and carry vectors [10].

6.2.4 Circuit and Technology Tradeoffs

At the circuit level, decisions on circuit implementation style, choice of supply and threshold voltages, clocking scheme, static or dynamic flip-flops that can be either edge-triggered or level-sensitive, and choice of gate sizes can impact the performance and power of a design. A comprehensive study of the impact of these circuit level choices on power-performance tradeoffs in custom circuits and signal processing ASICs is provided in dissertations by Zlatanovici [17] and Marković [12]. The results from these works have been highlighted in earlier chapters.

The choice of technology type also can dictate whether one architecture is superior to another in meeting system requirements. For example, if a technology process is optimized for memory or leakage, then it is quite possible that a memory intensive design is better suited to meet the power and performance specifications. If a process is optimized for something other than leakage, then a memory-intensive design may not be the best implementation choice.

6.3 Flexible Filter Design Exploration

This section presents the architecture exploration of a flexible communication core (FCC) digital front-end (DFE) FIR filter which supports multiple diverse wireless standards such as those given in Table 6.1. This section and the following sections represent work that was performed as part of an internship at Intel Research. The goal of the work

was to create a flexible filter kernel for use in a universal radio digital front-end processor (as outlined in the introduction to this chapter). Composition which was described earlier in Chapter 3 was used to generate energy-delay tradeoff information for each filter architecture rapidly for a number of different technologies and design constraints. The energy reductions gained by choosing the optimal architecture are used to implement flexibility. The power-performance-flexibility optimization is carried out by characterizing each different choice of filter architecture in the energy-delay tradeoff space. The cost of flexibility is measured by comparing fixed architecture area and power with that of flexible architectures.

Three different 90nm CMOS technologies were used in the design space exploration. The first two processes are high performance technologies: high-performance-1 and high-performance-2. The third 90nm CMOS process is optimized for low leakage. A 65nm CMOS process was used to understand the impact of newer technologies on architecture optimization. Section 6.4 focuses mainly on the architecture tradeoff analysis using the two 90nm CMOS high-performance technologies. In these technologies, the optimized filter is based on a conventional filter architecture.

Section 6.5 presents a tradeoff analysis carried out in a process that was optimized for low leakage rather than high performance. The optimal architecture in this technology was found to be a folded, parallel distributed arithmetic design that was eventually taped out in the low-leakage 90nm process. The majority of design space exploration in the low-leakage process was carried out by M. Ler and is described in [75]. The main results are summarized in Section 6.5 for convenience of the reader as they are referred to in Section 6.6 which describes the taped out filter.

The data generated from the comprehensive tradeoff analysis and design of the filter resulted in the development of the design methodology described in earlier chapters.

6.3.1 Design Space Exploration

The primary objective of this study was to manually apply the concepts of the sensitivity-based design methodology described in Chapters 2 and 3 to obtain a power-

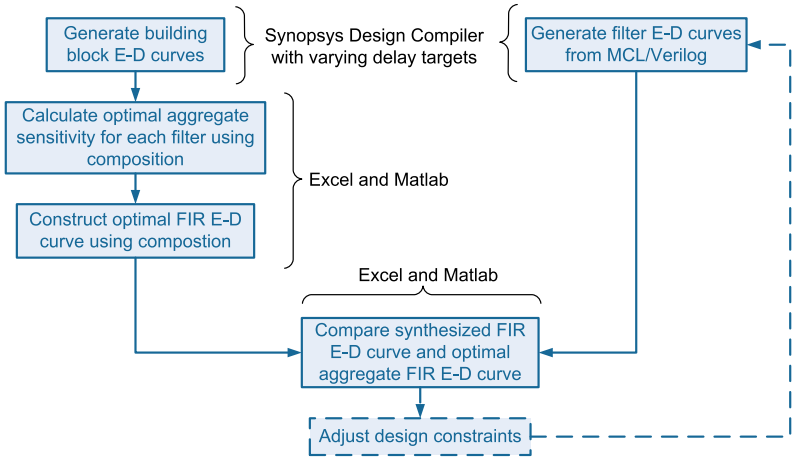


Figure 6.6. FIR architecture tradeoff analysis flow

performance-flexibility tradeoff analysis of a wide variety of architectures. The design methodology supports fast architecture exploration. In a matter of days, it was possible to obtain a wide range of results for many different architectures using multiple technologies.

The method uses energy-delay tradeoff curves to understand the tradeoff between energy and throughput of various design choices. Here, we implemented a bottom-up and top-down hierarchical approach so that a number of different architectures could be explored. Synopsys Design Compiler Tcl scripts were created to synthesize various building blocks such as multipliers and multiply-accumulate blocks for various delay and throughput targets. Composition rules allowed for quick power and throughput analysis of different filter architectures. Optimal aggregate sensitivity was used to construct energy-efficiency curves for the filters. Conventional transpose and transverse filter architectures were coded in Module Compiler and Verilog was generated for each different delay target. The best multiplier and adder architecture choice for the given performance constraints was determined automatically by Design Compiler. Filter architecture throughput and power dissipation was estimated using Design Compiler. Simulink and ModelSim were used later to generate test vectors and activity factors were back-annotated so that Design Compiler could provide more accurate power estimation. Synthesized filter architectures were compared

with the energy-efficiency curves generated from use of composition and optimal aggregate sensitivity calculations. A flow diagram shown in Figure 6.6 outlines the procedure.

Power analysis without accurate activity factors does not invalidate the architecture analysis at early design stages, since each architecture will scale similarly. For example, if the estimate is possibly twice the actual power number for a particular architecture, it will be generally the same for the other architectures. Once an appropriate architecture is chosen that best meets the design constraints, a more accurate analysis can be made. The underlying reason behind this is that at early stages in exploration, only the sensitivities to architecture changes are required. That is, only the relative differences in energy and delay between two different architecture choices are required.

At the micro-architecture level, pipelining, parallelism, and folding were used as design tuning variables. At the logic and arithmetic level, the choice of multiplier and adder type was left to the discretion of the tool. The level of memory partitioning and offset binary encoding was explored thoroughly for the distributed arithmetic architecture. The main circuit tuning variable was gate size. The effect of changing supply voltage or threshold are planned for future work.

6.3.2 Architectures

A wide variety of architectures were evaluated in the energy-delay tradeoff space. These included both types of conventional filters — transpose and transverse — and variations on them such as parallel and pipelined structures. In addition, the distributed arithmetic architecture was also evaluated for similar filter order, and input and coefficient word lengths. The tradeoff analysis is carried out using three different 90nm CMOS technologies: two are high performance CMOS technologies, and the third is a low leakage 90nm CMOS process.

The architectures that are explored are shown in Figure 6.7. As one can see there are numerous combinations of architectures, micro-architecture, and logic and arithmetic modifications. In the figure, OBC refers to offset binary encoding; 1x folded means that the architecture is folded in time once, so that two clock cycles are required to process an

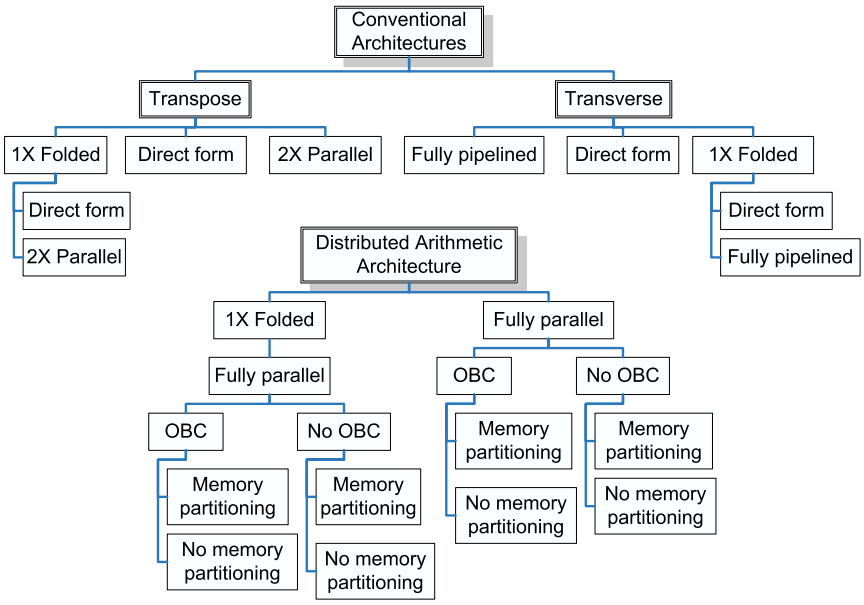


Figure 6.7. Architecture candidates for flexible FIR

entire sample of data; 2X parallel refers to parallelizing the filter so that two samples can be processed in one clock cycle. The fully parallel distributed arithmetic filter refers to processing each input word bit in parallel. The folded, fully parallel distributed arithmetic filter processes half the input word bits in parallel in one clock cycle and the other half in parallel in a second clock cycle.

The next two subsections describe flexible conventional architectures and flexible distributed arithmetic architectures. The distributed arithmetic architecture does not require multipliers.

6.3.3 Flexible Conventional Architectures

Flexibility can be added to conventional architectures by combining parallelism and time-multiplexing, and by adding some control and memory. The general system architecture is shown in Figure 6.8.

The design is time-multiplexed (or folded) to support tap programmability in steps of

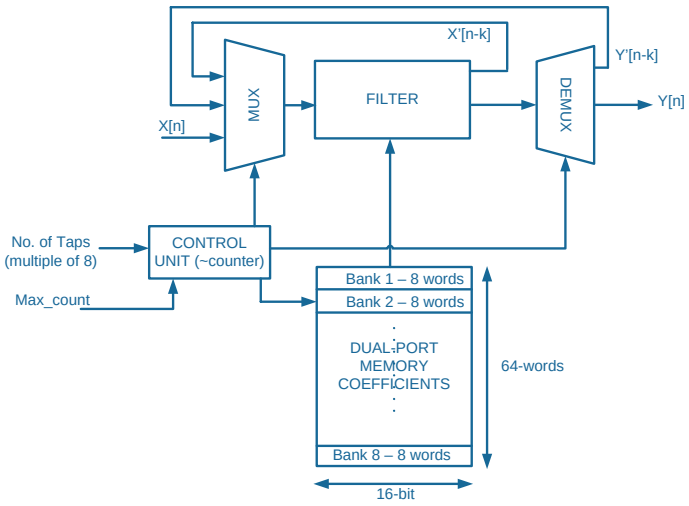


Figure 6.8. Flexible conventional filter

8. The memory stores the coefficient words in blocks of 8 words so that at each cycle, eight taps are processed. A counter is used to load the appropriate block of 8 words at the rising edge of the main clock. The clock can be varied for throughput flexibility, depending on the filter order. In order to support a wide range of throughputs and taps, the 8-tap filter is parallelized so that the odd and even streams are processed in parallel. Thus, an 8-tap filter running at 250MHz can support a maximum throughput of 500MSample/s.

The control block (e.g. counter) runs at a lower frequency than the filter, depending on the filter order. If the filter order is 47, then the control clock would be set to 80MHz to meet a required throughput of 80MSample/s. This requires the filter clock to be set to 250MHz for a parallel, time-multiplexed 8-tap real filter. The multiplexing and demultiplexing operations assist the folding operation. The results of previously processed taps need to be included in the processing of the next set, until the entire set of taps are processed and accumulated to create the final filter output. Any conventional filter can be substituted in the filter block. The basic structure in Figure 6.8 was used to evaluate different types of conventional architectures.

6.3.4 Flexible Distributed Arithmetic Architecture

The multiplier-less distributed arithmetic architecture lends itself well to flexibility due its inherently programmable structure. Programmable filter order can easily be accommodated by partitioning memory. Memory banks that are not required can be easily enabled and disabled. For example if the look-up table of a 64-tap distributed arithmetic filter is partitioned into 8 partitions, then the filter order becomes programmable in steps of 8 taps.

Input word programmability can also be easily added to the distributed arithmetic FIR filter. In a bit-serial implementation, as shown in Figure 6.4, the number of accumulation clock cycles necessary to produce the final filter output can be varied to match the input stream word length. For example, if the number of bits in the input word length is W, then W accumulation clock cycles are required to produce the final output. In the parallel implementation of the filter, as shown in Figure 6.5, each of the W replicas of the LUT containing the partial coefficient product sums can be placed in idle mode or put in sleep mode to accommodate word lengths less than W.

The basic parallel distributed arithmetic architecture used for comparison with the conventional filter architectures is shown in Figure 6.9. The parallel implementation of the distributed arithmetic filter uses an address generation block that takes the input word and generates an N-bit address, where N is the number of taps. The address is passed to an address encoder which uses offset binary encoding to generate a $(N-1)$-bit address which addresses the look-up tables. There is one look-up table per input word bit. After the partial coefficient sum is retrieved from the memory, it is decoded and partially accumulated. The LUT partition adder accumulates each partial coefficient sum from the internal registers of each LUT. The result is then shifted and accumulated to form the final filter output. The select signal selects current block and partition, based on the number of bits in the input word and the number of taps.

A folded version of the fully parallel filter is shown in Figure 6.10. The filter in Figure 6.10 is folded in time so only half the number of look-up tables are required. The

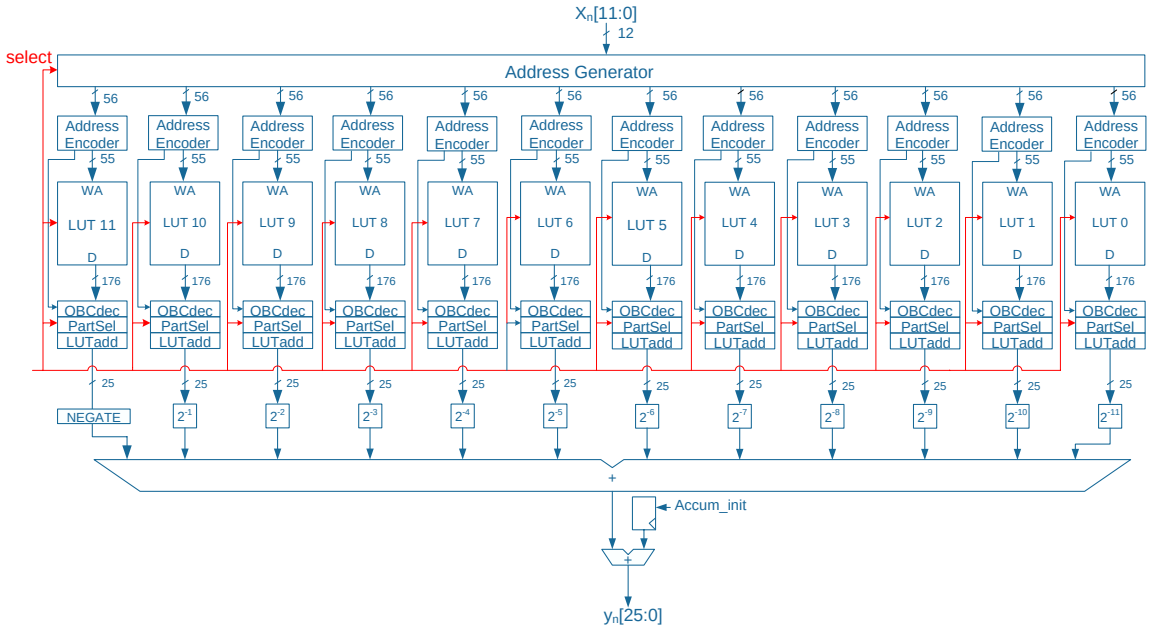


Figure 6.9. Implementation of a parallel distributed arithmetic FIR

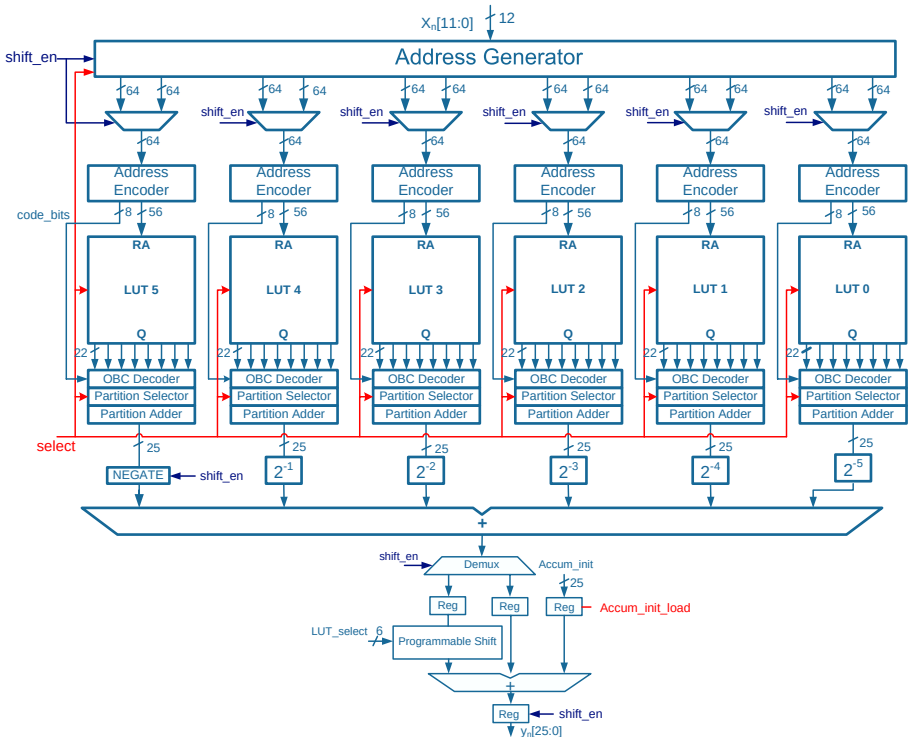


Figure 6.10. Implementation of a parallel folded distributed arithmetic FIR

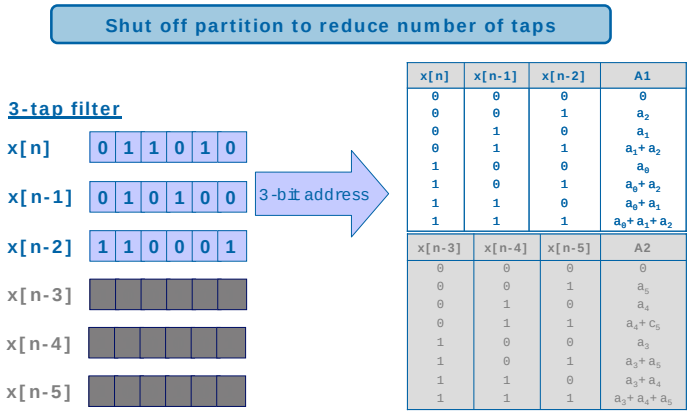


Figure 6.11. Tap programmability in a distributed arithmetic FIR

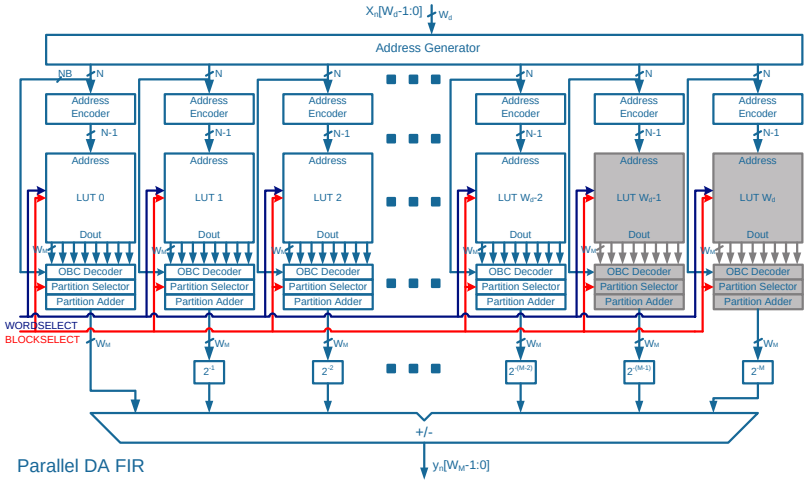


Figure 6.12. Input word programmability in a distributed arithmetic FIR

shift_en signal is used to select either the six MSB bits of the input word or the six LSB bits. The shift_en signal runs at half the frequency of the system clock.

Figures 6.11 and 6.12 illustrate how tap programmability and input word programmability may be implemented using the architecture described in Figure 6.9. Tap programmability is implemented by simply turning off memory partitions in each look-up table. Input word programmability is achieved by turning off entire look-up tables in a parallel implementation. In a serial implementation, the number of clock cycles can be varied to match the number of bits in the input word.

FILTER PARAMETER	RECONFIGURABLE FIR SPECIFICATION
System clock	80MHz to 250MHz
Number of taps	8 to 48
Input data length	12 bits
Input format	2's complement
Output data length	30 bits
Coefficient memory	6 sets of 8 coefficients
Coefficient length	16 bits

Table 6.3. Reconfigurable filter requirements

Coefficient word length programmability can be accommodated by slicing each LUT partition into separate memory banks with smaller word size. Each memory bank is selected using a K-bit word (where K is the number of banks) select signal along with the block select signal that together choose the appropriate memory bank in the given partition. The reader is referred to [75] for further details.

6.4 Flexible Digital Filters – High Performance Technologies

This section focuses mainly on the architecture tradeoff analysis using 90nm processes that were optimized for high performance rather than reduced leakage current. In these technologies, the optimized filter is based on a conventional filter architecture. The optimized filter supports the requirements listed in Table 6.3.

The goal is to create an architecture that is scalable to support larger number of taps. This filter can be used for either baseband processing or in the digital front-end of a multi-standard radio transceiver as described in the introduction of this chapter. Conventional architectures are compared with a parallel distributed arithmetic filter with memory partitioning and offset binary encoding.

6.4.1 Results

Results of the architecture tradeoff analysis are presented here for each of the various architectures considered. The results from the study show that flexibility requires a distributed arithmetic filter or a hybrid parallel-time multiplexed conventional filter, de-

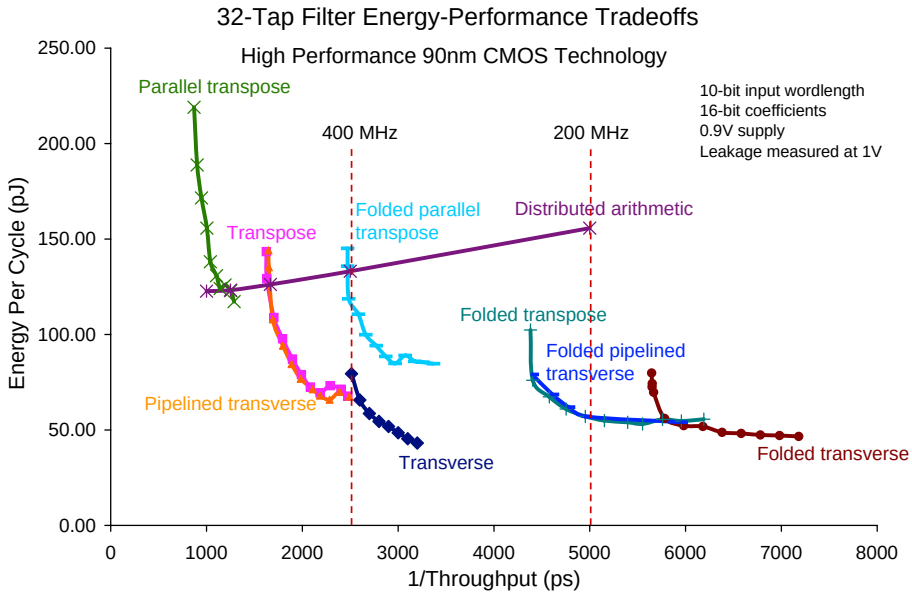


Figure 6.13. 32-tap filter architecture tradeoffs (high performance 90nm CMOS process)

pending on the type of technology used and throughput requirements. The distributed arithmetic filter is preferred for high throughput applications and flexibility in terms of tap programmability, variable input and coefficient word length.

First High-performance 90nm Technology Results

For a 32-tap filter with 4-tap granularity at 400MHz clock, the distributed arithmetic architecture energy efficiency is 1202 million operations per milliwatt (MOPS/mW) with a total area of 309.7K gates. The 32-tap distributed arithmetic architecture is compared with conventional architectures in Figure 6.13. Figure 6.13 shows that parallelism provides high throughput (see parallel transpose curve) and that folding in time provides low energy (see folded transverse plot). As one can see from Figures 6.13, 6.14, and 6.15, at lower throughputs a flexible conventional architecture, such as the transpose or transverse, outperforms the distributed arithmetic FIR. However, at very high throughputs, the distributed arithmetic FIR is the most energy-efficient. Area can be reduced for the distributed arithmetic architecture by folding it in time. It is possible to do this and still meet throughput specifications because the filter can operate at such high throughput rates at low power. The cost

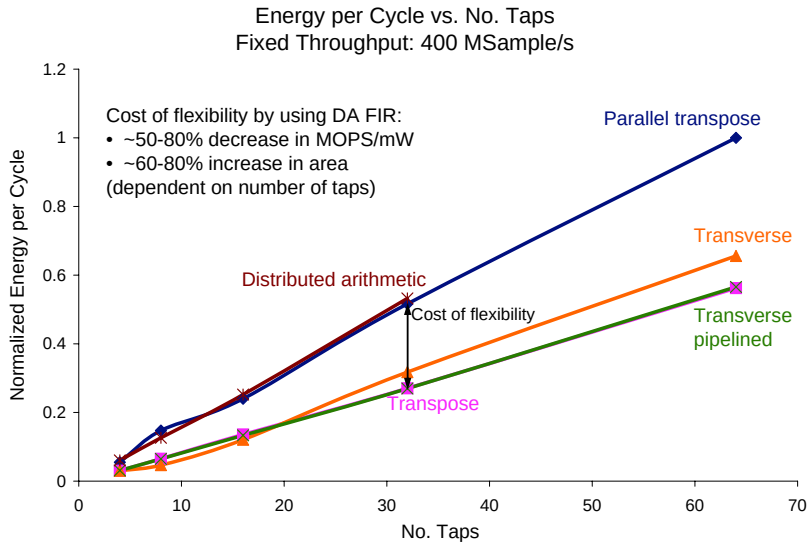


Figure 6.14. Cost of flexibility – energy

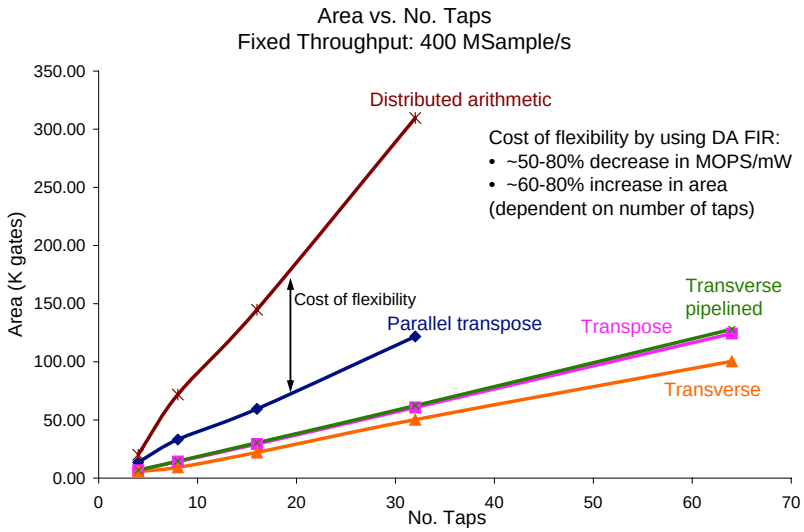


Figure 6.15. Cost of flexibility – area

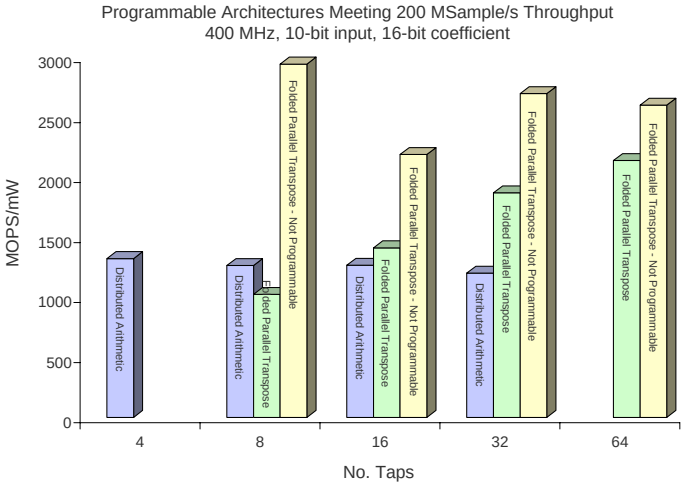


Figure 6.16. Relative cost of programmability – energy

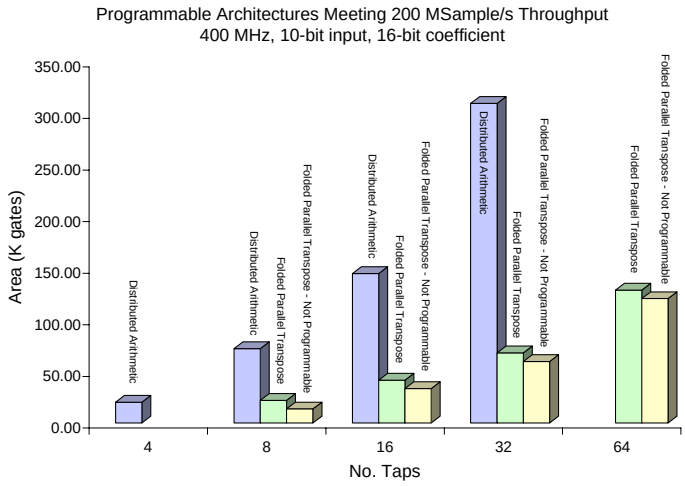


Figure 6.17. Relative cost of programmability – area

of the flexibility provided by the distributed arithmetic architecture is not overly large over the fixed conventional style architectures: approximately an average 50% decrease in energy efficiency and 60-80% increase in area, depending on the number of taps. This is within the goals of the project where a maximum overhead of 2 to 4 times for a programmable architecture is targeted.

By adding limited flexibility (i.e. tap programmability only) to the conventional style architecture as shown in Figure 6.8, the cost ranges from 65% to 18% decrease in energy efficiency depending on the number of taps and type of filter. There is a 6% to 38% increase in area depending on number of taps and type of filter. Figures 6.16 and 6.17 show the relative cost of flexibility over a fixed non-programmable architecture for both a distributed arithmetic architecture and folded parallel transpose design.

Second High-Performance 90nm Technology Results

A similar but more limited study was carried out for a filter supporting tap programmability and a full-band/half-band mode as given in Table 6.3. Unfortunately, in this second high-performance technology it was found that the conventional direct form filters could not meet the timing requirements for a 400MHz clock as in the first high-performance 90nm technology. The maximum clock rate for this technology for the direct form transverse filters was found to be 250MHz for 16-tap filter and lower for higher order filters; and for the transpose filters it was 375MHz for a 4-tap filter and slightly lower for higher order filters. Even though the multiply-accumulate component of the filter was able to meet timing at a 400MHz clock, the addition of wiring overhead in construction of the filters did not allow for a 400MHz clock. Figure 6.18 shows these tradeoffs.

Figure 6.19(a) and (b) show that in the second high-performance technology, the best choice for an 8-tap filter in the programmable tap paradigm is the direct form transverse filter in terms of area and power. However, a parallel version is necessary to support large throughput rates for filters of high order. Unfortunately, in this technology, it is not possible to support filter orders above 47 without incurring a latency penalty. The largest

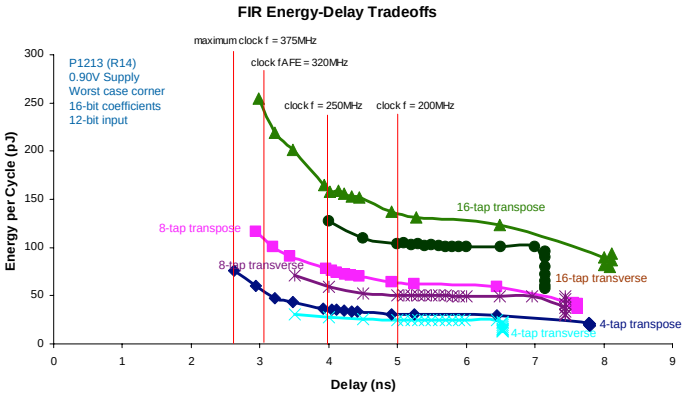
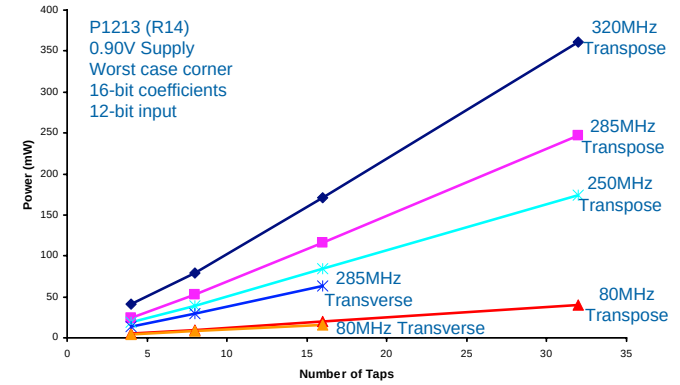
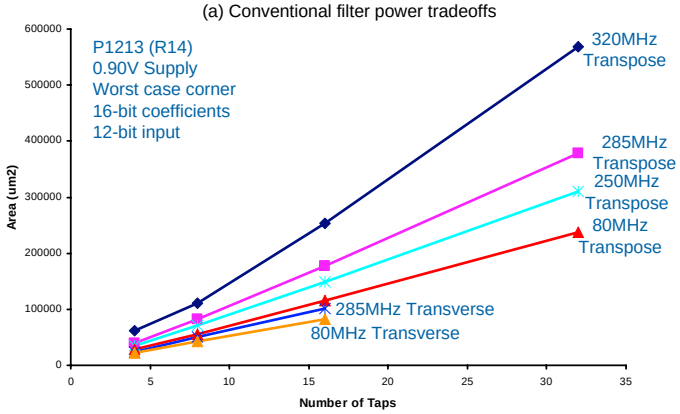


Figure 6.18. Filter energy-delay tradeoffs in second high-performance 90nm technology



(a) Conventional filter power tradeoffs



(b) Conventional filter area tradeoffs

Figure 6.19. Filter energy-delay tradeoffs in second high-performance 90nm technology

250MHz CLOCK	HALF-BAND MODE	FULL-BAND MODE
8-TAPS		
Throughput (MS/s)	500	500
Power (mW)	30	60
16-TAPS		
Throughput (MS/s)	250	250
Power (mW)	60	120
24-TAPS		
Throughput (MS/s)	166	166
Power (mW)	90	180
32-TAPS		
Throughput (MS/s)	125	125
Power (mW)	120	240
40-TAPS		
Throughput (MS/s)	100	100
Power (mW)	150	300
48-TAPS		
Throughput (MS/s)	83	83
Power (mW)	180	360

Table 6.4. Tap programmable filter summary at 250MHz in second high-performance 90nm CMOS

throughput rate supported for a 48-tap filter is 83 MSample/s. Table 6.4 gives a summary of estimated power and throughput at 250MHz.

In either technology, if a more flexible approach is desired with variable input and coefficient word length, then it is recommended that the distributed arithmetic architecture be implemented as it supports much higher throughput rates and increased flexibility without cost in throughput or latency.

Memories and technology comparison

It was found that memory leakage power was high for the high performance technology compared to the low leakage CMOS technologies. The memories in the high performance 90nm CMOS technology required 2.8 times more power than the ones available in the low leakage 90nm CMOS process.

A quick analysis of the 65nm process with the 90nm process was also carried out and showed that 65nm performance is approximately 12% better than 90nm; and 65nm energy per cycle is approximately 65% better than 90nm. These results were obtained for a 32-tap direct transverse filter synthesized in both technologies. The energy savings in

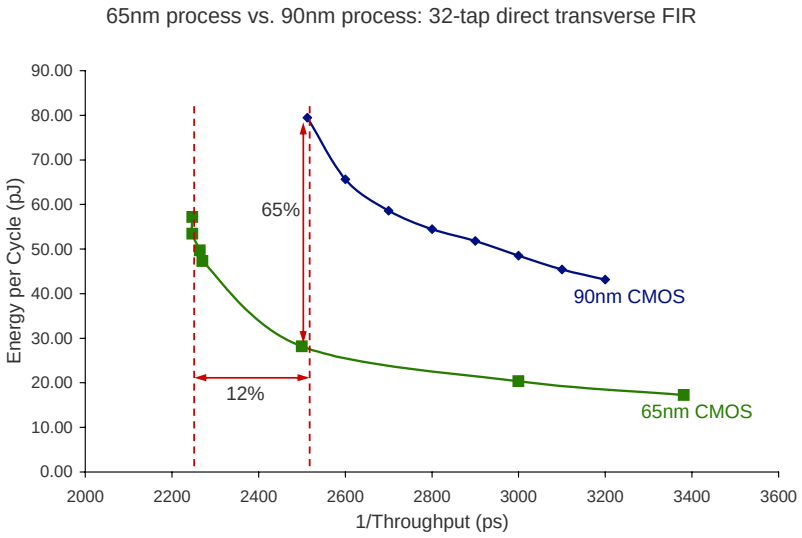


Figure 6.20. Comparison of 65nm process with 90nm process

the 65nm design resulted mainly from a significant decrease in area, approximately 60%. Figure 6.20 shows the comparison between the two technologies for a 32-tap transverse filter implementation. All conventional filter implementations scaled in the same manner.

6.4.2 Sensitivity Analysis

Section 3.2.1 in Chapter 3 described the composition procedure for a 32-tap transpose filter in detail. Composition rules are derived from optimal aggregate sensitivity calculations as given in Table 3.1. These are applied to each architecture explored to construct the energy-efficiency boundary as shown in Figure 6.13. The calculation of the extreme optimal aggregate sensitivity points (minimum delay and minimum energy) are difficult since sensitivities are saturated at either infinity or zero. These points for each filter architecture are approximated using the minimum delay and energy points obtained for each component block, and the architecture models for energy and delay.

In some cases, only a single point on the knee of the curve is necessary. This is seen in the composition of the multiply-accumulate block in Figure 3.6. The register energy-efficiency point is effectively a single point, at the knee of the curve (sensitivity is 1.03).

The relative contribution to total delay and energy of this block compared to the adder and multiplier is very small. Similarly, the add block has a smaller contribution to total system energy and delay as compared to the multiply block. In the case of the add block, only three points are relevant: the knee (sensitivity is 2.53), the minimum delay point, and the minimum energy point.

The MAC example clearly shows that balancing sensitivity is difficult since three different values for optimal aggregate sensitivity were obtained for the MAC (see Chapter 3, Section 3.2.1 for details). However, they were balanced within a 6% threshold of one another. As mentioned earlier, artifacts of synthesis cause these discrepancies in optimal aggregate sensitivity calculations.

At the micro-architecture level, the composition rules given in Table 3.1 are applied to the direct transpose and direct transverse architectures to yield energy-efficiency curves for parallel transpose, pipelined transverse, and their folded versions. At the logic and arithmetic level, the synthesis tool was given a free-hand to choose the appropriate adder and multiplier architecture for the given throughput constraint. At the circuit level, only gate sizing was used as the tuning variable.

The plot in Figure 6.13 represents the energy-efficiency tradeoffs for a set of different filter architectures given a wide range of throughput constraints. At 400MSample/s, a subset of architectures are plotted using the energy-delay tradeoff information on an energy versus taps plot (see Figure 6.14). This plot shows the relative cost of using the distributed arithmetic filter for flexibility over conventional filter architectures. The sensitivity of energy to number of taps is different for each architecture and is given by the slope of the graph. The flexible distributed arithmetic architecture slope is the same as the fixed parallel transpose architecture slope.

In every technology, each standard cell library is optimized for different applications. The first high performance library resulted in very good performance at the expense of leakage. This library was targeted towards high performance RF mobile systems. The second high performance library was targeted to general purpose digital functions that do

not require the same kind of performance as those blocks in RF mobile systems. The first library included hand-crafted standard cells in a newer 90nm technology with updated design rules whereas all the cells in the second library were generated from automated tools and used an older version of the 90nm process. For example, the sensitivity of delay and energy of a two-input NAND gate from a standard cell library may be used to characterize the sensitivity of using one library over another.

The sensitivity of process and standard cell parameters to delay and energy impacts the choice of architecture as seen in the results from the energy-delay tradeoffs for the second high performance 90nm technology. If a library is optimized for low leakage rather than high performance, then the sensitivity of memory parameters to total system energy and delay can be used to gauge the efficiency of one technology over another for a given architecture. Some examples of valid technology parameters that can be used to characterize SRAM cells can be found in [88]. These include write line margin, writeability current, and bit-line current at the '0' storage node during a read. The authors in [88] propose a measurement methodology that characterize large SRAM arrays using the various characterized parameters; the methodology is demonstrated in a 45nm technology.

6.4.3 Optimized Filter Description

Based on the above analysis an 8–48 tap programmable FIR was designed in the high performance 90nm CMOS technology with the following features.

- Operation in half-band mode and full-band mode
- Programmability in number of taps: 8, 16, 24, 32, 40, 48
- Maximum 12-bit input
- Maximum 16-bit coefficients
- 30-bit output
- Sleep mode supported

SIGNAL NAME	DIRECTION	DESCRIPTION
Clk_DFE	IN	250MHz filter clock
X[11:0]	IN	Filter input 12-bit word
Y[30:0]	OUT	Filter output 30-bit word
C[15:0][0:7]	IN	8 16-bit filter coefficients
SELECT1	IN	Demux select to generate two parallel odd and even streams
SELECT2	IN	Mux select to interleave the parallel streams into a single stream
FIR_half_band_mode	IN	Half-band filter mode select
FIR_sleep_mode	IN	Sleep mode select

Table 6.5. DFE FIR input/output ports

This filter supports both full-band and half-band mode operation. Folding in time was used to support tap programmability. It uses a parallel transverse architecture to meet the throughput constraints. It is programmable in steps of 8 taps. The filter clock ranges from 80MHz to 250MHz depending on the desired throughput. The input word length and coefficient word length remain fixed at the maximum number of bits required for all supported standards.

The two main modules of this filter are the actual filter block and the control unit. There are two versions of each. The first version of the design supports an interleaved filter that computes the odd and even paths in parallel, allowing the filter to support up to 500 MSample/s for an 8-tap filter. Initially it was thought that clock gating the odd path would provide an efficient and elegant implementation of a programmable half-band filter however, this design did not support a functional half-band filter as the center odd coefficient was clock-gated off. However, this filter is fully functional in full-band mode. An alternate architecture was used to mitigate the half-band problem in the first architecture. Both of these filter architectures and their corresponding control blocks are described in the next few subsections.

Filter Input/Output Ports

Table 6.5 shows the input/output ports for both flexible filter designs in the second high-performance 90nm technology. For the second design that does not zero out the center coefficient, additional control signals are generated to select or deselect a constant 0 by 0 multiply.

Module Clocking

The filter can be clocked at a maximum clock rate of 250MHz. This allows 8 taps to be processed, dissipating 30mW which includes the overhead of the mux, demux, interleaver, deinterleaver, memory and control unit. A parallel version results in approximately doubling the power to 60mW to support a high throughput rate.

A secondary clock is required for time multiplexing which is dependent on the desired throughput for each standard. If a throughput of 80MSample/s is required then an 80 MHz clock is required which would support a filter of maximum of 48-taps.

Clock Gating

In the first filter design that includes clock gating, the entire odd stream can be clock gated as two paths for the system clock are generated, one for the odd path and one for the even path. This saves power.

Timing

At each rising edge of the 250MHz clock, the counter counts up and a new block of 8 coefficients are loaded into memory so that the filter can proceed to process the input stream. At the rising edge of the secondary clock, 80MHz, the output of the filter is generated as the selects on the multiplexer and demultiplexer are toggled.

Parallel Interleaved Filter with Clock Gating

The first version of the filter features a parallel 8-tap direct form transverse filter that supports up to 48 taps. The block diagram is shown in Figure 6.21. The odd and even paths are processed in parallel as shown in Figure 6.22. The half-band mode is used to clock-gate the odd path. However, this also results in the center odd coefficient being gated off as well which is an unfortunate side-effect of this design. This is remedied in the design

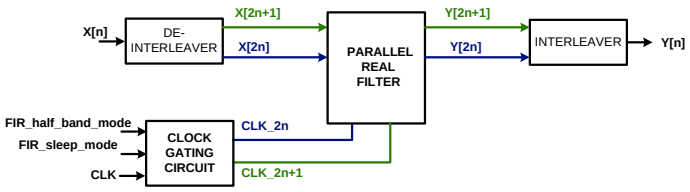


Figure 6.21. Block diagram of parallel interleaved filter with clock gating

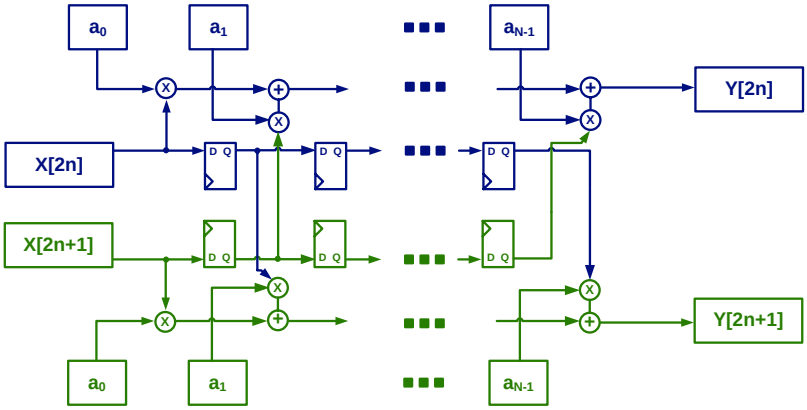


Figure 6.22. Parallel interleaved filter with clock gating

without clock gating. The estimated power dissipation of the design is annotated on the block diagram in Figure 6.23.

The control block is a simple counter that counts based on the number of blocks of 8 coefficients required. For example a 48-tap filter requires 6 blocks of 8 coefficients. The counter is reset on each rising edge of the outer slower clock (e.g. 80MHz). If the sleep mode is desired then the clocks for the odd and even paths are gated to turn off the filter. If the half-band mode is desired and the sleep mode is not on, then the clock for the odd path is gated to save power.

Parallel Interleaved Filter without Clock Gating

A parallel interleaved filter without clock gating similar to the one used above is created to allow for the center odd tap in a half-band to remain active during filtering. This is done

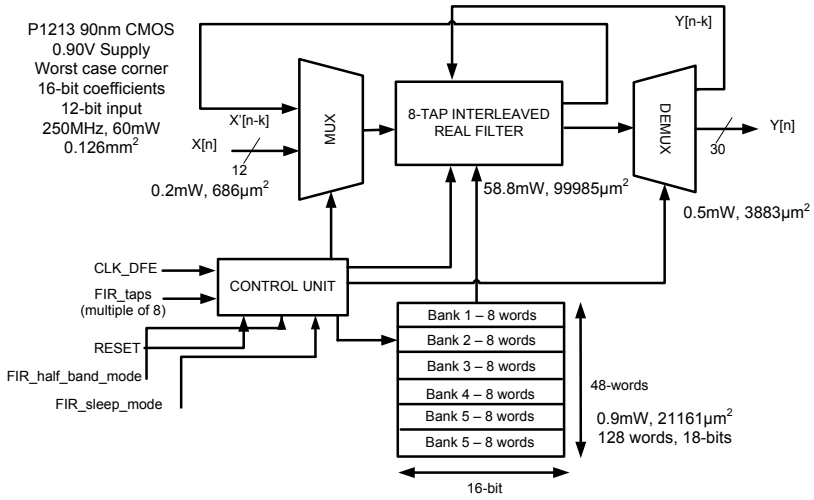


Figure 6.23. Distribution of power and area cost for flexible filter

by using multiplexers to turn on and off the required multipliers as depicted in Figure 6.24. The only change from the previous design is that instead of using clock gating, we are using multiplexers to turn on and off multipliers depending on the operation mode of the filter. The rest remains the same except that the control block has now to generate each of the select signals for the multiplexers.

Figure 6.24 shows the changes to the filter required to support the selection of multipliers depending on whether the coefficients are zero or otherwise. The mselect[i] signals are generated by the control block depending on whether the loaded coefficient is zero. If the coefficient is zero then the mselect[i] is 0 and the multiplexer selects the constant zero to use in the multiply rather than the input data. Otherwise the multiplier operates using the input data.

The power and area numbers are as follows for an 8-tap non-parallelized version of this direct form transverse filter with multiplexers at each odd numbered multiplier:

- 250MHz: area is $0.13mm^2$, power is 30mW.
- 80MHz: area is $0.13mm^2$, power is 8.4mW

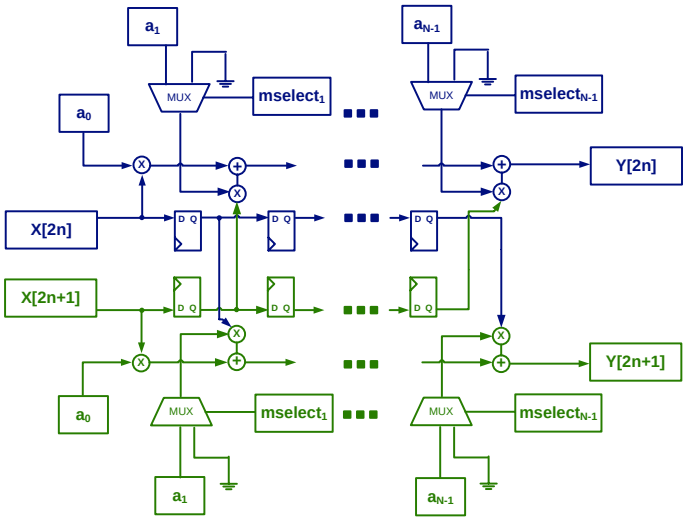


Figure 6.24. Distribution of power and area cost for flexible filter

The power dissipation of this filter is only slightly higher than the direct form transverse filter without the additional multiplexers (i.e. 29.4mW at 250MHz clock).

This control block is slightly different from the one with clock gating. The control block for this version of the filter is more complicated because it must generate the mselect[i] signals for each odd coefficient based on whether the coefficient is zero or non-zero. This is not an overly complicated task as it can be done when the coefficients are loaded into memory. A simple comparison of the coefficient with zero is made and the mselect[i] signal is generated accordingly at each rising edge of the 250MHz clock.

6.4.4 Cost of Flexibility

An estimate of the cost of flexibility was measured by synthesizing a dedicated half-band WLAN 27-tap filter and comparing it to a programmable half-band filter supporting the same number of taps and coefficients. The half-band dedicated filter was first designed and simulated in Simulink and then Verilog code was generated and synthesized to the second 90nm high-performance technology for 250MHz and 80MHz clock. This was compared to the power and area estimates for the synthesized programmable half-band parallel filter without clock gating.

Dedicated Half-Band WLAN 27-Tap Filter

The dedicated half-band filter only requires 8 multipliers which significantly reduces power compared to a flexible approach. At 250MHz, the filter dissipates a total of 21.6mW and requires $0.11mm^2$ area. At 80MHz, the filter dissipates 5.7mW and requires $0.10mm^2$ area.

Programmable Half-Band WLAN 27-Tap Filter

The programmable half-band filter supports 27-taps using 32-tap filter. A 32-tap filter was used because the filter is only programmable in steps of 8; 27-taps are implemented by using zero coefficients for the last five taps. With more than half of its multipliers turned off, the filter consumes approximately 23mW resulting in a throughput of 120MSample/s. At 80 MSample/s throughput, the power is reduced to approximately 15.3mW, giving the estimated cost of flexibility to be three times that of the dedicated filter which is within the required two to four times target. The area penalty for flexibility is a little under two times.

6.5 Flexible Digital Filters – Low Leakage Technology

This section presents the tradeoff analysis carried out in a 90nm CMOS process that was optimized for low leakage rather than high performance. The optimal architecture in this technology was found to be a folded, parallel distributed arithmetic design that was eventually taped out in the low-leakage process. The block diagram of the architecture was shown earlier in Figure 6.10. The work done in this process is described in detail by M. Ler in her Master's thesis [75]. It is summarized here.

Figure 6.25 shows the resulting architecture tradeoff space for a 32-tap FIR filter for some of the evaluated architectures. In a low leakage process technology, the distributed arithmetic filter is the best choice for filter architecture. In the high performance process, a hybrid parallel, folded conventional filter is the most energy-efficient choice.

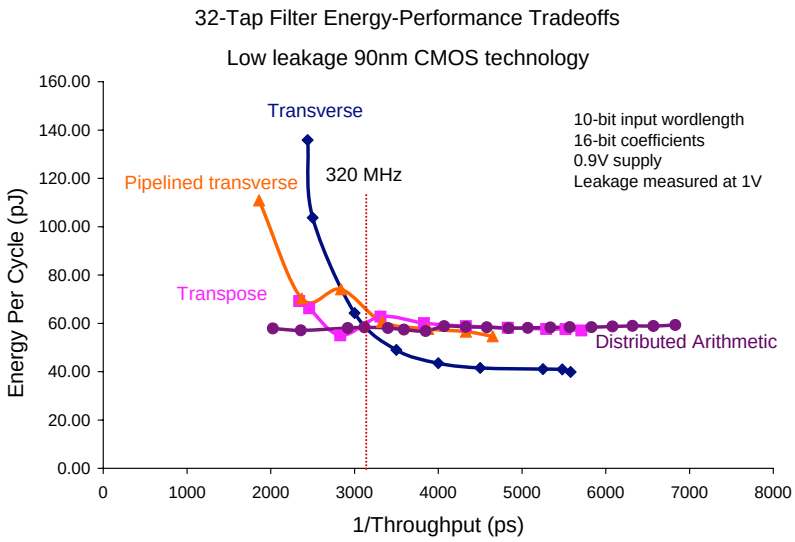


Figure 6.25. 32-tap filter architecture tradeoffs (low-leakage 90nm CMOS process)

The tradeoff analysis carried out in M. Ler's thesis shows that the optimum design for the distributed arithmetic filter is a hybrid parallel, time multiplexed distributed arithmetic architecture that has offset binary encoding to reduce the number of words in the look-up tables. The optimum number (in terms of area and energy efficiency) of partitions was determined to be 16, providing programmability of taps in steps of 8. The partition size is 128x24. Coefficient programmability was not implemented since the energy efficiency significantly reduced with coefficient programmability when two or more memory banks were active [75].

Ler in her design did not implement full clock gating or power gating on the memory blocks. The taped out design presented in this thesis uses Ler's filter design but modifies to include better leakage power management. In the taped out design described in the next section, clock gating and power gating are implemented to improve energy-efficiency.

6.6 Distributed Arithmetic Digital FIR Prototype

The distributed arithmetic filter is preferred for high throughput applications and flexibility in terms of tap programmability, variable input and coefficient word length. For the

given specifications of throughput and clock rate, results from the optimization using a low leakage technology showed that a time-multiplexed, parallel distributed arithmetic filter using an offset binary coding scheme for a partitioned memory was the most energy-efficient flexible digital FIR. The block diagram of the implemented filter is given in Figure 6.10.

This section describes the implementation and chip design for the distributed arithmetic FIR filter. Measured results from the chip prototype are presented at the end of this section.

6.6.1 Filter Overview and Specifications

The flexible distributed arithmetic filter has been fabricated in a low-leakage 90nm static CMOS technology. It supports tap programmability in steps of 8 taps; the number of taps can range from 8 to 64. The filter supports programmability of the input word length with a maximum of 12-bits and programmable in steps of 2. The main clock is at 320MHz and the secondary clock that supports time-multiplexing runs at 160MHz. The clock is generated and divided by a custom designed clock generation circuit using a simple bias and current mirror. This work was done by M. Ler [75]. The energy-delay tradeoff curve for this filter without scan-in and scan-out blocks is shown in Figure 6.26. The core supply is set to 1.0V and the input/output pads run at an external 2.5V. The total area of the chip, including pads is 2mm x 2mm.

The total number of pins is 91 for the chip and one extra ground pin for the I/O reference compensation block that must be included in all 90nm designs taped out in the low-leakage process. The pin-out diagram is given in Figure 6.27. Table 6.6 describes the signals in detail.

The main blocks in the design are summarized in the following.

Clock Generation

The full custom clock generation circuit is designed using a bias circuit, pre-amplifier and a latch. A simple flip-flop is used to divide the clock in half to generate the secondary

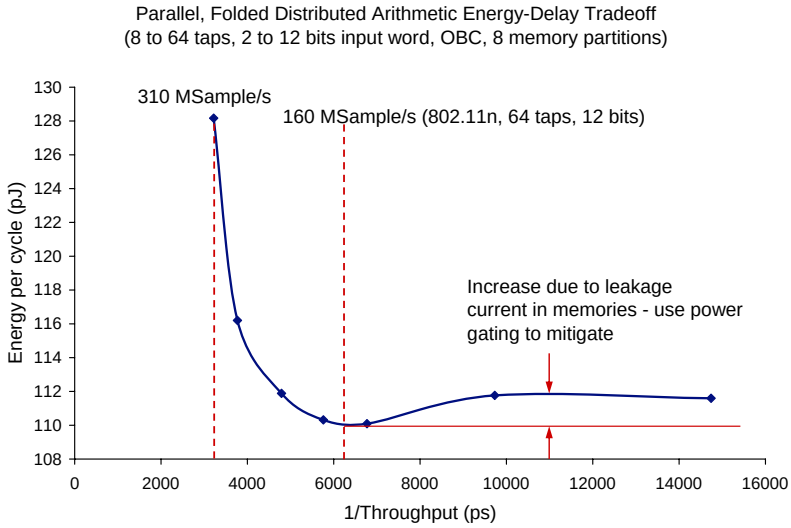


Figure 6.26. Energy-delay tradeoffs of 8–64 tap, 2–12 bit input word, programmable distributed arithmetic filter in 90nm CMOS

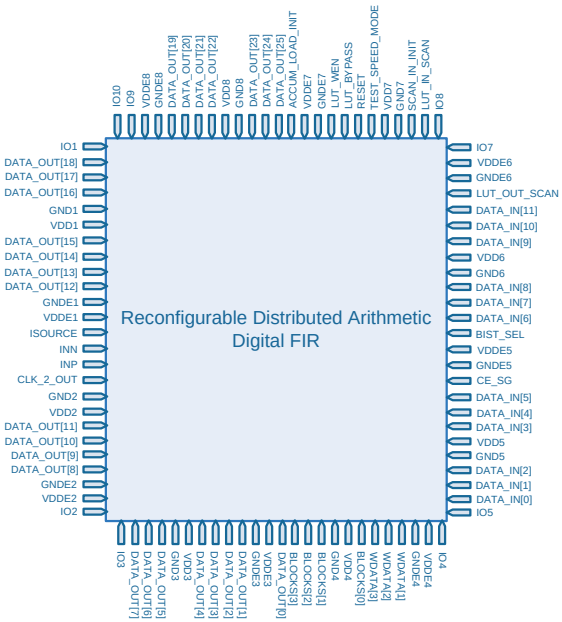


Figure 6.27. Pin-out diagram for programmable distributed arithmetic filter

I/O	NO. OF PORTS	DESCRIPTION
data_in	12	Input signal to be filtered (signed 2's complement)
wdata[3:1]	3	Control bits indicating number of LUTs to be active (wdata[0]=0)
blocks[3:0]	4	Control bits indicating number of blocks (maximum number of taps/number of partitions)
data_out[25:0]	26	Filter output (signed 2's complement)
inn	1	Clock generation input 1
inp	1	Clock generation input 2
isource	1	Clock generation input 3 – current source
LUT_in_scan	1	Input for LUT initialization
LUT_out_scan	1	Output scan of memory contents for testing purposes
LUT_wen	1	LUT write enable
LUT_bypass	1	Memory bypass (active high)
bist_sel	1	Selector for mux between clk and bist_clk for LUT (active high)
reset	1	Reset pin (active low)
scan_in_init	1	Enables scan_in registers (active low)
clk_2_out	1	Divided clock output (160MHz nominal)
test_speed_mode	1	Select between external input and scan chain input
accum_init_load	1	Initialize accumulation tree with offset (active high)
ce_sg	1	Main clock enable
vdd, gnd, vdde, gnde	32	Core and external I/O supply pins

Table 6.6. Programmable Distributed Arithmetic Filter Pin-out Specification

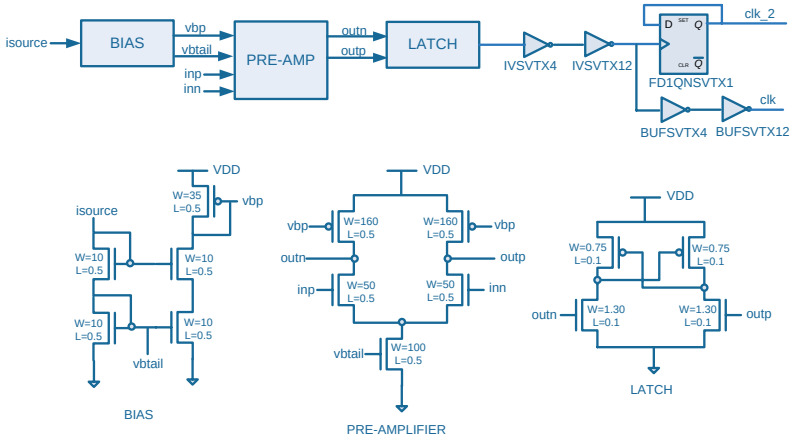


Figure 6.28. Clock generation and division

clock. The clock generation circuit blocks are shown in Figure 6.28. The simulation of this circuit is provided in [75].

Memory, Input, and Output Scan

The look-up table scan in of coefficients, the input word scan in and the output word scan out are coded in module compiler and Verilog is subsequently generated and customized. These blocks are simply a chain of shift registers implementing scan functionality. A block diagram of the scan in relation to the filter is shown in Figure 6.29.

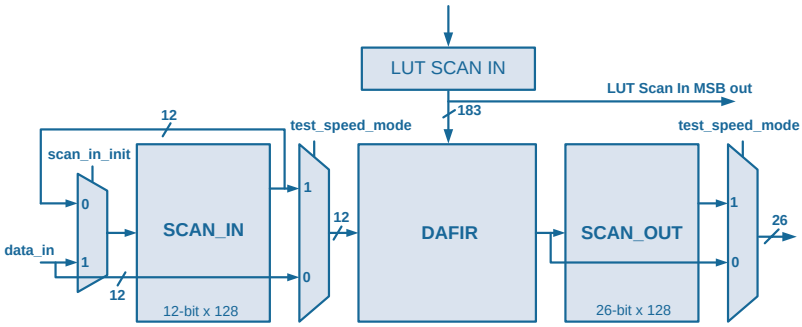


Figure 6.29. Scan in and out

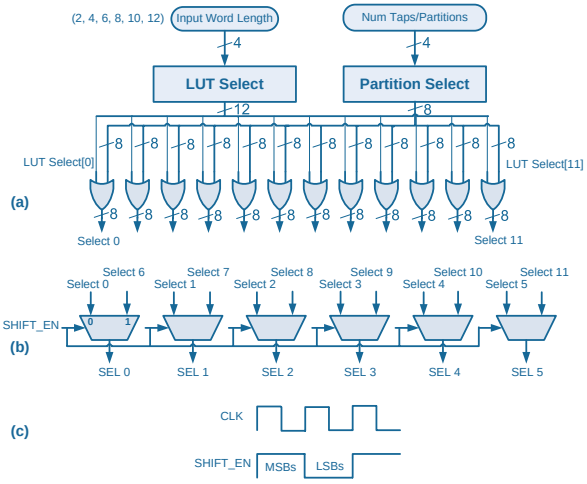


Figure 6.30. Block selection based on input word length and tap requirements

Block Select

Block select and block select multiplexer select the memory partitions in each of the look-up tables based on the number of taps. Six 8-bit select signals are generated based on the control inputs of the number of input word bits (wdata[3:0]) and the number of taps divided by 8 (blocks[3:0]). These two blocks were hand-coded using Verilog. One clock cycle selects the MSBs of the input word and a second cycle is used to select the LSBs. Figure 6.30(a) and (b) show the implementation of the circuits, and Figure 6.30(c) shows the relationship of the main clock to the shift_en signal.

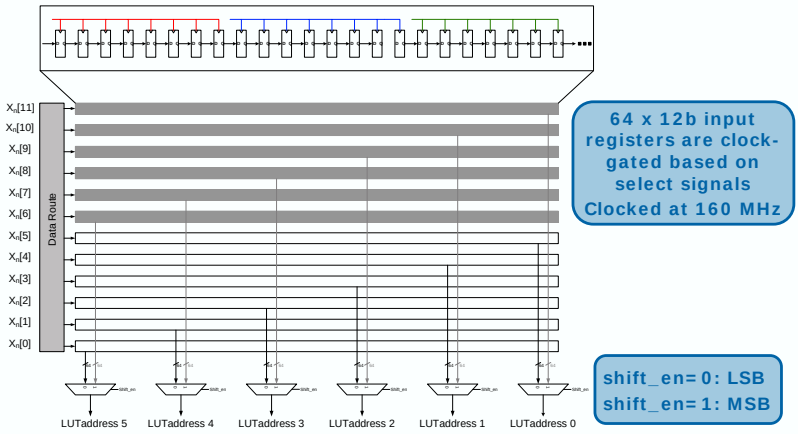


Figure 6.31. Address generation

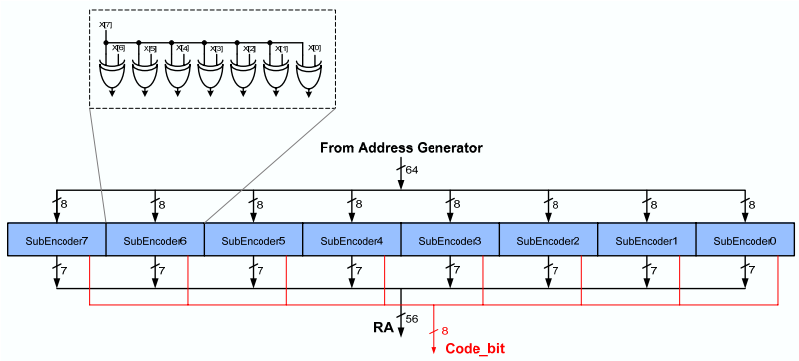


Figure 6.32. Address encoding

Address Generation and Encoding

The address generation, address encoder and input FIFO blocks are designed in custom Verilog and implement address generation from the incoming input word. The encoder encodes the 64-bit generated address using offset binary encoding to produce a 56-bit address and an 8-bit coding signal used later for decoding. A block diagram of its components are shown in Figure 6.31 and Figure 6.32 [75]. The 64x12-bit input registers are clock-gated based on select signals.

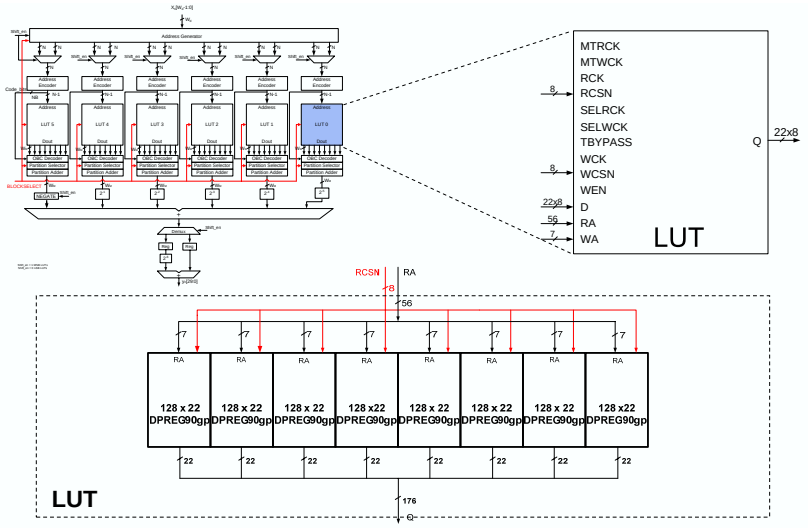


Figure 6.33. Memory overview

Memory

The memory for the look-up tables uses generated memory blocks from the foundry. Custom power gating in the form of sleep transistors was added to the memory block layouts so that the look-up tables can be turned off when the input word is less than 12-bits. The memory is partitioned such that the 64-bit memory address is divided into 8 clusters. The memory requirement is then $8 \cdot 2^8 = 2048$ words for each look-up table. Memory code compression using offset binary encoding further reduces the memory requirement by half, resulting in a total of $8 \cdot 2^7 = 1024$ words for each look-up table. Thus each look-up table has 8 memory blocks, each representing a memory partition. There are 6 replicas of each look-up table, resulting in a total of 48 foundry-generated memory blocks for the entire filter. Figure 6.33 [75] shows an overview of the memory implementation.

Leakage Control

The estimated leakage from synthesis verification shows that power due to leakage current is approximately 50% of the total power for the filter. This is mainly due to leakage from the memories. Power gating is implemented to suppress leakage current when memory

blocks are not being used. The select signals are used to power gate the memories which reduces power due to leakage current. Sleep mode switches are inserted to disconnect register files from power and ground rails. A custom memory cell was created which implemented the power gating for the foundry-generated memory block. The custom cell instantiates a single memory partition which is surrounded by a power ring consisting of ground, V_{DD} supply, and sleep rail V_{DDS}. A PMOS transistor was selected as the sleep transistor since the read degradation due to power gating with a PMOS was 1.5% as compared to 2% when using an NMOS device. The enable signal is tied to the block select lines which control which memory partition must be enabled based on the number of bits in the input word and the required number of taps. An additional NMOS pulldown was included to tie the outputs to ground when the memory partition is not needed. The circuit schematic and layout are shown in Figure 6.34. When all the memories are power-gated off, the leakage power drops from an estimated 87mW to an estimated 9mW (at 320MHz), reducing leakage by 90% (78mW).

Decoding, Partition selection, and Accumulation

The OBC decoder, partition select, and partition accumulation blocks were modeled in Simulink so that the fixed point arithmetic was easily implemented in Verilog using System Generator. The OBC decoder decodes the output of the memory reads and the partition select block outputs the correct word from the enabled partitions. The accumulation block shifts and accumulates the partial coefficient sums from each look-up table resulting the filter output. The appendix contains the figures for each of the Simulink blocks.

6.6.2 RTL and Simulink Modeling

The majority of the blocks were modeled in Module Compiler and Verilog. These blocks were modeled in Simulink as block boxes. The decoding, partition selection, and accumulation were modeled in Simulink and System Generator was used to generate the

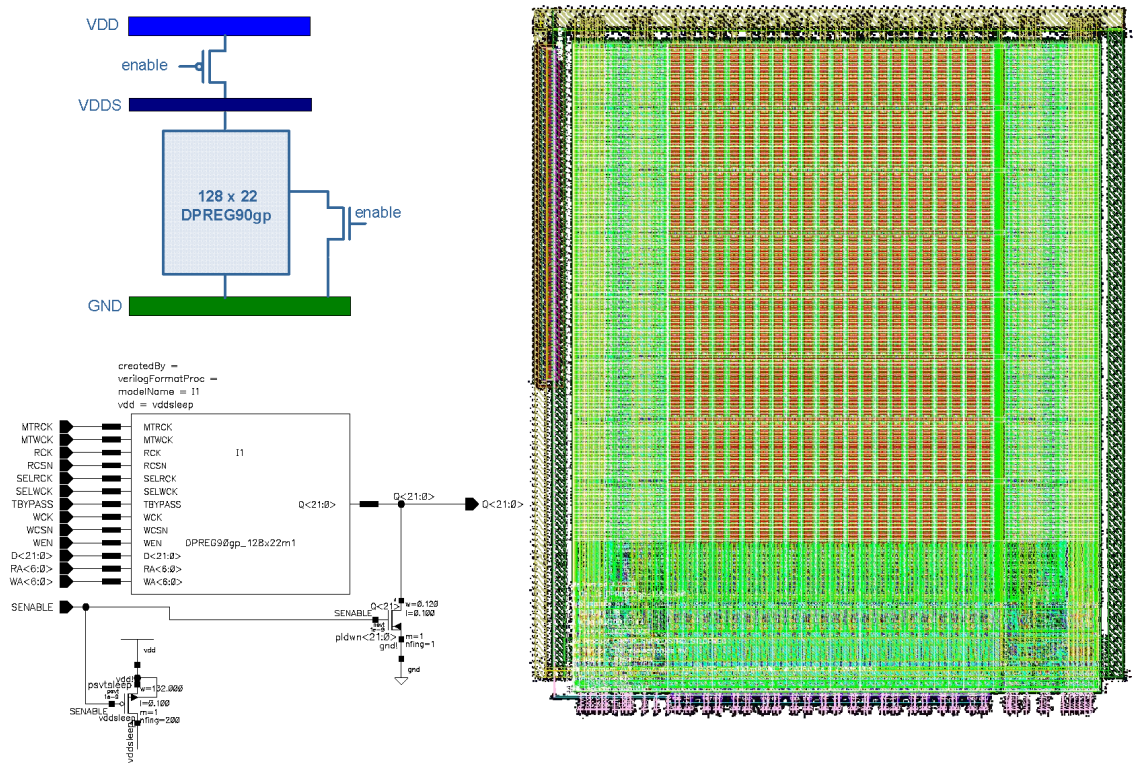


Figure 6.34. Power gating of memory partitions

Verilog for these blocks. The entire system was simulated and verified functionally in Simulink and ModelSim.

6.6.3 Functional Verification

First, a block-by-block simulation was done to verify functionality; the simulation was timing accurate. Then as each block was integrated into the overall system, a separate integration verification was performed, until the entire system was verified. A simple ramp test was first performed to make sure that the filter was walking through each memory location correctly. Then two separate tests for programmability were done: one test checked if it was possible to implement a 32-tap WLAN filter and the second test checked if a 64-tap

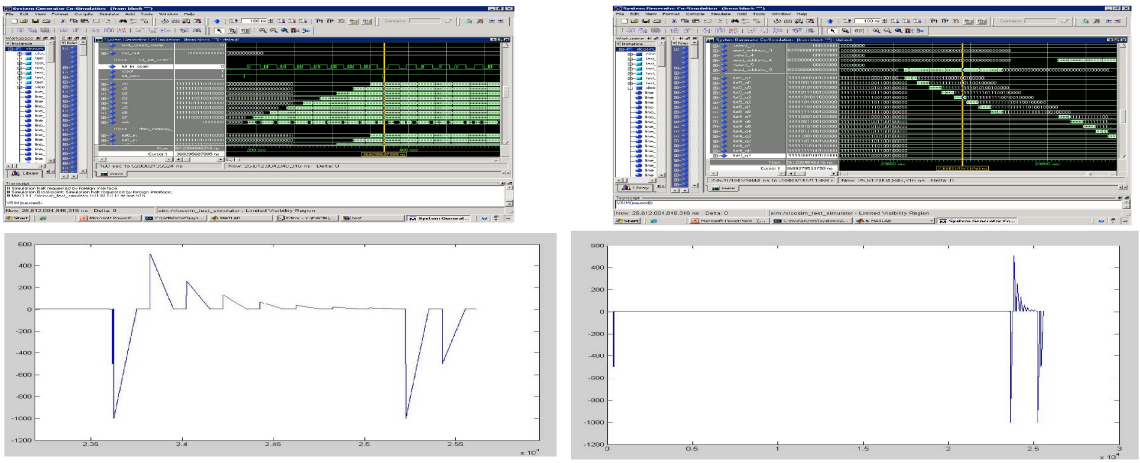


Figure 6.35. Ramp test verification of final taped out design

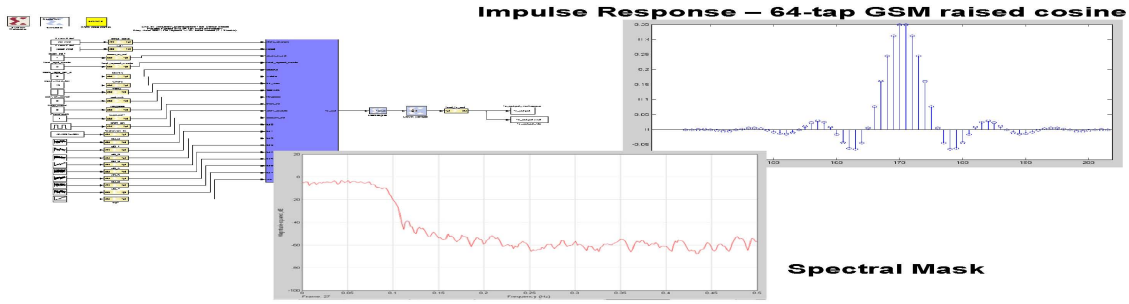


Figure 6.36. GSM test verification of final taped out design

GSM filter was implemented correctly. The same test vectors and models are used for chip testing. Figures 6.35, 6.36, and 6.37 show the results of the verification.

6.6.4 Silicon Implementation and Verification

The entire design was synthesized to an ASIC targeting the low-leakage 90nm CMOS process. Cycle-accurate and bit-accurate co-simulation of the final taped Verilog was performed at the gate-level; the netlist had both SDF and activity factor annotation for timing and power verification. Full-chip synthesis, place and route, LVS, and DRC were carried out

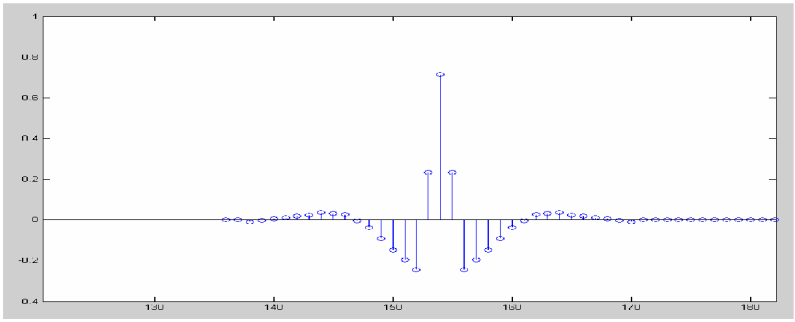


Figure 6.37. WLAN test verification of final taped out design

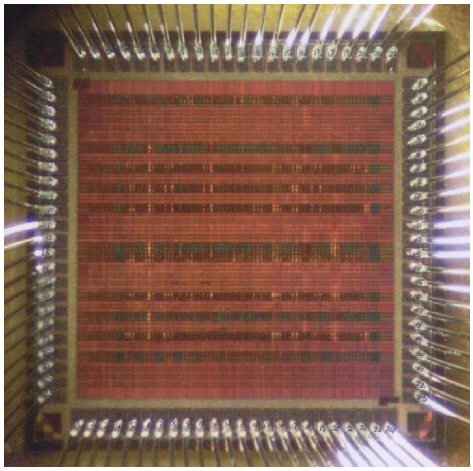


Figure 6.38. Programmable distributed arithmetic filter die photo

using Synopsys Design Compiler and the SOC Encounter flow from the foundry. The estimated power from synthesis is 78mW of dynamic power and 87mW of leakage power (with no power-gating); with power-gating the leakage power drops to 9mW with all memories gated off. The total cell area estimate without input/output pads is 1.08mm^2.

6.6.5 Measured Results

This section presents the results of measurements taken from the distributed arithmetic flexible filter chip. Figure 6.38 shows the chip die photo.

At the time of writing this dissertation, there was a problem found with the supply

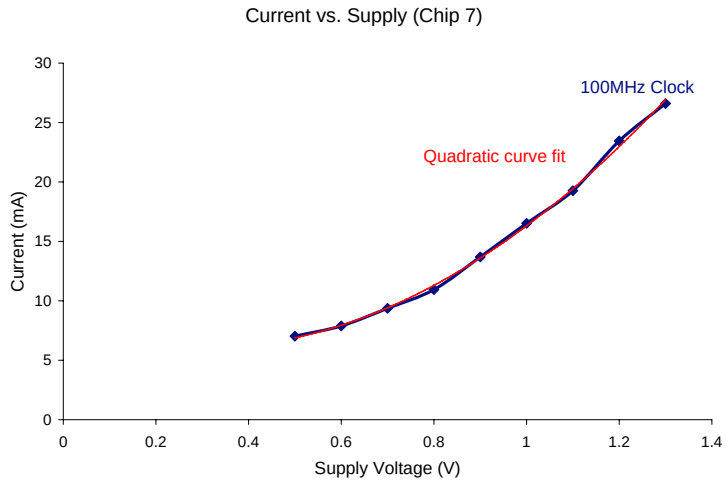


Figure 6.39. Supply versus current

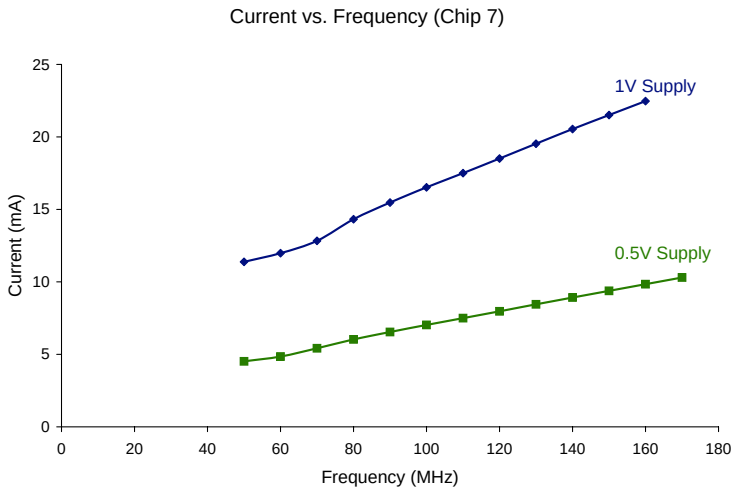


Figure 6.40. Clock frequency versus current

distribution on the test board which was initially designed for a previous version of the chip which had I/O pads at 1V instead of 2.5V. The effect on chip testing is that output of the divided clock, the scan out of the LUTs, and the filter output are not measurable. Work is continuing to locate the source of this issue, however, the chip definitely has a clock tree that is working. A simple test that measures the current drawn by the chip (without input of test vectors) while the clock frequency and supply voltage are varied was carried out. The results are shown in Figures 6.39 and 6.40. They show a linear relationship between change in frequency and current and a quadratic relationship between current and supply voltage. Further testing will continue once the test board is fixed. The test setup is described in the appendix.

6.7 Summary

The architecture exploration and design of a flexible digital filter for a multi-mode, multi-standard wireless radio transceiver was presented. The architecture exploration was performed using a sensitivity-based design methodology which employed composition to rapidly generate energy-delay tradeoff information for numerous architectures. Within a week, it was possible to evaluate all the different architecture choices for a single technology process. The architecture selected for implementation and tape-out was a parallel, folded distributed arithmetic design with clock-gating and power-gating of memories to reduce leakage. It was shown that the optimal architecture choice is highly dependent on the underlying technology (high performance or low leakage) and the memory-to-logic ratio for a particular design. The cost of flexibility was determined to be a maximum of 2 to 4 times that of a filter designed to support a single wireless standard.

Chapter 7

Conclusion and Future Directions

> *Every day you may make progress. Every step may be fruitful. Yet there will stretch out before you an ever-lengthening, ever-ascending, ever-improving path. You know you will never get to the end of the journey. But this, so far from discouraging, only adds to the joy and the glory of the climb.*
> – Sir Winston Churchill

This research addresses a challenging problem with very little formalism around it except at the circuit level. The following list summarizes the accomplishments and progress towards producing a viable sensitivity-based hierarchical design methodology and formalism so that a tool or algorithm maybe developed to automate the entire design process. This dissertation proposes a systematic design methodology for hierarchical power-performance optimization of ASICs, where energy-efficiency is the primary design constraint. The proposed methodology is a hierarchical design optimization framework that cascades design constraints and targets from the system level down to its lower level blocks and circuits in a systematic fashion. Sensitivities to tuning variables are balanced up from lower level blocks to higher-level sub-systems through design composition that meets optimal aggregate sensitivity criteria. Models are used to abstract energy-delay sensitivity to circuit tuning variables so that it is unnecessary to calculate derivatives and to allow circuit-level constraints to flow to higher levels of abstraction. This "top-down, bottom-up" design approach ensures energy-efficiency, consistency and optimality of design decisions across the entire ASIC.

The design methodology is applied to three levels of hierarchy: architecture, micro-architecture, and circuit level. The proposed hierarchical design framework is validated through design of key circuit components of multi-standard mobile platforms. So far it has been manually applied to the design of an optimally energy-efficient flexible digital FIR filter for use in digital front-end components of a multi-standard wireless radio receiver.

7.1 Research Accomplishments

- An optimization framework is conceived using a custom circuit optimizer developed in [17] for power-performance optimization at the leaf cell. Due to the short computation time, this optimizer is a good choice for optimizing small blocks using sizing, supplies, and threshold voltages as circuit tuning variables.

- The viability of using physical circuit parameters to estimate sensitivity is investigated and shown to be instrumental in reducing design time required to uncover power-performance optimal architectures.

- A thorough exploration of employing C_{gate}/C_{wire} as an estimator of sensitivity to gate sizing is presented. Results show that there is a linear relationship between $(C_{gate}/C_{wire}, C_{in})$ and sensitivity to sizing. This implies that C_{gate}/C_{wire} can be used as a first-order estimator of sensitivity to gate size without having to calculate derivatives or evaluate analytical expressions. The investigation of using this metric leads to a thorough study of the impact of wires on sensitivity-based optimization.

- The use of composition rules is investigated to enable fast generation of energy-delay curves for larger circuit blocks comprised of smaller leaf cells. Energy-efficiency curves are generated for multiple architectures for each of the benchmarks within short periods of time (on the order of days), allowing rapid evaluation of architectures in the context of lower level design constraints and tuning variables such as circuit sizing. A multi-standard wireless communication digital filter was used as the prototype benchmark for architecture exploration using this method. Energy-delay curves were

generated for lower-level blocks and composition was used to populate the energy-delay space with a wide range of filter architectures. It was shown that a wide range of filter architectures could be evaluated within a single week as long as specifications and technology remained constant.

- The composition process is formalized into an algorithm that can be implemented as a convex optimization program. This provides an automated mechanism for fast design space exploration at architecture, micro-architecture and circuit levels.

- A digital filter kernel for multi-mode, multi-standard wireless radio system transceiver is designed and optimized using a manual application of the formalized design methodology to demonstrate its viability and flexibility.

7.2 Future Directions

In the future, the goal is to implement the proposed algorithm in software which would provide a means to automatically select optimal architectures for a set of given design constraints. A wider range of benchmarks that are relevant to multi-standard radio systems could be easily evaluated for optimality.

An excellent extension to this work would be to add variability to the problem description which would mean that not one single energy-efficiency boundary is generated but a range is required. A robust optimization approach maybe possible in this case.

A useful exercise would be to evaluate the impact of interconnect on multi-core architecture choices using newer technologies such as 45nm and this methodology once an automated tool is available. The automation would allow rapid evaluation and optimization of a broad range of architectures for multiple types of high-performance systems.

Bibliography

[1] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, July/August 1999.

[2] M. Horowitz and W. Dally, "How scaling will change processor architecture," in *Proceedings of IEEE International Solid State Circuits Conference*, San Francisco, CA, February 2004, pp. 132–133.

[3] L. Counts, "Analog and mixed-signal innovation: The process-circuit-system-application interaction," in *Proceedings of IEEE International Solid State Circuits Conference*, San Francisco, CA, February 2007, pp. 26–32.

[4] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. SC-9, no. 5, pp. 256–268, October 1974.

[5] S. Borkar, "Obeying Moore's Law beyond 0.18μm," in *Proceedings of 13th Annual IEEE International ASIC/SOC Conference*, Arlington, VA, September 2000, pp. 26–31.

[6] R. Ho, "On-chip wires: Scaling and efficiency," Ph.D. dissertation, Dept.of Electrical Engineering, Stanford University, August 2003.

[7] Semiconductor Industry Association. (2005) International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://www.itrs.net/Common/2005ITRS/Home2005.htm

[8] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, vol. 38, no. 8, April 1965.

[9] International Telecommunication Union. (1991-2006) ITU Statistics and Indicators. [Online]. Available: http://www.itu.org

[10] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective, Second Edition.* New Jersey: Prentice-Hall, 2003.

[11] D. Marković, V. Stojanović, B. Nikolić, M. A. Horowitz, and R. W. Brodersen, "Methods for true energy-performance optimization," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 8, pp. 1282–1293, August 2004.

[12] D. M. Marković, "A power/area optimal approach to vlsi signal processing," Ph.D. dissertation, Dept.of EECS, University of California, Berkeley, May 2006.

[13] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, April 2001.

[14] V. Stojanović, D. Marković, B. Nikolić, M. A. Horowitz, and R. W. Brodersen, "Energy-delay tradeoffs in combinational logic using gate sizing and supply voltage optimization," in *Proceedings of the 28th European Solid-State Circuits Conference, ESSCIRC 2002*, Florence, Italy, September 24–26 2002, pp. 211–214.

[15] V. Zyuban and P. Strenski, "Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISPLED)*, Monterey, CA USA, August 2002, pp. 166–171.

[16] H. P. Hofstee, "Power-constrained microprocessor design," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Freiberg, Germany, September 2002, pp. 14–18.

[17] R. Zlatanovici, "Power-performance optimization for digital circuits," Ph.D. dissertation, Dept.of EECS, University of California, Berkeley, December 2006.

[18] R. A. Rohrer, "Fully automated network design by digital computer: Preliminary considerations," *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1929–1939, November 1967.

[19] G. D. Hachtel and R. A. Rohrer, "Techniques for the optimal design and synthesis of switching circuits," *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1864–1877, November 1967.

[20] S. W. Director and R. A. Rohrer, "The generalized adjoint network and network sensitivities," *IEEE Transactions on Circuit Theory*, vol. CT-16, no. 3, pp. 318–323, August 1969.

[21] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, USA, November 1985, pp. 326–338.

[22] A. R. Conn, I. M. Elfadel, W. M. M. Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, "Gradient-based optimization of custom circuits using a static-timing formulation," in *Proceedings of the ACM/IEEE 36th Annual Conference on Design Automation*, New Orleans, LA, USA, June 1999, pp. 452–459.

[23] T. Burd and R. Brodersen, "Energy efficient CMOS microprocessor design," in *Proceedings of the IEEE 28th Annual Hawaii International Conference on System Sciences*, Hawaii, USA, January 3–6 1995, pp. 288–297.

[24] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1283, September 1996.

[25] A. J. Martin, "Towards an energy complexity of computation," *Information Processing Letters*, vol. 77, pp. 123–129, February 2001.

[26] V. Zyuban and P. Strenski, "Balancing hardware intensity in microprocessor pipelines," *IBM Journal of Research and Development*, vol. 47, no. 5/6, pp. 585–598, September/November 2003.

[27] S. Kao, R. Zlatanovici, and B. Nikolić, "A 250ps 64-bit carry-lookahead adder in 90nm CMOS," in *Proceedings of the IEEE International Solid-State Circuits Conference*, San Francisco,CA USA, February 2006, pp. 438–439.

[28] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski, and P. G. Emma, "Balancing hardware intensity in microprocessor pipelines," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 1004–1016, August 2004.

[29] D. Marković, R. W. Brodersen, and B. Nikolić, "A 70 GOPS, 34mw multi-carrier MIMO chip in $3.5mm^2$," in *IEEE 2006 International Symposium on VLSI Circuits: Digest of Technical Papers*, Honolulu, HI, June 2006, pp. 158–159.

[30] T. F. Gonzalez, Ed., *Handbook of Approximation Algorithms and Metaheuristics*. Florida, USA: Chapman and Hall / CRC, Taylor and Francis Group, LLC, 2007.

[31] G. C. Temes and D. A. Calahan, "Computer-aided network optimization: The state-of-the-art," *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1832–1863, November 1967.

[32] D. P. Bertsekas, *Nonlinear Programming, Second Edition*. Belmont, MA USA: Athena Scientific, 1995.

[33] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, United Kingdom: Cambridge University Press, 2004.

[34] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-12, pp. 1621–1634, November 1993.

[35] M. Hershenson, S. Boyd, and T. Lee, "GPCAD: A tool for CMOS op-amp synthesis," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, San Jose, CA, November 1998, pp. 296–303.

[36] S. P. Boyd, S. Kim, D. D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, no. 6, pp. 899–932, November–December 2005.

[37] R. Zlatanovici and B. Nikolić, "Power-performance optimization for custom digital circuits," in *Proceedings of PATMOS 2005*, Leuven, Belgium, September 2005, pp. 404–414.

[38] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. Cambridge, MA: MIT Press, 2001.

[40] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, April 1990.

[41] A. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, April 1995.

[42] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Proceedings of the 39th ACM/IEEE Design Automation Conference*, New Orleans, Louisiana, June 10–14 2002, pp. 783–788.

[43] I. Sutherland, R. F. Sproul, and D. Harris, *Logical Effort:Designing Fast CMOS Circuits*. San Francisco, CA USA: Morgan Kaufmann Publishers, 1999.

[44] Y. Taur and T. H. Ning, *Fundamentals of Modern VLSI Devices*. Cambridge, U.K.: Cambridge University Press, 1998.

[45] B. Nikolić, "Lecture 4: Transistor and gate models," EE 241 Lecture Notes, University of California, Berkeley, Spring 2008.

[46] C. Hu, "Mosfet technology scaling, leakage current, and other topics," EECS 130 Lecture Notes, University of California, Berkeley, Spring 2006.

[47] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power cmos," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, August 1997.

[48] J. Choi and K. Lee, "Design of CMOS tapered buffer for minimum power-delay product," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 9, pp. 1142–1145, September 1994.

[49] B. S. Amrutur and M. A. Horowitz, "Fast low-power decoders for RAMs," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 10, pp. 1506–1515, October 2001.

[50] B. S. Landman and R. L. Russo, "On a pin versus block relationship for partitions of logic graphs," *IEEE Transactions on Computers*, vol. C-20, no. 12, pp. 1469–1479, December 1971.

[51] W. Donath, "Placement and average interconnection lengths of computer logic," *IEEE Transactions on Circuits and Systems*, vol. 26, no. 4, pp. 272–277, April 1979.

[52] ——, "Wire length distribution for placements of computer logic," *IBM Journal of Research and Development*, vol. 25, no. 3, pp. 152–155, May 1981.

[53] J. A. Davis, V. K. De, and J. D. Meindl, "A stochastic wire-length distribution for gigascale integration (GSI) – part I: Derivation and validation," *IEEE Transactions on Electron Devices*, vol. 45, no. 3, pp. 580–589, March 1998.

[54] K. C. Saraswat, "Ee 371 notes: Scaling of interconnections," EE 371 Lecture Notes, Stanford University, Spring 2006.

[55] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," in *Proceedings of the 27th International Symposium of Computer Architecture*, Vancouver, Canada, June 12–14 2000, pp. 248–259.

[56] R. Ho, K. Mai, and M. Horowitz, "Managing wire scaling: A circuit perspective," in *Proceedings of the IEEE 2003 International Interconnect Technology Conference*, Burlingame, CA, June 2–4 2003, pp. 177–179.

[57] M. Joshi, N. S. Nagaraj, and A. Hill, "Impact of (metal) interconnect scaling and process variation on performance," Slide Presentation, Texas Instruments, 2006.

[58] J. Garrett, "Compact models for exploring energy-delay tradeoffs in deeply scaled CMOS digital design," Master's Report, Dept.of EECS, University of California, Berkeley, May 2004.

[59] K. Y. Toh, P. K. Ko, and R. G. Meyer, "An engineering model for short-channel cmos devices," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 4, pp. 584–594, August 1988.

[60] Device Group. (2005) BSIM3/BSIM4 SPICE Device Models. [Online]. Available: http://www-device.eecs.berkekey.edu/~bsim3

[61] A. E. Ruehli, S. P.K. Wolff, and G. Goertzel, "Analytical power/timing optimization technique for digital system," in *Proceedings of the 14th ACM/IEEE Design Automation Conference (DAC 1977)*, New Orleans, Louisiana, USA, June 20–22 1977, pp. 142–146.

[62] S. Augsburger and B. Nikolić, "Combining dual-supply, dual-threshold and transistor sizing for power reduction," in *Proceedings of 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 2002)*, Freiburg, Germany, September 16–18 2002, pp. 316–321.

[63] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization," in *Proceedings of the 2003 International Symposium of Low Power Electronics and Design (ISPLED 2003)*, Seoul, Korea, August 25–27 2003, pp. 158–163.

[64] A. Srivastava, D. Sylvester, and D. Blaauw, "Concurrent sizing, v_dd, and v_{TH} assignment for low-power design," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE 2004)*, vol. 1, Seoul, Korea, February 16–20 2004, pp. 718–719.

[65] D. G. Chinnery and K. Keutzer, "Linear programming for sizing, v_{TH}, and v_{DD} assignment," in *Proceedings of the 2005 International Symposium of Low Power Electronics and Design (ISPLED 2005)*, San Diego, CA, USA, August 8–10 2005, pp. 149–154.

[66] A. Wachter, C. Visweswariah, and A. R. Conn, "Large-scale nonlinear optimization in circuit sizing," *Future Generation Computer Systems*, vol. 21, no. 8, pp. 1251–1262, October 2005.

[67] R. Zlatanovici and B. Nikolić, "Power - performance optimal 64-bit carry-lookahead adders," in *European Solid-State Circuits Conference*, Estoril, Portugal, February 16–18 2003, pp. 321–324.

[68] M. Horowitz, "Ee371 handout: Logic effort revisited," EE 371 Handout, Stanford University, Spring 1998–1999.

[69] G. Dimitrakopoulos and D. Nikolos, "Closed-form bounds for interconnect-aware minimum-delay gate sizing," *Lecture Notes on Computer Science*, vol. 3728, pp. 308–317, August 2005.

[70] A. Morgenshtein, E. G. Friedman, R. Ginosar, and A. Kolodny, "Timing optimization in logic with interconnect," in *Proceedings of 2008 International Workshop on System-Level Interconnect Prediction*, Newcastle, United Kingdom, April 5–6 2008, pp. 19–26.

[71] E. S. Fetzer, M. Gibson, A. Klein, N. Calick, C. Zhu, E. Busta, and B. Mohammad, "A fully bypassed six-issue integer datapath and register file on the Itanium-2 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1433–1440, November 2002.

[72] L. Maurer, T. Burger, T. Dellsperger, R. Stuhlberger, M. Schmidt, and R. Weigel, "On the architectural design of frequency-agile multi-standard wireless receivers," in *IST Mobile and Wireless Summit*, Dresden, Germany, June 19–23 2005.

[73] E. Tsui, A. Chun, and K. Skeba, Private discussions, Intel Corporation, 2005–2006.

[74] F.Sheikh, M. Ler, R. Zlatanovici, D. Marković, and B. Nikolić, "Power-performance optimal dsp architectures and asic implementation," in *Proceedings of Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove,CA USA, October 29 – November 1 2006, pp. 1480–1485.

[75] M. Ler, "An energy-efficient reconfigurable FIR architecture for a multi-protocol digital front-end," Master's Report, Dept. of EECS, University of California, Berkeley, March 2006.

[76] T. Gemmeke, M. Gansen, H. J. Stockmanns, and T. G. Noll, "Design optimization of low-power high-performance DSP building blocks," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 7, pp. 1131–1139, July 2004.

[77] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* New York, USA: John Wiley & Sons, Inc., 1999.

[78] L. Thon, P. Sutardja, F. Lai, and G. Coleman, "A 240mhz 8-tap programmable fir filter for disk-drive read channels," in *IEEE International Solid-State Circuits Conference*, San Francisco,CA USA, February 15–17 1995, pp. 82–83.

[79] R. B. Staszewski, K. Muhammad, and P. Balasra, "A 550-msample/s 8-tap FIR digital filter for magnetic recording read channels," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 8, pp. 1205–1210, August 2000.

[80] W. Wilhelm and T. G. Noll, "A new mapping technique for automated design of highly efficient multiplexed FIR digital filters," in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems*, vol. 4, Hong Kong, June 9–12 1997, pp. 2252–2255.

[81] D. Moloney, J.O'Brien, E. O'Rourke, and F. Brianti, "Low-power 200-mbps, area-efficient, five-tap programmable FIR filter," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 7, pp. 1134–1138, July 1998.

[82] K. Kim and K. Lee, "Low-power and area-efficient FIR filter implementation suitable for multiple taps," *IEEE Transactions on VLSI Systems*, vol. 11, no. 1, pp. 150–153, February 2003.

[83] S. Rylov, A. Rylyakov, J. Tierno, M. Immediato, M. Beakes, M. Kapur, P. Ampadu, and D. Pearson, "A 2.3 GSample/s 10-tap digital FIR for magnetic recording read channels," in *IEEE International Solid-State Circuits Conference*, San Francisco,CA USA, February 5–7 2001, pp. 190–191.

[84] D. J. Pearson, S. K. Reynolds, A. C. Megdanis, S. Gowda, K. R. Wrenner, M. Immediato, and R. L. Galbraith, "Digital FIR filters for high speed PRML disk read channels," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 12, pp. 1517–1523, December 1995.

[85] J. Tierno, A. Rylyakov, S. Rylov, M. Singh, P. Ampadu, S. Nowick, M. Immediato, and S. Gowda, "A 1.3GSample/s 10-tap full-rate variable-latency self-timed FIR filter with clocked interfaces," in *IEEE International Solid-State Circuits Conference*, San Francisco,CA USA, February 3–7 2002, pp. 60–61.

[86] C. Lutkemeyer and T. G. Noll, "A transversal equalizer with an increased adaptation speed and tracking capability," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, pp. 503–507, March 1998.

[87] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4–19, July 1989.

[88] Z. Guo, A. Carlson, L.-T. Pang, K. Duong, T.-J. King-Liu, and B. Nikolić, "Large-scale read/write margin measurement in 45nm CMOS SRAM arrays," in *Proceedings of 2008 IEEE Symposium on VLSI Circuits*, Honolulu, HI, USA, June 18–20 2008, pp. 42–43.

[89] W. R. D. et. al., "An automated design flow for low-power, high-throughput dedicated signal processing systems," in *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA USA, November 4–7 2001, pp. 475–480.

Appendix A

Simulink to ASIC Design Methodology

A.1 Design Flow

The distributed arithmetic flexible filter prototype that is fabricated in a 90nm static CMOS technology was synthesized using a Simulink to ASIC design flow developed at the Berkeley Wireless Research Center [89].

The design is entered as Simulink functional blocks, providing a data flow description of the system. From the Simulink environment, Xilinx System Generator is used translate the graphical representation into Verilog or VHDL code that can then be synthesized to a gate level netlist. Custom Verilog/VHDL blocks can be inserted into the Simulink model as black boxes. Once the code is synthesized to a gate-level netlist, it can be instantiated in the Simulink environment and co-simulated with ModelSim, as described in Chapter 6.

The Simulink environment is also used for chip testing as described in Appendix B. This appendix describes the Simulink models for each of the blocks of the distributed arithmetic filter.

A.2 Distributed Arithmetic FIR Simulink Models

The various Simulink models for the distributed arithmetic FIR are provided here for reference. The address generator, input FIFO, block select, memories, memory encoder, and scan blocks are all coded in Verilog and instantiated as black boxes in the Simulink model. The decoding, accumulation and shift functions are modeled in Simulink and code is automatically generated using System Generator from Xilinx.

The system Simulink model is shown in Figure A.1.

The decode, accumulation, and shift functions are modeled using Simulink and Xilinx building blocks. The main reason for implementing the backend of the filter in this manner was to take advantage of the floating to fixed point conversion. Figure A.2 shows the various functional blocks of the back end. The partial coefficient sums are read from memory and

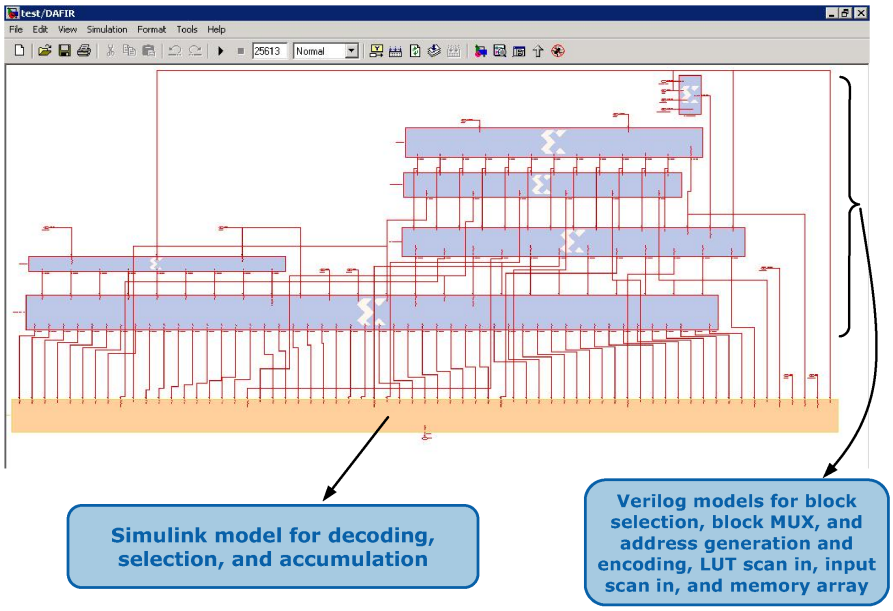


Figure A.1. System level Simulink model for the distributed arithmetic flexible FIR

decoded based on the current configuration of the filter in terms of number of taps and input word length. Then the sums are shifted appropriately and accumulated. The accumulation must be initialized with the $A(0)$ term as described in Chapter 6 which is integrated into the programmable shift and final accumulation portion of the model. Figure A.3 shows the partition selection, decode, and accumulation for partial coefficient sums read from a single 8-partition memory block. Figure A.4 shows Simulink model for the offset binary decoding block for each 8-partition memory block. The decoding simply checks if the memory contents need to be negated or kept as is. Figure A.5 shows Simulink model for the partition selection for each look-up table. Figure A.6 shows Simulink model for the accumulation of partial coefficient sums for each look-up table. Figure A.7 shows Simulink model for the accumulation of all partial coefficient sums from all six look-up tables. Figure A.8 shows Simulink model for the shift of partial coefficient sums based on the number of taps and input word length.

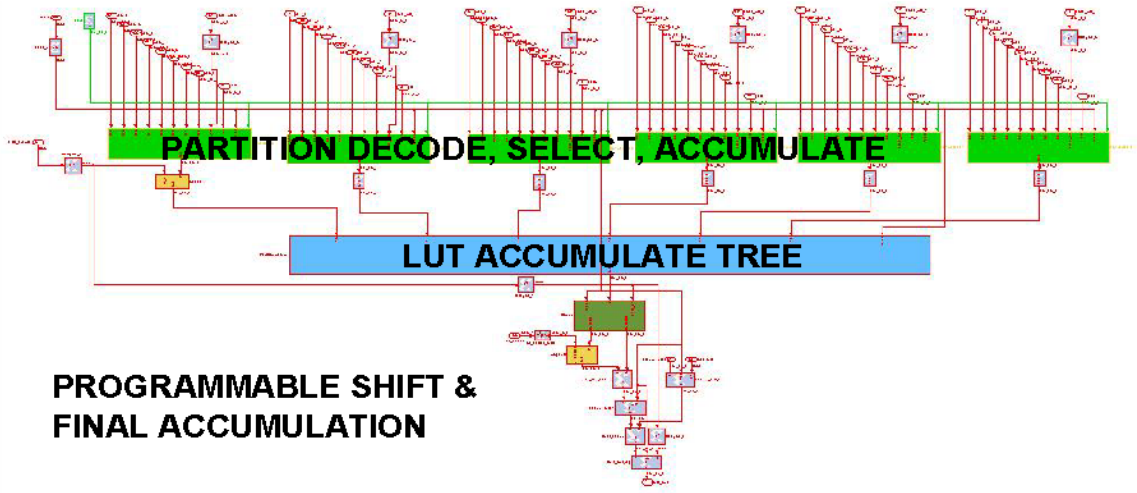


Figure A.2. Partition select, decode, and accumulation tree

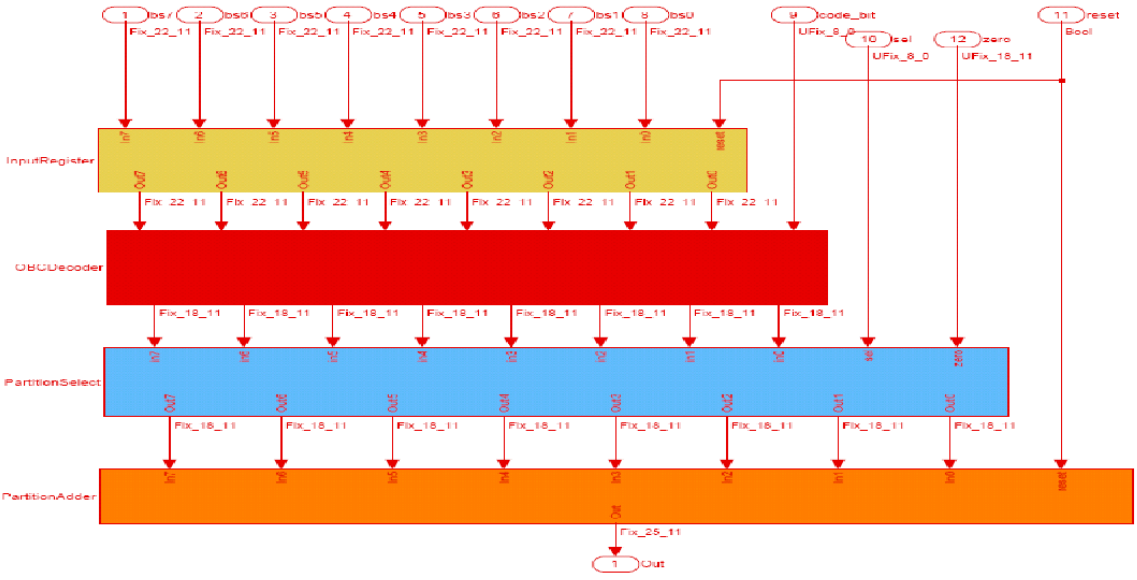


Figure A.3. Partition select, decode and accumulation for each look-up table

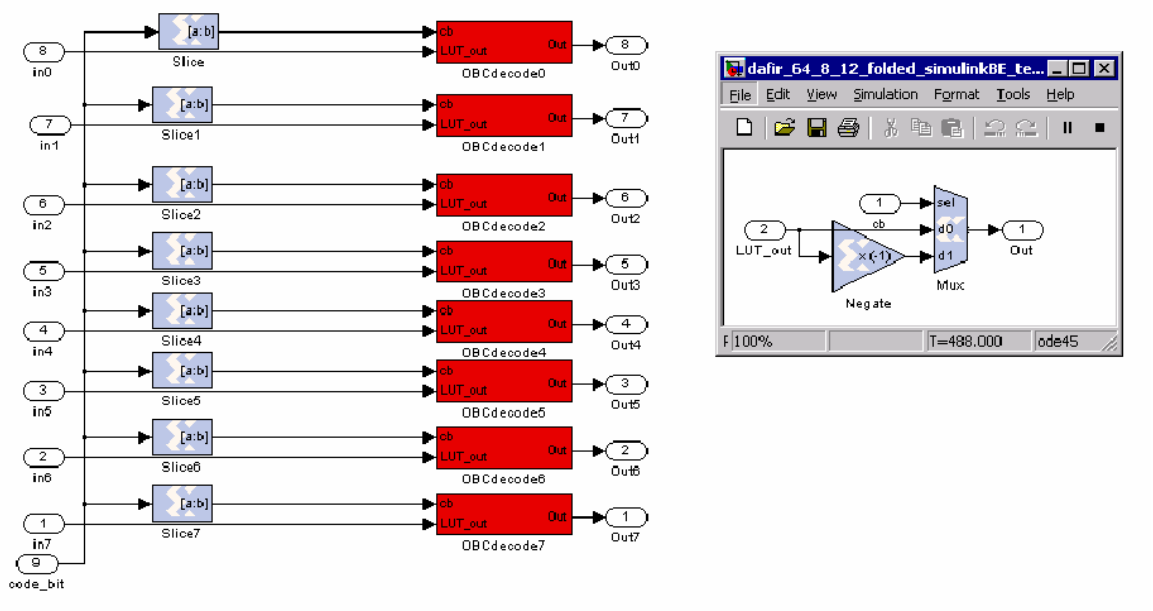


Figure A.4. Look-up table OBC decoder

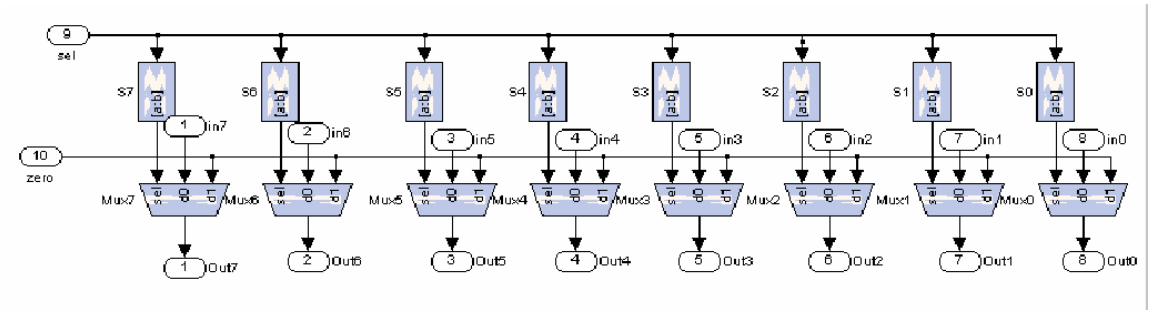


Figure A.5. Partition select for each look-up table

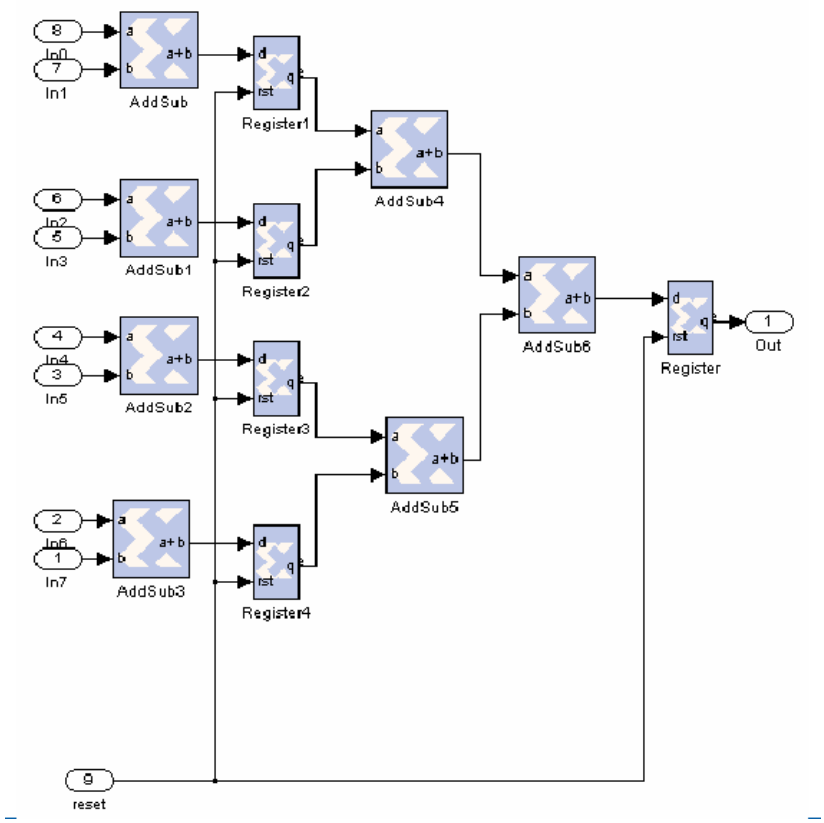


Figure A.6. Single look-up table accumulation

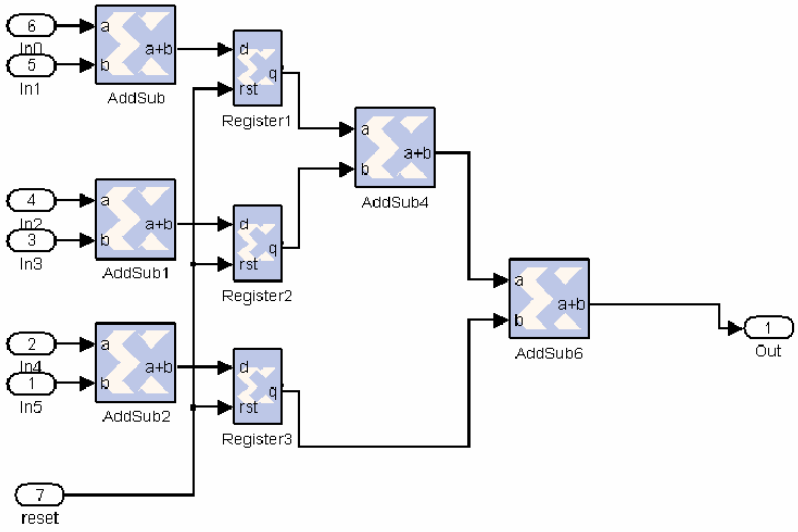


Figure A.7. Final accumulation tree

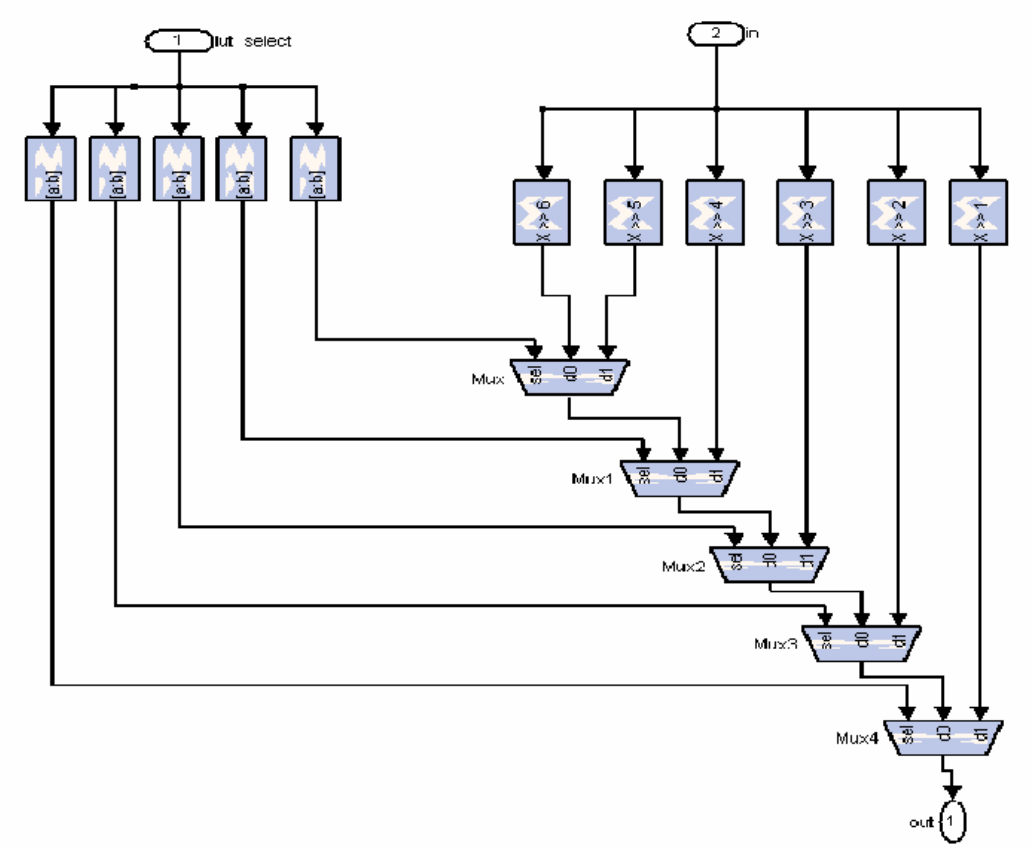


Figure A.8. Programmable shift

Appendix B

Test Setup for Distributed Arithmetic Prototype

B.1 Test Methodology

The Simulink environment is also used for testing fabricated prototypes in the lab. The Simulink design and test vectors are programmed onto an i-BOB (infiniband break-out board) board that houses memory and FPGAs; the i-BOB feeds data to the ASIC inputs. The i-BOB board also samples the data output from the ASIC for comparison with expected results. The read/write ports of the block RAMs, clock enable, reset, and other control signals can be set by the user through a software interface to the i-BOB.

B.2 Lab Setup

The lab setup is shown in Figure B.1. The picture shows the connection of the i-BOB with the ASIC test board through ribbon cables. A general purpose laptop running the interface software for the i-BOB is used to monitor and set control signals on the i-BOB. Programming the i-BOB is carried out via the serial port on the laptop. Figure B.2 shows the communication between the laptop and the i-BOB board.

The i-BOB supply is set to 5.0V and the general purpose I/Os operate at 2.5V which is the same supply as the I/O pads of the test chip. The ASIC core operates at 1.0V. There are four supply domains on the ASIC test board. The $\pm$10V and $\pm$3.3V domains are required for the current supply and differential pair biasing of the clock generation circuit on the test chip. The 2.5V and 1.0V domains are required for the ASIC chip. Figure B.1 shows a picture of the test board housing the ASIC test chip.

The ASIC test board was designed for the initial version of the distributed arithmetic prototype which did not have power gating of memories. The initial prototype used 1.0V chip I/Os hence the board required resistive dividers to down-shift inputs to the chip, while active comparators were used to up-convert outputs to 3.3V. In the second version of the prototype, these are no longer required as the chip I/Os operate at 2.5V. Extensive testing revealed that the ad-hoc modifications made to the ASIC test board were insufficient and

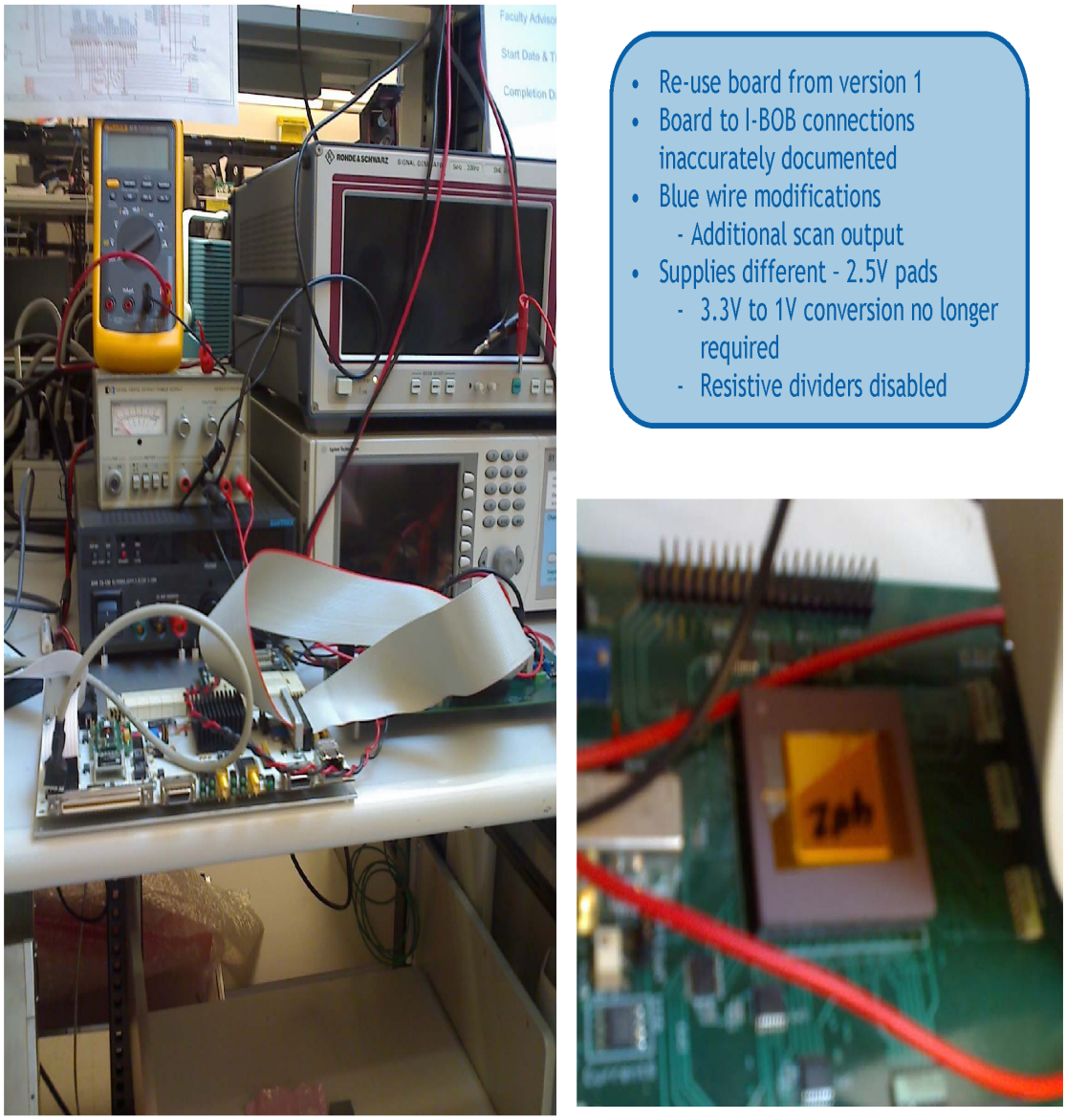


Figure B.1. ASIC test board and connection to i-BOB

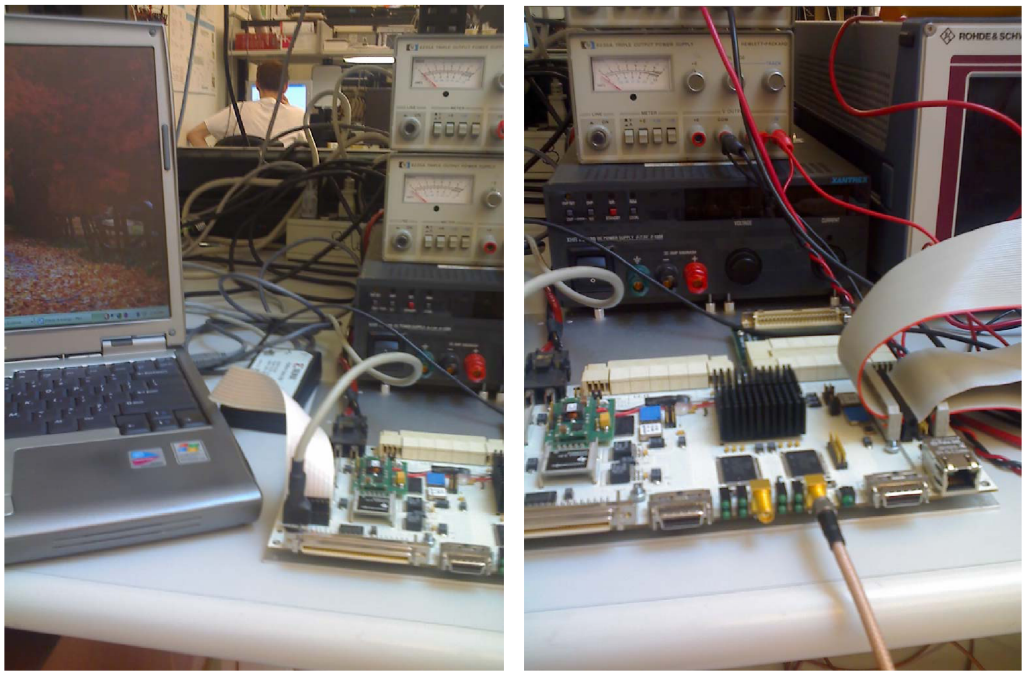


Figure B.2. i-BOB and connection to laptop

hence it was difficult to complete testing of the prototype. At the time of writing this dissertation the board is undergoing modifications to simplify its design.