UNIVERSITY OF CALIFORNIA
Santa Barbara

Algorithm/Architecture Design Space Co-exploration for Energy Efficient Wireless Communications Systems

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

Doctor of Philosophy

in

Electrical and Computer Engineering

by

Yan Meng

Committee in Charge:

Professor Ryan Kastner, Chair

Professor Timothy Sherwood, Co-Chair

Professor Forrest Brewer

Professor Ronald A. Iltis

September 2006

The Dissertation of Yan Meng is approved:

Professor Forrest Brewer

Professor Ronald A. Iltis

Professor Timothy Sherwood, Co-Chair

Professor Ryan Kastner, Chair

September 2006

Algorithm/Architecture Design Space Co-exploration for Energy Efficient

Wireless Communications Systems

by

Yan Meng

To mom and dad, Chunxia and Karen

Acknowledgements

This work described in this thesis could not have been accomplished without the help of others. Foremost, I would like to thank my research advisors Professor Ryan Kastner and Professor Timothy Sherwood for their guidances, supports and encouragement during my Ph.D. study. The vision and resources you provided have been truly extraordinary. I believe what I have learned from you will continue guiding me well in my future career. I would also like to thank Professor Forrest Brewer and Professor Ronald A. Iltis for reviewing this thesis and for providing helpful comments.

I have had the pleasure working with many fellow researchers in the ExPRESS lab, the Arch lab, and the Telemetry lab. Andrew P. Brown has helped me to better understand many aspects of digital communications. Thanks Andrew for explaining the channel estimation and multi-user detection theories for me. The success of the channel estimation project would not have been possible without your help and advice. I would also like to thank my lab-mates Anup Hosangadi, Gang Wang, Wenrui Gong, Shashi Mysore, Ted Huffmire, and Banit Agrawal for our friendship and for discussing interesting research ideas.

I extend my deepest appreciation for my parents. They are always there for me and always have faith in me. Your unconditional love and continual encouragement have been a source of strength without which I would have never got so far. Thank you for supporting me from the bottom of my heart.

Finally, I would like to dedicate this work to my family: Chunxia, and little Karen. Chunxia, from the first day we met, your love and support have been constantly there for me. Without your love, your caring, and your devotion, I can't think any of this. For my little daughter, your cute smiles always relieve me and refresh my mind and heart. For these reasons, I dedicate this dissertation to you with all my love.

Curriculum Vita
Yan Meng

Born

August 28, 1978	Linhe, Inner Mongolia, China

Education

1998	B.S. in Electrical Engineering, Inner Mongolia Polytechnic University, China
2001	M.S. in Computer Science, Tsinghua University, China
2003	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara
2006	Ph.D. in Electrical and Computer Engineering, University of California, Santa Barbara

Experience

09/98 – 08/01	Graduate Research Assistant, State Key Lab of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University
08/03 – 12/03	Intern in Compiler Group, Sun Microsystems Inc., Menlo Park, California
06/06 – 09/06	Intern in Catapult C Synthesis Group, Mentor Graphics Corp., Wilsonville, Oregon
09/01 – present	Graduate Research Assistant, Department of Electrical and Computer Engineering, University of California, Santa Barbara

Selected Publications

Yan Meng, Timothy Sherwood, and Ryan Kastner, "Leakage Power Reduction of Embedded Memories on FPGAs Through Location Assignment", In *Proceedings of the 43rd Design Automation Conference (DAC)*, July 2006.

Andrew P. Brown, Yan Meng, Ronald A. Iltis, Timothy Sherwood, Hua Lee and Ryan Kastner, "Matching Pursuit Channel Estimation: Efficient Algorithm Design and Reconfigurable Hardware Implementation", *submitted to IEEE Transactions on Wireless Communications, 2005*

Yan Meng, Andrew P. Brown, Timothy Sherwood, Ronald A. Iltis, Hua Lee and Ryan Kastner, "MP Core: Algorithm and Design Techniques for Efficient Channel Estimation in Wireless Applications", In *Proceedings of the 42nd Design Automation Conference (DAC)*, June 2005.

Yan Meng, Ryan Kastner, and Timothy Sherwood, "Algorithm Architecture Co-exploration for Designing Energy Efficient Wireless Channel Estimator", In *Mobile Computing Hardware Architectures: Design and Implementation (MOCHA), co-located with the Hawaii International Conference on System Sciences (HICSS)*, January 2006

Yan Meng, Wenrui Gong, Ryan Kastner, and Timothy Sherwood, "Algorithm Architecture Co-exploration for Designing Energy Efficient Wireless Channel Estimator", In *Journal of Low Power Electronics, (JOLPE)*, December 2005

Wenrui Gong, Yan Meng, Gang Wang, Ryan Kastner and Timothy Sherwood, "Data Partitioning for Reconfigurable Architectures with Distributed Block RAM", In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, June 2005

Yan Meng, Timothy Sherwood and Ryan Kastner, "Exploring the Limits of Leakage Power Reduction in Caches", In *ACM Transactions on Architecture and Code Optimization (TACO)*, November 2005

Yan Meng, Timothy Sherwood and Ryan Kastner, "On the Limits of Leakage Power Reduction in Caches", In *International Symposium on High-Performance Computer Architecture (HPCA)*, February 2005

Timothy Sherwood, Ryan Kastner, Yan Meng, Lin Tan, and Shreyas Prasad, "Supporting Interdisciplinary Domain Specific Architecture Research with Reconfigurable Devices", *Workshop on Architecture Research using FPGA Platforms*, co-located with the *International Symposium on High-Performance Computer Architecture (HPCA)*, February 2005

Arun Qamra, Yan Meng, and Edward Chang, "Enhanced Perceptual Distance Functions and Indexing for Image Near-replica Detection", In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2005

Yan Meng, Edward Chang, and Beitao Li, "Enhancing DPF for Near-replica Image Recognition", In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Madison, Wisconsin, June, 2003

Yan Meng and Edward Chang, "Image Copy Detection Using Dynamic Partial Function", In *IS&TSPIE International Conference on Storage and Retrieval for Media Databases*, San Jose, January 2003.

Yan Meng and Zengqi Sun, "Design of a Fuzzy-Adaptive Kalman Filter", In *Journal of Computer Engineering and Its Application*, 2002

Yan Meng, Zengqi Sun, and Jihong Zhu, “Fuzzy Controller Design for Smooth Changes between Modes”, In *Journal of Computer Engineering and Its Application*, 2002

Abstract

Algorithm/Architecture Design Space Co-exploration for Energy Efficient Wireless Communications Systems

Yan Meng

Wireless connectivity is playing an increasingly important role in communication systems. Advanced communication algorithms have been developed to combat multi-path and multi-user interference, as well as to achieve increased capacity and higher data rate. To meet the need of high performance with low energy consumption requires designs rely on the integration of both algorithm and architecture to exploit the full potential of theoretical communication results and advanced CMOS technology.

In this work, these strategies were employed to exploit the interactions between algorithm and architecture to derive fast and energy efficient solutions of wireless communications systems. First, a cross-cutting method is proposed to facilitate algorithm/architecture co-exploration of dedicated hardware design, resulting in novel optimizations at every level of the system design, from the theory and algorithms to the arithmetic and placement.

Second, through algorithm/architecture co-exploration, multiple orders of magnitude of improvement in both performance and energy efficiency can be obtained by using highly parallel architecture and dedicated hardwired IP cores compared with executing software on microprocessors. As a case-study, a matching pursuit (MP) core

was designed on a modern FPGA for efficient wireless channel estimation, achieving 216x speedup compared to a high performance microprocessor.

Third, the advantages of bringing together algorithm and architecture design are also demonstrated on optimizing leakage power consumption, which is projected to be the dominant power dissipation in future generations of digital systems as current technology scaling trends hold. In this study, the growing problem of transistor leakage is tackled through on-chip memories: caches in general purpose microprocessors and embedded memories in reconfigurable devices, which take the majority of on-chip real estates and identify themselves the ideal targets to battle the leakage problem. Through exploring the temporal and spatial information of variables within memories, substantial leakage power can be reduced with existing low leakage circuit and architecture techniques (5x reduction for instruction cache, and 2x reduction for data cache, compared to existing techniques). With leakage-aware data placement, significant leakage power reductions can be further achieved. On the case study of MP core, 94% of leakage power on embedded memories can be saved.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Wireless connectivity is playing an increasingly significant role in communication systems. Evolving wireless multimedia applications along with limited spectrum resources demand higher data rate and higher multi-user capacity. At the same time, low energy consumption and small silicon area (which directly relates to cost) are critical to meet portability requirements of future wireless systems. High performance and energy efficient implementations of advanced communications algorithms remain as a design challenge, which requires optimization at all levels of wireless system design: from the theory and algorithms to the arithmetic and placement.

First, at the algorithm level, new advances in information theory and communication algorithms have either been proved or demonstrated unprecedented wireless data rate / bandwidth efficiency. However, many of these sophisticated algorithms require very high computational power, and have not been possible for high data rate and low

energy consumption implementation using conventional implementations for various applications that require wireless connectivity.

The wireless application domain, using digital baseband signal processing for wireless communications, has distinctive characteristics that can be leveraged for reducing implementation costs:

1. Algorithms have high degrees of both temporal and spatial concurrency / parallelism.

2. Primary computation is dominated by a few kernels that take a relative large amount of computation.

Second, since the invention of the transistor in 1947, integrated circuits (IC), which have been permeating our world in almost every possible respect and constantly expanding, have been improved following Moore's law: the number of transistors per die has been doubled every 18 months. With huge transistor real estate available, four primary architectures, namely general purpose microprocessors, configurable processors, reconfigurable devices, and application specific integrated circuit, for the execution of algorithms have been evolved to meet the digital design quality metrics:

1. Performance — Performance or execution time of a behavior is defined as the average time required by the behavior from start to finish. It mainly depends on system clock rate and the amount of parallelism that are supported.

2. Power — Excessive power dissipation in integrated circuits not only discourages their use in a portable environment, but also cause overheating, which degrades performance and reduces chip lifetime.

3. Area — Area is directly related to the hardware cost. It also decides whether certain portion of the design can fit into a given chip area. Moreover, smaller area indicates shorter wires in the interconnect network, leading to higher performance and less interconnection power consumption.

4. Others — Design for testability, design for reliability, design for manufacturing, design time, and time-to-market are also important factors to consider in digital system design.

Indeed, Semiconductor Industry Association has identified power as a critical factor in designing a digital system. Higher energy dissipation requires more expensive packaging and cooling technology, which in turn increases cost and decreases system reliability. There are fundamentally two ways in which power can be dissipated: either dynamically (due to the switching activity of repeated capacitance charge and discharge on the output of the millions of gates), or statically (due to sub-threshold, gate, and junction leakage). Dynamic power consumption is proportional to the square of the supply voltage, which reduces as process technology scales. Scaling down the transistor supply voltage reduces the dynamic power dissipation. Yet, to maintain high switching speed under reduced voltages, the threshold voltage must also be scaled. As

the threshold voltage drops, it is easier for current to leak through the transistor, resulting in significant leakage power dissipation. If current technology scaling trends hold, leakage will soon become the dominant source of power consumption. As such new techniques are needed to battle this growing problem.

Third, architecture design plays a key role to optimize performance and energy efficiency to design advanced communications algorithms. Conventional microprocessor designs based on Von Neumann architecture was developed under the assumption that hardware was expensive and sharing of the hardware resources was absolutely necessary. The situation now, however, for system on a chip has changed. Hardware is almost free with potentially thousands of multipliers and embedded memories on a die. For the emerging wireless application domain the best way to utilize the available IC technology needs to be investigated. Moreover, with the fact that algorithm complexity has been increasing faster than Moore's law indicates if the architecture is not tuned to exploit the improved characteristics, purely relying on technology scaling will be not sufficient to deliver high performance and energy efficiency for more advanced algorithms that are demanded by wireless systems in the near future.

Therefore, an efficient design relies on the integration of algorithm developments and architecture design to exploit the full potential of communications theoretical results and advanced technology. Thus, there is a need to perform algorithm level selections and modifications based on efficient implementation criteria, which can lead to greatly reduced processing requirements without system performance degradation.

There is also a need to design architectures to adapt to the computational requirements of the algorithm, which can lead to largely reduced energy consumption, in particular leakage power reduction, of the implementation. Unfortunately, the lack of such a unified and systematic design method hurdles the exploration of various realizations over a broad range of algorithmic and architectural options.

1.2 Research Goals

The goal of this research is to bring architecture design together with system and algorithm design and technology to obtain a better understanding of the key trade-offs of important architecture parameters and thus more insight in digital signal processing architecture optimization in order to facilitate energy efficient design of high performance wireless communications algorithms. The research takes as an example one of the fundamental units in wireless communications systems: channel estimation. The results are extensible to other wireless communications systems, e.g, channel estimation in wireless sensor networks and DS-CDMA base stations.

1.2.1 Algorithm/Architecture Co-exploration

The proposed design method enables efficient algorithm and architecture design. The major strategies are:

1. In algorithm development, algorithm performance has been treated as the single most important criterion to evaluate and choose among possible algorithms. The inherent structures of algorithms such as parallelism are discovered based on the characteristics of algorithms to assist algorithm selection and architecture exploration. The performance evaluation metrics include not only the traditional performance measures, but also the feasibility for low power implementation.

2. Given an algorithm, selecting the computational architecture, as well as determining various parameters such as the amount of parallelism, pipelining scheme, memory design, and data and computation partitioning scheme demands a systematic method to evaluate various trade-offs to facilitate architectural level exploration and optimization, such as designing parallel data paths and memory structures, and choosing programmable granularity. This is an important step in the design process in order to make a better use of available resources and to realize the potential of advanced communication theoretical results, which require high computational power.

1.2.2 Application of Co-exploration for Building MP Core

The advantages of employing the cross-cutting method that brings together system and algorithm, architecture, and circuit design are demonstrated by the example of building a parameterized wireless channel estimator. The matching pursuit (MP) algorithm has been redesigned to reduce the complexity while maintaining the estima-

tion accuracy. In hardware implementation, the trade-offs in terms of channel estimation accuracy, the effect of fixed-point and floating-point data representation, and the distribution of the MP data due to different architectural implementations, have been evaluated to achieve high performance and low energy consumption. Through design space exploration, the final architecture is mapped onto a modern Virtex-II FPGA chip, and the resulting synthesizable MP core can execute over 216 times faster than a high performance microprocessor.

1.2.3 Co-exploration for Energy Efficiency

With the consideration that power is becoming an increasingly important factor in digital design, and leakage power, in particular, will soon become the dominant source of power dissipation, algorithm and architecture design techniques are exploited to tackle the leakage problem through memories, which are the major targets for battling leakage problem. Through investigating the leakage problem in caches of general microprocessors, an optimal policy based on the perfect knowledge of the address trace information has been proposed to achieve maximal leakage savings. And by incorporating leakage-aware data placement of data within embedded memories, significant leakage power savings can be accomplished. Different low power design techniques, including both dynamic power reduction techniques and leakage power reduction techniques are also explored in building the energy efficient MP core.

1.3 Dissertation Organization

This dissertation is organized into six chapters.

Chapter 2 presents the algorithm and architecture co-exploration, which includes the characterization of algorithm properties, the evaluation of architectures and low power design techniques.

Chapter 3 employs the algorithm / architecture co-exploration to the design example of building an efficient wireless channel estimator — Matching Pursuit core. Different algorithms and architectures are explored to reduce power consumption of the functional elements and interconnections.

Chapter 4 studies the leakage power minimization techniques. By exploring the temporal information to tackle the leakage problem in caches on general purpose microprocessors, a generalized model for optimizing cache leakage based on the perfect knowledge of the address trace is proposed to achieve maximal leakage power savings.

Based on the generalized model proposed in Chapter 4, Chapter 5 investigates leakage-aware placement methods to incorporate spatial information to further reduce leakage power of embedded memories on reconfigurable FPGA devices. MP core is further explored to study the effectiveness of the different schemes.

Chapter 6 concludes with a summary and suggestions for future work.

Chapter 2

Algorithm/Architecture Design Space Co-exploration

2.1 Introduction

There often exist various digital signal processing algorithms for solving a specific mathematical problem, but they exhibit vastly different characteristics and thus may lead to widely different implementation costs. For the same algorithmic specification a broad range of computational architectures, ranging from general-purpose microprocessor to dedicated hardware can be used, resulting in several orders of magnitude difference in performance and power consumption. Among different design parameters, the amount of parallelism, the degree of time-sharing of the processing elements and their structure, as well as various parameters such as word-lengths, and clock frequency need to be determined. The multidimensional design space offers a large range of possible trade-offs and since decisions made at each step are tightly coupled with other choices, an integrated and systematic design approach across the

algorithm, architecture, and circuit levels is essential to optimize the implementation for given design criteria.

Researches have shown that the most dramatic power reduction stems from optimization at the highest levels of the design hierarchy. In particular, case studies indicate that high level decisions regarding selection and optimization of algorithms and architectures can improve design metrics by an order of magnitude or more, while gate and circuit level optimizations typically offer a factor of two or less improvement. And the design time involved at high levels is typically much shorter than lower levels. It is thus important to be able to analyze the possibilities and trade-offs of high-level decisions, before investing effort in exploration at lower levels. This suggests that architectural implementation estimation mechanisms are needed to facilitate rapid high-level design exploration, especially when the goal is to achieve an area efficient, low-power integrated circuit implementation. Specifically, optimization efforts should begin at the algorithm/architecture level, which is the focus of this chapter.

2.2 Design Space Exploration

At algorithm development stage, algorithm performance has often been treated as the single most important design criterion to evaluate and choose among possible algorithms, while complexity is considered very roughly. For example, floating-point operation count is often used as a complexity measure, which is not sufficient for accurate estimates of power consumption and area. However, design decisions made at

algorithm level affect the implementation significantly. Even for a given algorithm, many optimization techniques can be applied to, for example, parallelizing the algorithm can increase speed by exploring various resources effectively. Some of these techniques are common to all architectures and some are architecture specific. Moreover, although IC technology has improved to the point that complex wireless systems can now be implemented for portable devices, with ever more algorithmic complexity, it is not possible for all algorithms to integrate now or even in the near future, since it is unlikely that the improvement in technology will surpass the historical gains of Moore's law. This complexity limit comes from the energy and silicon area constraints of typical portable applications.

Therefore, a method is needed to assist algorithm exploration, which determines the feasibility of algorithms for implementation and gives guidance to algorithm developers about the present and future capabilities of IC technology. This method should evaluate algorithms in terms of not only the traditional measures of performance issues such as throughput, but also the energy requirement.

For a given algorithm, a wide range of computational architecture can be selected. The difficulty in achieving high energy-efficiency in a programmable processor stems from the inherent costs of flexibility. Programmability requires generalized computation, storage, and communication structures, which can be used to implement different algorithms. Efficiency, on the other hand, dictates the use of dedicated structures that can fully exploit the properties of a given algorithm. While conventional pro-

grammable architectures, e.g., general purpose microprocessors, can be programmed to perform virtually any computational task, they achieve this flexibility by incurring the significant overhead of fetching, decoding, and performing computations on a general-purpose data-path under the control of an instruction stream, which most often dominates the energy dissipation.

The flexibility of programmable processors is highly desirable for handling general computing tasks. Wireless communication algorithms, on the other hand, have many distinguishable intrinsic properties that make them more amenable to efficient implementations and they do not require the full flexibility of a general-purpose device. These algorithms are data-path intensive, which encourages many digital signal processing optimization techniques; exhibit high degrees of concurrency, which makes parallel processing possible; and are dominated by a few regular computations that are responsible for a large fraction of execution time and energy, which suggests the use of highly optimized and parameterized components.

There is a wide spectrum of application specific architectures, which sit at different positions in the trade-off between efficiency and flexibility. While dedicated implementation and general-purpose microprocessors are at the two extremes respectively, many domain specific processors fill in the gap between them. Consequently, a method is needed to evaluate architectures to facilitate architectural level exploration and optimization, such as designing parallel data paths and memory structures, and choosing programmable granularity. This is an important step in the design process in order to

make a better use of available IC technology and to realize the potential of advanced communication theoretical results, which usually require even more processing.

To summarize, high-level design decisions include the choice of algorithm for a given application specification, the selection of the computational architecture and the determination of various implementation parameters. Because of the size and complexity of the multi-dimensional design space, the designer often cannot adequately explore many of these possibilities, which offer a large range of trade-offs. Therefore, there is a need for a design method for high-level algorithm and architecture exploration. in a more systematic way to achieve an efficient and low-power implementation.

The subsequent sections describe the main components of the algorithm / architecture design space co-exploration: algorithm and architecture level design characterization, design metric identification, and optimization techniques (especially for low-power designs).

2.3 Algorithm Exploration

Much research has been done towards automated behavioral-level synthesis, optimization, and performance estimation, with the goal of reducing high-level design and exploration time. However, they are often not sufficient. At the high level of abstraction, the degrees of design freedom are often so great as to make full analysis of design trade-offs impractical. Comprehensive analysis of designs is often prohibitively

time-consuming, as is developing a sufficient understanding of the various design options and optimization techniques. As a result, designers often search the design space in an ad-hoc manner and decisions are made subjectively and prematurely, based on hand-waving or partial analysis as well as factors such as convenience and familiarity.

In this design method, algorithms are evaluated at an early stage based on extraction of critical characteristics. A key step in realizing the potential advantages attained by using high-level optimization involves developing a better understanding of the correspondence between algorithms and various architectures. The design characteristics that are most directly related to the quality of algorithm-architecture mappings are identified and used to give valuable hints on how to improve implementation efficiency as well as the selection. For example, some algorithms have much higher computational complexity requirements compared to others without performance gain, thus they can be eliminated regardless of architecture choice for low-power implementation.

The underlying idea behind this property-based algorithm characterization is that a design can be characterized by a small set of relevant, measurable property metrics which are related to potential design metrics and can be used to provide guidance during the optimization of the design. Rather than dealing with the design in its entirety, the property metrics provide a simpler and more manageable representation.

Theoretical capacity and algorithm performance, such as convergence speed and stability of an algorithm, are evaluated through closed form analytical solution and/or

simulations. They are well studied by the information theory and signal processing community. The focus here is on implementation complexity evaluation.

Numerical Properties

Numerical properties indicate how the algorithm responds to various noise sources, non-idealities and errors and the resulting performance degradation, i.e., the robustness of an algorithm.

1. Word-length. This includes the dynamic range (the ratio between the largest and smallest numbers) and precision requirements, which lead to the choice of fixed-point, or floating-point implementation. With fixed-point arithmetic, numbers are represented as integers or as fractions in a fixed range, usually from -1 to 1. With floating-point arithmetic, values are represented by a mantissa and an exponent as (mantissa x $2^{exponent}$). The mantissa is generally a fraction in the range of -1 to 1, while the exponent is an integer that represents the number of places that the binary point must be shifted left or right in order to obtain the value represented. Obviously, floating-point representation can support wider dynamic range at the expense of increased implementation area and power consumption. Finite precision requirements not only affect both area and power of digital processing, it also directly relates to the requirement for A to D converter. This is evaluated typically through noise analysis and fixed-point simulations.

The word-length is typically chosen such that the performance degradation due to finite precision is small or tolerable compared to infinite precision.

2. Sensitivity to parameters such as noise levels. These properties are evaluated through complete system end-to-end simulations including the modeling of various non-idealities.

Size

The metric of size quantifies the amount of computation in terms of the total number of operations and data transfers executed in the computation. Intuitively, the larger the computation, the more power consumption and longer delay one expects. Approximately, the number of operations of each type (e.g., addition, shift, multiplication, read/write) and word-lengths for each operation type gives an estimation of the required hardware resources and power consumption. The number of multiplication accumulations (MACs), in particular, is a common metric used in comparing DSP algorithms. And the number of array write and read accesses is linked to the amount of memory accesses and storage needed.

The basic types of operations can be obtained by algorithm profiling and can be used together with their energy efficiency to determine the lower bound of power consumption for an algorithm. This lower bound not only provides an implementation feasibility measure to algorithm developers, but also provides a comparison base to evaluate any architectural implementation.

Regularity

Structured and regular designs are highly desirable for complex VLSI systems. These properties lead to a cell-based design approach where complex systems are constructed through repeated use of a small number of basic building blocks or primitives, resulting in modular and extendable designs.

Locality

The concept of locality has been heavily studied and utilized in designs such as memory hierarchies. Locality is the qualitative property of typical DSP programs that 80% of execution time is spent on 20% of the instructions. In the computer architecture domain, temporal locality describes the tendency for a program to reuse data or instructions that have been recently used and spatial locality describes the tendency for a program to use data or instructions neighboring those recently used. In the VLSI signal processing domain, temporal locality characterizes the persistence of the computation's variables (a computation is temporally local if its variables have short predicted lifetimes.) and spatial locality measures the communication patterns between blocks and indicates the degree to which a computation has natural isolated clusters of operations with few interconnections between them. Identification of spatially local sub-structures can be used to guide hardware partitioning and minimize global communications, which can lead to smaller area, lower global communications, and thus lower interconnect power.

Concurrency

Another very important structure is an algorithm's inherent concurrency, which measures the number of operations that can be executed concurrently. Concurrency has been studied extensively and its application has resulted in greatly improved performance in terms of metrics such as throughput and latency. Concurrency includes explicit parallelism, which is revealed by data dependency graph or a block diagram based description, and potential parallelism achievable through pipelining and chaining, which depends on the internal implementation structure of a block. While this inherent parallelism cannot be fully exploited by all architectures, it is an important factor to guide architecture exploration.

Timing

The timing metrics assume that each operator has an associated delay of execution. The exploration finally leads to the timing specification for each module. The timing metrics, shown as the following, give a measure of how constrained the design is.

1. Slacks for each operation. They give an idea of how much flexibility the scheduler has for time-sharing architectures or how much the performance can be reduced for energy consideration.

2. Critical path. This determines the time it takes to complete the computation.

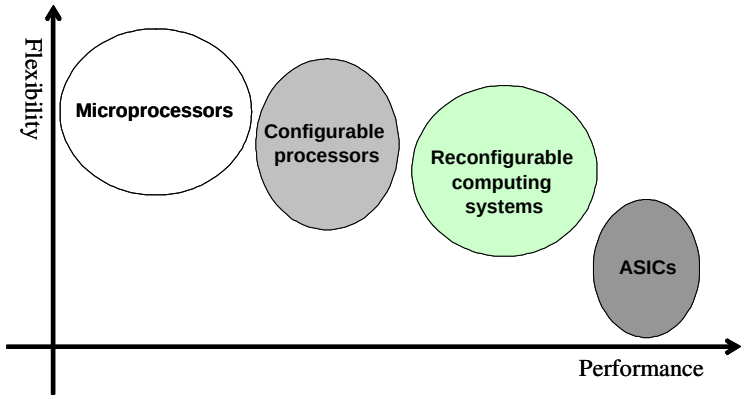


Figure 2.1: Computer system architecture design space.

3. Latency. This is the delay from the arrival of a set of input samples to the production of the corresponding output.

2.4 Architecture Exploration

Given an algorithmic specification, a designer is faced with selecting the computational architecture, as well as determining various parameters such as the amount of parallelism, and the pipelining scheme. Since it requires too much effort and is too time-consuming to complete all design choices and make final comparisons, fast high-level evaluation is needed for architectural exploration based on a set of design metrics for fair comparisons among different implementations.

2.4.1 Computing System Architectures

As Moore's law marches on, the number of transistors per die has been doubled every 18 months. To utilize the huge transistor real estate and fulfill the requirements of the applications, there are four primary architectures for the execution of algorithms, namely general purpose microprocessors, configurable processors, application specific integrated circuit, and reconfigurable devices. Two parameters are used to categorize architectures: the granularity of processing elements and the performance. Granularity gives a measure of the flexibility an architecture can provide: the larger the granularity, the less the flexibility. Figure 2.1 depicts the relative positions of different architectures in this two-dimensional space.

Microprocessors

Software-programmed general purpose microprocessors execute a set of instructions to perform a computation. By changing the software instructions, the functionality of the system can be achieved. However, the downside of this flexibility is that the performance can suffer. The processor must read each instruction from memory, decode its meaning, and only then execute it. This leads to a high execution overhead for each individual operation. Additionally, the set of instructions that may be used by a program is determined at the fabrication time of the processor. Any other operations that are to be implemented must be built out of existing instructions.

Configurable Processors

A configurable processor is one that can be modified or extended to address specific design issues by changing the processor's feature set. Developers can add their product's differentiating "secret sauce" to perform a task much faster, in a much smaller area or with less power consumption. Configurability and extendibility are two distinct benefits of configurable processors. Configurability lets the designer change the processor's predefined architectural framework to meet design requirements. Examples of configurability include altering cache sizes or the number of registers in a register file, or deciding whether to include a multiplier or barrel shifter. Extendibility means that additions can be made to the processor. However, with the middle-of-the-road approach, there is no limit on what designers can do to the processor, such as changing the bus structure or the bus interfaces, or extending the architecture to perform special tasks never contemplated by the original processor architects.

Application Specific Integrated Circuits

Another method is to construct an application specific integrated circuit (ASIC) or a group of hard-wired individual components to perform computations in hardware. ASICs are designed specifically to perform a given computation, and thus they are very fast and efficient when executing the exact computation for their desired design. Yet, the circuit cannot be altered after fabrication. This forces a redesign and re-fabrication of the chip if any part of its circuit requires modification. This is a very costly process,

especially when one considers the difficulties in frequently replacing ASICs in a large number of deployed embedded-systems, e.g. sensor networks.

Reconfigurable Devices

Reconfigurable computing promises a trade-off between performance and flexibility, and enables higher productivity [19, 22]. Reconfigurable devices, including field-programmable gate arrays (FPGAs), contain an array of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, known as logic blocks, are connected using a set of routing resources that are also programmable. In this way, custom digital circuits can be mapped to the reconfigurable hardware by computing the logic functions of the circuit within the logic blocks, and using the configurable routing to connect the blocks together to form the necessary circuits. The functionality of the logic blocks and the connections in the interconnection network can be modified by downloading bits of configuration data onto the hardware. Currently, hybrid architectures integrate programmable logic and interconnect together with more components, such as microprocessors, digital signal processors, ASICs, and memory [110]. The reconfigurable device is similar to the ASIC in the sense that the computation is done spatially, and can be tailored to the specific application. It is also similar to the general-purpose processor in the aspect of silicon re-usability. The fundamental differences between reconfigurable logic and traditional microprocessors include:

1. *Spatial Computation.* The computation is processed by spatially distributing the computations rather than temporally sequencing through a shared computational unit.

2. *Configurable Datapath.* The functionality of the computational units and the interconnection work can be adapted at run-time by using a configurable mechanism.

3. *Distributed Resources.* The required resources for computation, such as computational units and memory, are distributed throughout the device instead of being localized in a single location.

4. *Distributed Control.* The computational units process data based on local configuration units rather than an instruction broadcast to all the function units.

With the above characteristics, FPGA and reconfigurable computing is suitable to accelerate a variety of applications that exhibit fine grain and coarse grain parallelism, such as image processing, telecommunication, bioinformatics, data encryption, etc. Exploiting this parallelism provides significant performance advantages compared to its microprocessor counterpart. Complex functions can be mapped onto the architecture achieving higher silicon utilization and reducing the instruction fetch and execution bottleneck.

2.4.2 Design Quality Metrics

During the past four decades, the nature of the IC designs has slowly changed; the continued scaling of the underlying technology has moved designs from being limited by the functionality on a chip, to being power, performance and area efficient. In this section, we describe the quality metrics commonly used to compare different architectures, namely performance (the time required to accomplish a defined task), power consumption and cost. Many factors have direct impact on the cost, such as design time (including that for both software and hardware development), integrated circuit fabrication and testing cost, packaging, and quantity. In this thesis, cost is simply evaluated as silicon area.

Power

The continuing decrease in feature size and the corresponding increase in chip density and operating frequency have made power consumption a major concern in VLSI circuit design. Modern microprocessors are hot: current high-performance processors have reached 150W and it will reach 250W in 2013. Excessive power dissipation in integrated circuits not only discourages their use in a portable environment, but also causes overheating, which degrades performance and reduces chip lifetime. To control their temperature levels, high power chips require specialized and costly packaging and heat-sink arrangements. This, combined with the recently growing demand for low-power portable communications and computing systems, has created a need to limit the

power consumption in many designs. Indeed, the Semiconductor Industry Association has identified low-power design techniques as a critical technological need.

Performance

Due to the demanding need of high speed for a variety of applications, performance has always been one of the most important metrics to characterize the system design quality. Researchers have proposed different techniques to achieve high performance for microprocessors, ASICs, and reconfigurable computing devices, for instance, branch prediction, caches, pipelining, memory prefetching, multi-threading, speculative execution, parallelism, etc.

Performance metrics can be divided into communication and computation metrics. Communication metrics are related to the time spent by a behavior in interacting with the other behavior in the system through an abstract channel. The channel transfer rates directly affect the execution times of two behaviors communicating over the channel. Computation metrics measure the time required to perform the computations within a behavior. Generally two types of units are used as computation metrics: clock cycles, and execution time. The choice of a clock cycle could affect the execution time and the resources required to implement the design. The execution time of a behavior in the design is defined as the average time required by the behavior from start to finish. For a design with pipelined implementation to improve system throughput, the execution

time for a pipeline is the total elapsed time between the arrival of data to the pipeline and the generation of its corresponding results.

Area

The area metric is directly related to the hardware cost. It also decides whether certain portion of the design can fit into a given chip area. For example, in semi-custom (gate array) and programmable (FPGA) technologies, the number of gates or the number of combinational logic blocks on any one chip are limited. The area cost will enable the system designer to determine how many gates or CLBs will be required for implementing the design. If a design is estimated to have an area larger than the maximum allowed chip area. The design will have to be partitioned and implemented as two or more chips. Moreover, smaller area indicates shorter wires in the interconnect network, which will achieve higher performance and less interconnection power consumption.

These three metrics are not orthogonal and one can be traded for another. For example, performance can be improved with hardware duplication. Moreover, energy consumption, which determines battery life, is usually more important than power consumption, which determines the packaging and cooling requirements. For example, an architecture that can execute a given application quickly and then enter a power-saving mode may consume less energy for the particular application than other architectures with lower power consumption.

Other Metrics

There are several other quality metrics which are also applicable to system design, for instance, design for testability, design for reliability, design for manufacturing, design time, and time-to-market, and so on. Design for testability produces a design with a minimal test cost, which is essentially a trade-off between built-in test hardware and the cost of testing after the design has been implemented. Design for reliability includes redundant components to enhance the reliability and get the correct functionality, especially when the technology scales down to below 100nm. Design for manufacturing is a process in which manufacturing inputs is used at every stage of design in order to design parts and products that can be produced more easily and economically. Design time is defined as the time required to obtain an implementation from the functional specification. The abstraction level at which design automation is introduced in the methodology will greatly impact the design time. Capturing the system using a high-level language and using high-level synthesis, logic synthesis, and physical synthesis tools will require significantly less time than manually converting a functionally specification to transistor schematics. Also, design time can be reduced by using more programmable and off-the-shelf customized components. Time-to-market can be defined as the total time elapsed from design conceptualization to the actual delivery of the product. In addition to design time, it includes the time for performing specification, fabrication, testing, and marketing.

2.5 Low Power Design Techniques

Motivated by emerging battery-operated applications that demand intensive computation in portable devices, techniques have been investigated to reduce energy consumption in CMOS digital circuits, or equivalently power consumption while maintaining computational efficiency.

2.5.1 Source of Power Dissipation

The power consumed in a digital system is the sum of dynamic, static and short-circuit power.

$$P_{total} = P_{switching} + P_{leakage} + P_{short-circuit} = \alpha C_{eff} V_{dd}^2 f + I_{leakage} V_{dd} + I_{SC} V_{dd} \quad (2.1)$$

The first term represents the switching component associated with the charging and discharging of the parasitic capacitance, where α is the activity factor (or the probability that a power-consuming transition occurs), C_{eff} is the average switched capacitance per cycle, f is the clock frequency, and V_{dd} is the supply voltage. The second term, $I_{leakage}$, arises from substrate injection and sub-threshold effects, which is primarily determined by fabrication technology. The third term is due to the direct-path short circuit current I_{SC}, which arises when both the NMOS and PMOS transistors are simultaneously active during signal transitions.

Dynamic power dissipation in system components refers to the dissipation of energy caused by the charging and discharging of load capacitances during the switch-

ing of CMOS circuits. When the technology generation goes below 100nm, leakage power consumption becomes the dominant issue. The problem of leakage stems from the need for a trade-off between dynamic power and performance. One of the most effective ways of reducing the dynamic energy dissipation is to scale down the transistor supply voltage. To maintain high switching speed under reduced voltages, the threshold voltage must be also scaled down accordingly. As the threshold voltage drops, it is easier for current to leak through the transistor resulting in significant leakage energy dissipation. New technologies targeted at reducing dynamic power and increasing performance, such as low threshold voltage and gate oxide scaling, further increase the relative importance of leakage power.

2.5.2 Algorithm Level Optimization

Many low-power design techniques can be applied at multi-levels to vastly reduce the energy consumption. At algorithm level, an optimization may involve performing algorithm modifications, selecting new arithmetic operations, or applying transformations. Transformations involve changes in the structure of an algorithm without changing the input-output behavior, while the number and type of computational modules, their interconnection and their sequence of operation are optimized. It is often possible through the exploration of concurrency of an algorithm including code hoisting, functional pipelining, loop restructuring (e.g., loop unfolding and software pipelining), and so forth. Transformations usually reduce the total number of operations required,

replace some operations with more convenient ones, i.e., strength reduction depending on the cost functions of operation types on a particular architecture (such as constant propagation and replacement of constant multiplications with additions and shifts), optimize the resource usage, and reduce the word-length required for data representation.

2.5.3 Architecture Level Optimization

At the architecture level, the main energy reduction approaches are to optimize fundamental components, which include computation, communication, and global variable storage, and to reduce overhead components such as control, interconnect, and temporary variable storage.

Dropping the supply voltage can greatly reduce the energy consumption of a circuit, but at the cost of increased delay time and thus slower circuits. To compensate for this degradation in circuit speed at architecture level, it is necessary to exploit concurrency in the form of pipelining and parallelism.

Thus, an important design objective for an energy efficient architecture is the ability to execute concurrent computations. This can be accomplished in a number of ways depending on the computational architecture. One approach, at the maximum flexibility end of the architecture spectrum, is to use a number of software programmed processors. The overhead of splitting the algorithm into a number of processors can become prohibitive if a large number of processors are required (e.g. > 10). Another approach is to use dedicated hardware. For example, a direct mapping strategy maximizes the

hardware parallelism by directly implementing the data flow graph of the algorithm onto silicon. While this method is the most energy efficient, because it removes the overhead associated with programmability, it is the least flexible. A prerequisite to optimizing a program for low power is to design an algorithm that maps well to the given processor architecture, exploiting the performance features and parallel processing capacity of the hardware as much as possible. Maximizing performance also gives increased latitude to apply other power optimization techniques such as reducing clock rate and shutting off idle components through sleep and drowsy modes.

The number of bits used in the arithmetic strongly affects all key parameters of a design. It is desirable to minimize the number of bits for energy and area optimizations while maintaining sufficient algorithmic accuracy and performance.

2.6 Summary

Often different algorithms and architectures are available for the same wireless communication application, but they achieve quite different system performance and have quite different complexities that lead to vastly different implementation costs. An algorithm/architecture co-exploration method is studied, allowing for a detailed assessment of key performance measures (e.g., bit error rate) and implementation parameters (i.e., performance, area, and power consumption) at an early stage of the design process. Consequently system designers can explore and evaluate the performance of algorithms as well as their suitability for highly integrated CMOS implementations.

Chapter 3

Case Study — MP Core: An Efficient Wireless Channel Estimator

3.1 Introduction

With the vision that ubiquitous connectivity is quickly becoming a reality due in large part to the rapid development and deployment of wireless networks, reconfigurable devices will certainly play an important role in this new era. There are many computational challenging problems to be solved in this domain, and the extreme time-to-market and rapidly shifting protocols and standards make this an area ripe for a reconfigurable solution. State of the art wireless devices are optimized for dealing with a few low bandwidth users in a relatively unobstructed environment. When many users attempt to access the network at once, or when there is significant distortion due to multiple transmission paths, these systems either slow to a crawl or even simply cease to function [108]. Exacerbating these problems is the fact that users demand not only more extensive connectivity, but also increased bandwidth and additional features such

as position information. To address these problems, intelligent wireless systems must be developed that can mitigate and even exploit the existence of multiple transmission paths.

To meet the needs of high performance with low energy consumption for supporting ever increasing bandwidths demands and increased connectivity from multiple users, performance and energy efficient implementations are required and optimized at all levels of wireless system design, from protocols to signal processing and from system level design to physical design. At the heart of the next generation wireless devices is the ability to accurately estimate characteristics of the channel. Once these characteristics are determined, the device can correct for them to enable more simultaneous users, higher bandwidth, and lower power communications. In this chapter, we focus on investigating and applying optimization techniques in the algorithmic, architectural and bit levels to implement an energy efficient channel estimator in a modern reconfigurable device with high performance.

Channel estimation is a fundamental problem in communication systems with the goal of characterizing the media over which communication is propagating [81]. Wireless communication channels typically contain multiple paths due to scattering effects, and thus the received signal is composed of many delayed and attenuated versions of the transmitted signal. If, for example, a user is trying to connect to an access point located in an office down the hall, there will typically be many transmission paths, including a highly attenuated direct path and other multiple-reflection paths.

The received signals from multiple paths may be either destructive or constructive. When there is destructive interference, the signal may be corrupted. Not only does unmitigated multipath interference result in decreased data rate, but other applications such as accurate radiolocation become almost impossible. Rayleigh fading [81] due to multipath propagation will cause catastrophic failure in received signal strength-based radiolocation. For accurate radiolocation, time-of-arrival (TOA)-based methods are much more promising, but accurate estimation of the direct path arrival time is required [57]. Thus, efficient implementation of an algorithm for accurate estimation of dynamic multipath channels is required. Given estimates of the channel parameters, signal corruption due to multipath propagation can be easily reversed, and the signals due to multiple paths can be combined coherently for increased noise immunity, resistance to signal loss due to shadowing, and overall improvements in data rate and bit/frame error rates.

The overriding trend among the modern wireless communication systems is that higher data rates and bandwidth requires increasingly complicated physical and data link layer approaches [108]. As such, more computational power is required from the hardware. In order to achieve high data rates using these complicated transmission techniques, we must enable efficient and flexible signal processing devices starting with channel estimation algorithms. Unfortunately, little work has been done on the hardware implementation. Rajaopal [86] presented a multiprocessor implementation of a multiuser channel estimator, which includes dual DSPs to speed up the algorithm

and incorporated FPGAs to accelerate parts of channel estimation algorithms. While there have been many theoretically-sound approaches proposed for multipath channel estimation and multiuser detection [16, 44, 57, 108], these approaches have not yet been adopted by hardware designers because of the complexity of the algorithms involved and the cost associated with realizing them in an actual implementation.

In order to realize high bandwidth wireless communication schemes, we must develop tools and methods for efficient multiuser, multipath channel estimation. To make the leap from theory to reality an efficient and flexible high performance platform is required. Reconfigurable systems offer the necessary balance between flexibility and performance by allowing the device to be configured to the algorithm at hand [48]. Reconfigurable systems allow for the post-fabrication programmability of software with the spatial computational style most commonly employed in hardware designs and are becoming an attractive option for implementing signal processing applications [6, 26, 61, 86] because of their high processing power and customizability. The inclusion of new features in the FPGA fabrics, such as a large number of embedded multipliers, microprocessor cores, on-chip distributed memories, adds to this attractiveness. One such example is software-defined radio (SDR) [21], which attempts to provide an efficient and inexpensive mechanism for the production of multi-mode, multi-band, and multifunctional wireless devices. The performance and flexibility of reconfigurable devices make them viable and ideal for implementing the SDR systems.

In this chapter we present a parameterized design for a small, high throughput, channel estimation engine that can effectively handle a large number of multiple transmission paths. The result of our research is a synthesizeable IP-core on FPGA that can be quickly tuned to the requirements of its application and then instantiated in any number of wireless devices. We begin with the matching pursuit (MP) algorithm [20, 57] for DS-CDMA signals, which is able to achieve accurate channel estimation with reasonable complexity, and which can be combined with the GSIC algorithm for efficient multiuser detection [57]. By redesigning the MP algorithm, we are able to achieve a substantial increase in efficiency with zero decrease in estimation accuracy. In fact, the key to our design is a cross-cutting method that has resulted in novel optimizations at every level, from the theory and algorithms to the arithmetic and placement.

We describe our design and quantify the trade-offs in terms of channel estimation accuracy, performance, area, and power of the implementation. We explore the possibilities of extracting parallelism from the MP formulation, the effect of fixed and floating point on both area and error, the benefit seen from deep pipelining and dedicated hardware units on a reconfigurable device, and the advantage of exploring low power design techniques. Based on the design space exploration, our final architecture is mapped onto a Virtex-II XC2V3000 FPGA, resulting in a speedup of over 216 times compared with the execution time on a high performance desktop machine and power saving of 25.4% with clock gating technique. Our synthesisable MP core meets the

needs for high-bandwidth, reliable communication and radiolocation for diverse applications in public safety (e.g. search and rescue), environmental monitoring, ubiquitous computing, and homeland security. Furthermore, it can also be employed to increase the capacity and coverage of wireless networks in environments which are rich with transmission paths such as in large buildings and urban cityscapes.

The rest of the chapter is organized as follows. In section 3.2, we discuss a model for multipath propagation, present a new formulation of the MP algorithm, and compare it with other channel estimation algorithms. In section 3.4, different design tradeoffs are discussed for the design space exploration of the parameterized MP core implementation. We discuss related work in section 3.7 and draw conclusions in section 3.8.

3.2 Matching Pursuit Algorithm

3.2.1 Multipath Channel Propagation

Wireless communication channels typically contain multiple paths due to scattering effects, and thus the received signal is composed of many delayed and attenuated versions of the transmitted signal. For outdoor communications, the scatterers may be buildings, mountains, etc., while for indoor communications, the scatterers may be walls, furniture, etc. Path lengths may vary greatly. In this work, the multipath propagation delays are assumed to be less than the symbol duration, a reasonable assumption in most cases.

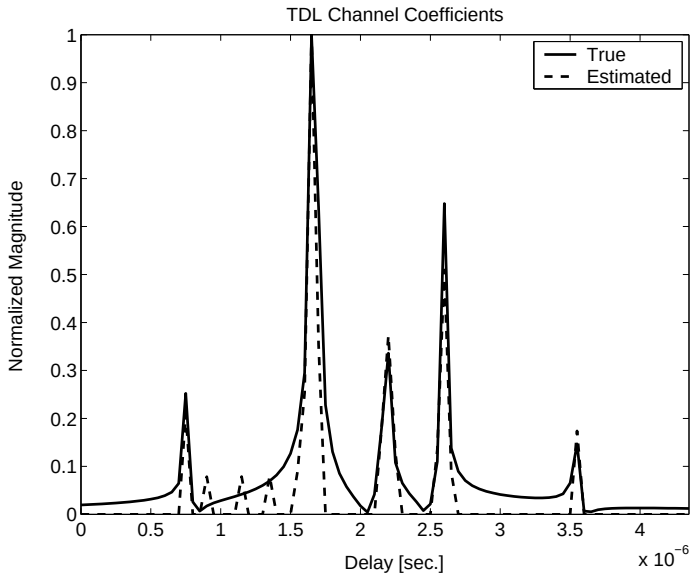


Figure 3.1: Multipath channel estimation with MP.

The multipath channel with continuous-valued delays can be represented by a tapped-delay-line (TDL) filter with discrete-valued delays iT_s, for $i = 0, 1, \ldots, N_s - 1$, where $1/T_s$ is the Nyquist sampling rate (twice the chip rate), and N_s is the number of samples per symbol duration [108]. Associated with each TDL path i is a complex-valued channel coefficient f_i. The f_i are given by the sum of sampled sinc$(.)$ functions centered at the true delays τ [31, 57]. A *sparse* channel is one in which $N_f << N_s$ channel coefficients have non-negligible magnitude. The TDL representation of an example 5-path channel is shown in Figure 3.1 (solid line).

The received signal after RF-to-baseband down-conversion and A/D sampling is denoted by

$$r = Sf + n \in C^{MN_s * 1}, \tag{3.1}$$

where M is the number of training symbols, n is the sampled additive white Gaussian noise vector, $f = [f_0, f_1, \cdots, f_{N_s-1}]^T \in C^{N_s*1}$ is the channel coefficient vector, and $S \in \Re^{MN_s*N_s}$ is the characteristic signal matrix. $\Re$ and C represent real and complex numbers, respectively, and $(.)^T$ denotes the transpose operation. The ith column S_i of S is the received signal due to path i if $f_i = 1$, and in general $f_i S_i$ is the received signal due to path i. S is given in [57] and is known *a priori*, since it depends only on the DS-CDMA spreading sequence and the transmit and receive filters. Referring to the received signal model (3.1), the multipath channel estimation problem is that of computing an estimate $\hat{f}$ of f, given S and the received signal vector r containing noise n.

3.2.2 Redesigning MP Algorithm

The Matching Pursuit (MP) channel estimation algorithm [20, 57] algorithm provides a low complexity approximation to the Maximum Likelihood (ML) [44, 108] solution for sparse channels, i.e., under the constraint that only N_f elements in $\hat{f}$ are non-zero: $\left|\hat{f}\right| = N_f$. The exact ML solution under the sparse channel constraint is given by

$$\hat{f} = \underset{f \in A_{N_f}}{\arg\min} \left\{ \|r - Sf\|^2 \right\}, \tag{3.2}$$

where $A_{N_f} = \{f : |f| = N_f\}$. Note that if the channel is not sparse, $N_f = N_s$ and estimates are computed for all elements in f. Since the channel estimation cost function

minimized in (3.2) is non-convex, an exhaustive search is required. The complexity (in terms of the number of scalar multiplications) of the optimal ML algorithm is shown in Table 3.1, where the binomial coefficient $C_{N_f}^{N_s} = (N_s!)/(N_f!(N_s - N_f)!)$ and $2/Q$ is the precision of the channel coefficient estimates. Clearly, real-time implementation of ML channel estimation is infeasible. By contrast, the MP algorithm [57] is highly efficient.

The algorithm implemented in the Matching Pursuit IP core has been redesigned for a speed improvement of N_f times, with zero reduction in channel estimation accuracy. The new fast MP algorithm is obtained by posing the ML estimation problem in terms of sufficient statistics, as follows.

$$-\|r - Sf\|^2 \;\propto\; 2\mathrm{Re}\left\{(V^0)^H f\right\} - f^H A f \tag{3.3}$$

is a sufficient statistic for signal parameter estimation and data symbol detection [108], where $V^0 = S^T r \in C^{N_s * 1}$ and $A = S^T S \in \Re^{N_s * N_s}$. S is known a priori, as mentioned in Section 3.2.1, and therefore S and A are pre-computed once for all time and stored in memory. The computation of V^0 can be parallelized as N_s vector inner products (correlations) $V_i^0 = S_i^T r$. Since the columns S_i of S are generated as filtered circular-shifted versions of the transmitted DS-CDMA spreading sequence, the computation of V^0 is equivalent to matched filtering the received signal r.

MP maximizes (3.3) iteratively, one channel coefficient $\hat{f}_{q_j}$ at a time, using a greedy algorithm in which q_j and $\hat{f}_{q_j}$ are selected such that the increase in (3.3) at each stage

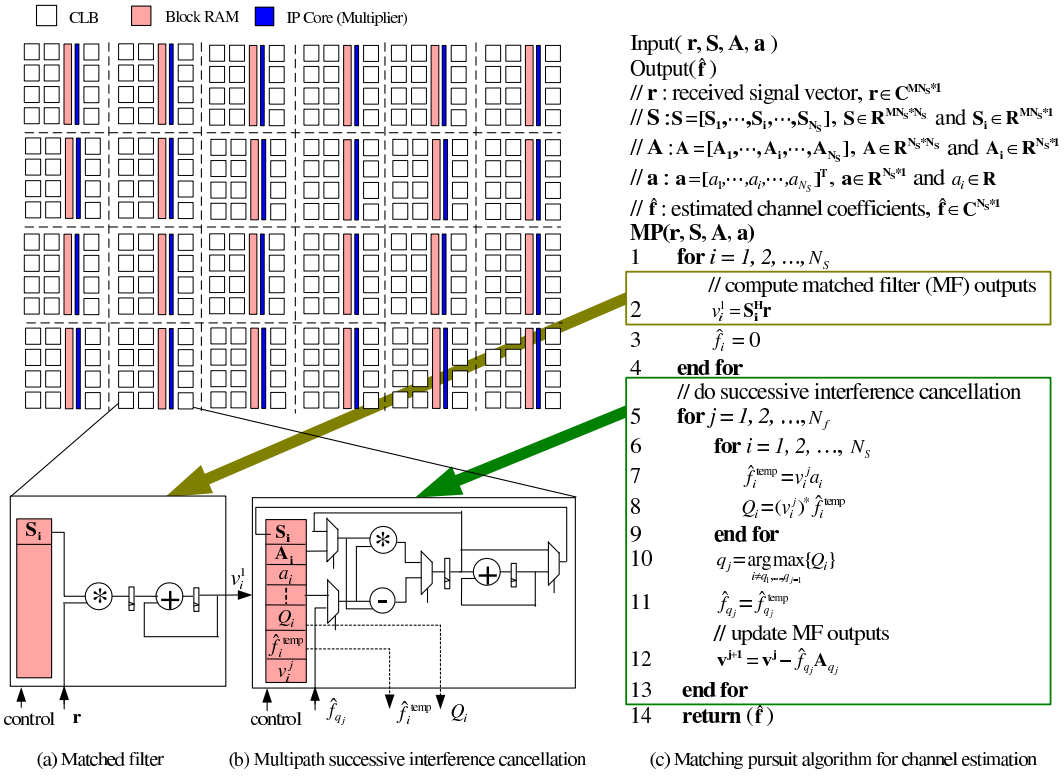


Figure 3.2: Mapping of the MP algorithm on a modern FPGA. The matched filter (a) and multipath successive interference cancellation (b) are distributed across the FPGA to parallelize the MP algorithm (c).

Sparse ML	MP	Fast MP
$O(MN_S C_{N_f}^{N_s} Q^{2(N_s-N_f)})$	$O(2N_f MN_S{}^2)$	$O(2MN_S{}^2)$

Table 3.1: Complexity comparison for sparse multipath channel estimation algorithms. M = number of training symbols, N_s = number of samples per symbol, N_f = number of non-zero channel coefficients, $2/Q$ = channel coefficient precision.

j is the largest possible. That is, the multipath signal components are estimated via successive interference cancellation. The algorithm is summarized in Figure 3.2(c). Note that the "hat" symbol on f is omitted for convenience. To eliminate the need for division operations, the vector a, with $a_k = 1/A(k,k)$ and $A(k,k)$ denoting the kth diagonal element in A, is pre-computed once for all time and stored in memory. The storage of A and a in memory corresponds to the only increase in memory requirements for the reformulation of the MP algorithm. The increase is insignificant for large M (training sequence length).

After each multipath successive interference cancellation stage, V^j is updated at the start of the next stage j as

$$V^j \leftarrow V^{j-1} - \hat{f}_{q_{j-1}} A_{q_{j-1}}. \tag{3.4}$$

Since the estimation of $\hat{f}$ via (3.3) depends only on V^0 and A, with A fixed, effectively the sufficient statistic is updated to reflect cancellation of the signal due to path q_{j-1}. Thus the matched filter output vector V^0 is updated to V^j, compared with the procedure in [57] where an intermediate canceled received signal r^j is formed and matched

filtering is repeated. The result is a speedup on the order of N_f times, as shown in Table 3.1.

The algorithm terminates after stage $j = N_f$. In practice, N_f can be determined on the fly based on $|\hat{f}_{q_j}|$ and/or the SNR (ratio of energy per symbol to noise energy per symbol duration). For the example in Figure 3.1, $N_f = 15$, $M = 1$, $N_c = 44$ binary DS-CDMA chips, and the MP channel estimate is shown (dotted line) for an SNR of 20 dB.

3.3 Reconfigurable Computation Model

The reconfigurable device paradigm is similar to that of software defined radio, in that devices can be easily re-programmed for adaptive response to operating conditions and applications. For instance, operating parameters including inter alia, frequency range, modulation type, and/or output power limitations can be set or altered. However, reconfigurable devices provide the additional benefit of programmable hardware, which allows the flexibility of software while yielding the high performance of a hardware implementation [48]. The performance and flexibility of reconfigurable computing systems make them ideal for implementing software defined radio systems.

Reconfigurable devices are a regular arrangement of programmable computational elements and communication structures, whose functionality is determined through configuration bits. There is a wide range of reconfigurable devices, which can be roughly classified by their granularity [48]. The *granularity* of a reconfigurable device

is the abstraction level used to program or configure the device. FPGAs and CPLDs have logical level of abstraction. Instruction level reconfigurable devices (e.g., PRISC [87], Chimaera [111] and Garp [37]) consist of computational units that perform arithmetic operations. Coarser grain reconfigurable devices, e.g., PADDI [102], MATRIX [77], RAW [104], synchroscalar [79] and NAPA [89], have even larger programmable computational units.

The granularity gives a notion of the underlying freedom of the device. A coarse grain device limits your flexibility. For example, you may be forced to store data in a specific register and choose from a prespecified set of operations. A finer granularity level allows you to specify arbitrary memory organizations and complex customized functional units. These fine grain devices can be configured to efficiently implement irregular functions. Furthermore, we can implement functions using any data width, e.g. an 18 bit multiplier or 24 bit adder, which can be customized to the application at hand. However, if we are executing only common operations, a coarser grain device will be the better option, since these operations are implemented using fixed hardwired "ASIC" components. The prespecified operations are built precisely for that operation and do not incur the overhead associated with building it using programmable logic elements. For example, a DSP application that requires a lot of word-size addition and multiplications would be best suited to a device with instruction level granularity. If an application requires a large number of Boolean operations, then a device with logic level reconfigurability would perform the task most efficiently. In general, the more

closely the application data is matched to the granularity, the more efficient the device will execute the application.

We choose a computational model that logically and physically divides the fine grain logic fabric into coarser grain *Block RAM-level operation blocks (BLOBs)*. Each BLOB consists of a Block RAM, fixed multiplier, and neighboring configurable logic blocks (CLBs). The CLBs are equally divided across the BLOBs. For example, the target chip in our experiments, Virtex-II XC2V3000 FPGA, has 3584 CLBs. Dividing by 96 (the number of BRAMs and fixed multipliers) yields approximately 37 CLBs in each BLOB. A BLOB is capable of performing any number of simple instructions, e.g., multiplication (on the fixed multiplier), addition (on the CLBs), and any other type of custom instruction that can be implemented on the CLBs. Additionally, it has an 18-Kbit BRAM that can act as a register file, mini cache, etc. Each BLOB is essentially a fully customizable data path, which causes most of the system energy consumption and delay.

Dividing the reconfigurable device into BLOBs has many advantages. First, we allow application developers to design using a higher level of abstraction. They can view the reconfigurable device as a sea of processors, which is an increasingly common method for developing computational fabrics [48]. Second, since the BLOBs are configurable at the logic level, we can program the fabric using a variety of different data flow and control methods, including SIMD and MIMD. Finally, the BLOB organization maintains the spatial model of computation that allows the reconfigurable

device to perform a large number of parallel operations, and therefore achieve high performance.

3.4 Parameterized MP Core

MP, due to its inherent parallelism, is an ideal candidate for efficient implementation on modern reconfigurable platforms. Since on-chip resources are limited, in this section we will study trade-offs of performance and area in implementing the parameterized MP core. To this end, we will explore the design space from the following three perspectives: parameterization of MP core, data representation, and data distribution.

3.4.1 Parameterization of MP Core

A good parameterization of MP provides trade-offs between important system metrics, such as estimation accuracy, latency, area, and power/energy consumption. The MP algorithm is inherently parameterizable in terms of the number of training symbols M, the number N_f of nonzero estimated channel coefficients, the number of samples N_s per symbol, and the length N_c and type of the spreading sequence.

The utilization of hardware resources on the target platform provides additional dimensions for parameterization of the MP core. MP involves matrix-vector multiplication, which can be decomposed into parallel vector-vector multiplications. The multiplication/accumulation in vector-vector operations can be further parallelized.

Bits	(S,E,F)	Adder		Multiplier	
		Performance (ns)	Area (slices)	Performance (ns)	Area (slices)
16	(1,6,9)	59.99	224	39.84	284
32	(1,8,23)	70.45	475	57.59	565
64	(1,11,52)	89.67	1054	69.84	2021
128	(1,32,95)	117.13	2200	100.91	5868

Table 3.2: Performance and area for floating-point functional units. *(S,E,F)* indicates the number of *sign*, *exponent*, and *fractional* bits, respectively.

Moreover, the channel coefficients of the uncanceled multipath signals can be estimated in parallel. Parallel computing can improve the latency of the platform, but this does come at a cost: more computation resources are required, which increases the cost of the target device. By varying the number of signal estimation resources used in the architecture, we can trade off latency for number of resources.

3.4.2 Data Representation

The second design trade-off can be performed by optimizing the binary representation of the data in the MP algorithm. Specifically, the number of bits used to represent each data value can be varied, and there is a choice between fixed point and floating point representations. Floating point provides large dynamic range and very high precision, but it can be costly. Fixed-point, on the contrary, represents the data less precisely, but it can save resources and have much better performance.

In MP, adders and multipliers are the basic functional units. Their estimated performance and area shown in Table 3.2 and Figure 3.3 can be used to provide rules

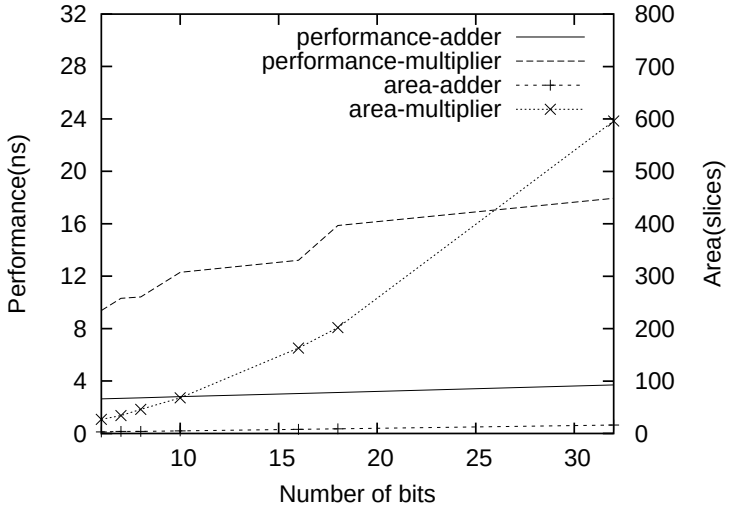


Figure 3.3: Performance and area for fixed-point functional units.

of thumb for choosing the right data representation scheme in the parameterized MP core. From Table 3.2 and Figure 3.3, we can see that compared with the fixed point representation (Figure 3.3), floating point functional units (Table 3.2) consume much more hardware resources and have much longer execution time[1]. Since the amount of hardware resources MP needs for multiplication and addition operations is very large, employing floating point data representation will require tight resource sharing, which correspondingly degrades the system performance. Fixed point representation restricts the accuracy of the signals in the digital domain compared with its floating point counterpart. However, it requires integer functional units, which are more efficient in both area and performance (Figure 3.3). Thus, fixed-point representation is employed in designing the parameterized MP core.

[1] Addition is one of the most computationally expensive arithmetic operations in floating point operations [13], as the results in Table 3.2 demonstrate.

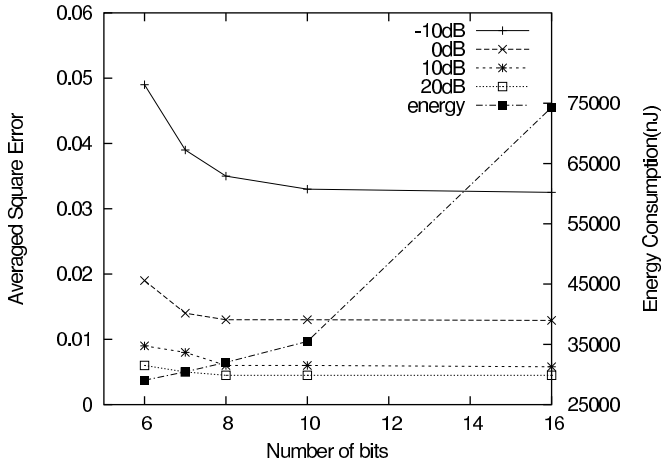


Figure 3.4: Channel estimation accuracy vs. number of bits for fixed point representation.

The other data representation consideration involves the trade-off between the number of fixed-point bits and the channel estimation accuracy. To explore the design space in this dimension, we conducted bit width analysis [32, 100]. Figure 3.4 shows the results for channel estimation average squared error (ASE) vs. number of fixed-point bits for SNRs of -10, 0, 10, 20, 30 dB, where SNR is defined as the ratio of the desired signal energy to the noise signal energy, both measured over one symbol duration. The results are averaged over three different multipath channels, with 30 ensemble runs (different noise realizations) per channel. From Figure 3.4, it is clear that eight bits is sufficient for accurate multipath channel estimation with optimal dynamic range scaling [32] throughout the implementation.

3.4.3 Data Distribution Schemes

Since reconfigurable devices provide the freedom to map various architectures, choosing the appropriate architecture affects the performance and energy dissipation. Another important trade-off to consider for speeding up the MP core is the data distribution schemes. MP processes a quite large amount of data, depending on the number of symbols, the number of chips per symbol, and the number of samples per chip. In such data dominated designs, poor data distribution can eliminate all benefits gained through parallelization, due to large data transfer times. Thus, it is necessary to perform a careful distribution of the data onto the target platform to achieve good latency. In the following section, we discuss the importance of the data and computation problem for on-chip communications and evaluate different architectures based on different schemes of partitioning the MP data and computations to meet the needs of high performance and low energy consumption.

Data and Computation Partitioning Problem

Clock frequency has risen exponentially over the years and the fraction of the chip that is reachable by a signal in a single clock cycle has decreased exponentially [40]. Architectures that rely on global signals are quickly becoming infeasible [28]. Therefore, care must be taken to distribute the data and operations onto reconfigurable systems in a manner that limits the amount of global communication.

We formalize the data and computation partitioning problem using the following architectural assumptions.

1. The programmable logic contains C configurable logic blocks (CLBs).

2. There are B BRAMs. Local BRAM can be utilized by architectural synthesis tools for local intermediate data; however, it is only used for this purpose.

3. The CLBs and BRAMs are equally distributed across the chip. Furthermore, we assume that the CLBs and BRAMs are equally divided into B BLOBs where each group consists of 1 BRAM and C/B CLBs.

4. CLBs can read/write data in the BRAM of the same group, which is called local access with a total latency of l clock cycles. If CLBs access data stored in BRAMs from another group (called a remote access), it takes a total r clock cycles ($r = l + d$) since we assume an average of d clock cycles will be taken because of the longer routing distance. Note that d is dependent on the distance of the BRAM that is being accessed.

Data and Computation Partitioning

Before going into details of the specific data and computation distributions schemes, we provide a stripped down overview of the matching pursuit algorithm presented in its full, formal, mathematical glory in Section 3.2.1. MP compares the received signal vector $\mathbf{r}$ with time delayed versions of a known training sequence that it expects

to receive from a transmitting user. The **S** matrix represents these time delayed training sequences. The i-th column in the **S** matrix corresponds to the training sequence delayed by i samples. The training sequence can be viewed as the transmitting device signature. The transmitting device sends its signature before sending data, so that the receiver can characterize the wireless channel between the transmitting device and itself. It then uses this channel estimate to demodulate future unknown data that it receives from that transmitting user. The data in the **S** is calculated using the signature (training sequence), CDMA spreading sequence and the transmit and receive filters, all of which are known a prior [57].

Matched filtering boils down to multiplying each sample of the received vector **r** with the corresponding sample of a column in the **S** matrix. Then, we accumulate all of these multiplied values to get a single value. This value represents the correlation of the received signal vector (**r**) with a time delayed version of the training sequence **(**S_i**)**. The correlation value between **r** and column S_i corresponds to the likelihood that the received signal has been delayed by i samples. However, it is important to note that multipath may cause destructive or constructive interference. Therefore, a high correlation value does not necessarily mean that the received signal has a path delayed by that number of samples.

Matching pursuits works by performing matching filtering of the received signal with all of the delayed training sequences (again, corresponding to the columns of the **S**). It takes the delay with the highest correlation and subtracts that signal from

the received signal vector. The algorithm iterates, continually canceling more signals, until it finds a sufficient characterization of the channel. Matched filtering takes a large majority of the computation time in MP, hence we focus on data and computation distribution for the filter. We evaluate different data and computation partitioning schemes for trading off energy and delay, and apply the optimal partitioning scheme into building the energy efficient MP core.

An important consideration for implementing the matching pursuit algorithm is the distribution of the data and computation. The matching pursuit algorithm and, in particular, the matched filtering, exhibits enormous opportunity for parallelization. Each matched filter operation can be performed in parallel. This corresponds to computing the received signal and path delays correlations in parallel. Furthermore, each multiplication of the matched filter can be performed in parallel. This corresponds to computing the samples of the received vector and path delay in parallel. However, we must then accumulate all of these sample correlations. The fastest method of doing this would be through the use of an adder tree. This would allow us to compute the matched filter in $O(log_2|r|)$ time at the expense of $O(|r|^2)$ multipliers. While the number of samples varies depending on the spreading sequence, typically, you need around 100 samples for the training sequences (we use 88 samples in our experiments). This would require approximately 10000 multipliers, which is far more resources that is available on even the largest reconfigurable devices. Even if such a device did exist, and we can easily extrapolate Moore's Law a few years to where such a device ex-

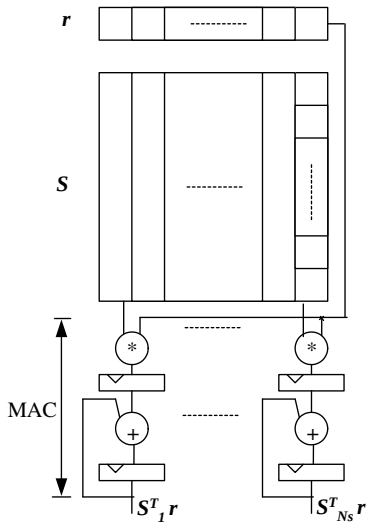


Figure 3.5: The local scheme for computing matched filter outputs. Matrix **S** are partitioned into columns, which are then distributed into block RAMs. Each matched filter shares an embedded multiplier and an adder, and its output is from a multiplication and accumulation (MAC) unit.

ists, we can rarely afford to devote the entire system to matched filtering. Therefore, it is imperative that we study the relationship of performance and design parameters between different data and computation partitioning schemes.

Seeing how the fully parallel scheme is infeasible, we must look to alternative schemes to serialize parts, if not all, of the operations to different trade-off design metrics, e.g. delay, throughput, area, power, etc. Matched filters involve a quite large amount of data. A poor data distribution will result in large data transfer times, which can eliminate all of the benefits gained through the parallelization. Therefore, it is necessary to carefully distribute the data onto the target device to achieve good performance. The two-dimensional nature of the **S** matrix guides us to the following two schemes for data distribution.

The local scheme distributes the $\mathbf{S}$ matrix into BLOBs by column (see Figure 3.5). Additionally, the received vector $\mathbf{r}$ is replicated and distributed to each BRAM. Therefore, each BLOB computes a matched filtering of $\mathbf{r}$ and a delayed training sequence. This sequentializes the computation of the individual matched filters, but computes all of the filters in parallel.

The BLOB is configured as a multiply-accumulate (MAC) datapath, using the fixed multiplier and an adder implemented on the CLBs. This scheme uses distributed local control logic, i.e. each BLOB is controlled locally. This requires a BLOB for each filter, which is equal to the number of samples (88 in our case). One could also imagine distributing multiple columns into the same BLOB. This provides a trade-off between execution time and area. Since the BRAM has limited number of ports, we would have to sequentialize the matched filtering for each column in the BLOB in all but the smallest column sharing schemes. For example, if two columns $\mathbf{S_i}$ and $\mathbf{S_{i+1}}$ are distributed in the same BLOB, then the computation of the matched filter $\mathbf{S_{i+1}^T r}$ follows the computation of matched filter $\mathbf{S_i^T r}$.

The global scheme distributes the $\mathbf{S}$ matrix into the BLOB by row (see Figure 3.6). Here, matched filters are computed in parallel, while the path correlations are computed sequentially. More precisely, the matched filter sample multiplications are computed in parallel, and the accumulate stage requires an adder tree. The adder tree is fully pipelined to allow overlapping execution of multiple matched filter accumulate stages.

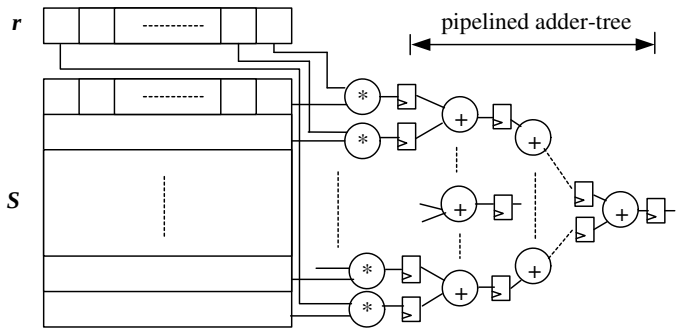


Figure 3.6: The global scheme for computing matched filter outputs. Matrix $\mathbf{S}$ are partitioned into rows, which are then distributed into block RAMs. Each match filter uses N_S number of multipliers and its output is from the pipelined adder-tree, which combines the multiplication results.

Each sample of the received vector **r** is divided into the BRAM of the BLOB. The BLOB is simply configured to perform multiplication. The accumulate stage is computed separately and requires global control logic and data transfers of the multiplied samples from each BLOB.

Once again, we can trade-off execution time for area by assigning multiple rows to each BLOB. This corresponds to sequentializing the matched filter operations. Each sample present in the BLOB would be executed in parallel, since we are limited by the number of ports on the BRAM and the number of multipliers that we implement in the BLOB. This would still require a separate adder tree, however, we can start performing MACs within the BLOB. Once again, consider the case where we partition two rows $\mathbf{S_j}$ and $\mathbf{S_{j+1}}$ onto the same BLOB. In this case, we could perform two multiplications, corresponding to the two samples from every path delay from $\mathbf{S}$. However, we could

also accumulate these two samples locally and send the two sample accumulated result to the adder tree for accumulation of the full delayed path. The BLOB data path here would resemble the data path from the local scheme. As both schemes move towards more serial implementations, we would indeed reach a point where the global and local schemes are equivalent. This would happen when both schemes become fully sequentialized.

Figure 3.2 (a) and (b) depict how the MP data S,A,a, are distributed in one section of a modern reconfigurable device, which has distributed memory, and dedicated IP cores (e.g. multipliers). The top-left figure shows how the RAMs and multipliers are equally distributed across the columns of the chip. All the blobs are identical. The bottom figure shows the configuration of one of the blobs, which includes a block RAM, a multiplier, and surrounding CLBs and programmable interconnections. From the MP algorithm (Figure 3.2 (c)), we can see that all the correlations (Line 2) in the matched filters can run independently. So the matrix S can be evenly distributed on the block RAMs, with each column S_i stored in one of the block RAMs for the discrete path delay i. The stored S_i are provided as the operands for multiplications. This parallelization of the matched filters results in a speedup factor of $O(N_s)$. Similarly, in the multipath successive interference cancellation steps (Line 5 through Line 13), a and A can be disseminated equally in the block RAMs, where a_k and A_k are saved in the same block RAM i with S_i. The block RAM i is also used as registers to save intermediate values of Q_k, g_k, and V_k^j, which are also independent between different

block RAMs. Due to resource limitations, the multiplications within the same blob share a multiplier in both steps of matched filtering and interference cancellation. By distributing those MP data that are independent onto block RAMs, the memory operations of multiple ports and operations on different block RAMs can be performed simultaneously. Thus, the system can gain more performance improvement.

In general, a major goal in the implementation of the MP core is the use of different techniques to optimize the core toward fast channel estimation. Based on inherent properties of the MP algorithm, the design space parameters are summarized as: (1) hardware resources, (2) parameterized parallelism of the MP algorithm, (3) fixed point architecture, and (4) distributed data storage schemes. Furthermore, the dedicated resources on the reconfigurable device are used to maximize the efficiency of the device.

3.4.4 Putting It All Together

Bits	Multiplier	BRAM	Slices	Performance (ns)
6	88	89	7806	4896.4
7	88	89	8342	4896.4
8	88	89	8969	4896.4
10	88	89	10134	4896.4
16	88	89	13630	8460.24

Table 3.3: Performance and area results for the MP core.

We present several experimental results for implementing the MP core on a Xilinx Virtex-II XC2V3000 FPGA [110] to better illustrate the general trade-offs involved in our study, specifically in terms of area/performance. Table 3.3 shows the results for

different numbers of bits per data value. From the table, we can see that the data are distributed across 89 block RAMs and 88 multipliers to support parallel execution, and when increasing the number of bits, the area (number of slices) increases accordingly, while the performance decreases. We also compared the execution time of the MP core with a high performance desktop computer, which has a 2.17GHz 3000+ AMD Athlon XP processor and about 1GB DDR PC3200 RAMs. The 8-bit MP core runs about 216 times faster, and the 16-bit MP core runs about 125 times faster. Here it is important to note that even though MP runs quickly on a high performance microprocessor, the achieved speed falls short for the high data rate required by 3G/4G wireless systems. In contrast, the proposed MP core meets the 3G/4G speed requirements, justifying its applicability for implementation in future generations of wireless communications systems.

We would like to mention that another MP core was also designed, which fully exploits the parallelism of the MP algorithm and is deeply pipelined. Yet, due to the large resource requirement, it does not entirely fit into the largest currently available FPGA. As fabrication technology improves and more transistors are integrated into a silicon chip, it will be possible to map the fully-parallelized MP core onto a single FPGA chip, enabling very high data rate processing.

3.5 Low Power Design Techniques

Traditionally, the performance metrics for signal processing and indeed, most processing in general, have been latency and throughput. Yet, with the proliferation of mobile, portable devices, it has become increasingly important that systems are not only fast, but also energy efficient. Currently, commercially available FPGAs either do not have both millions of gates and low-power features, or their support for low power feature is very limited. Purely relying on technology scaling will fall short of computational capability for more advanced algorithms which are demanded by wireless system in the near future. Thus, instead of studying low-level optimization techniques, in this section, we investigate and apply algorithmic and architectural level optimization techniques for minimizing energy consumed by FPGAs in building a multipath channel estimator. Our techniques can also be used for a next generation FPGA that has low power dissipation feature as well as high computing power. Our main contribution is a quantitative analysis of several energy efficient techniques that has resulted in novel optimizations at every levels, from the theory and algorithms to the architecture and bitwidth. We describe our design and quantify the trade-offs in terms of channel estimation accuracy and the energy of our implementation. Along with exploring the clock gating technique, our final result is an energy efficient MP core, resulting in 25.4% of total power savings.

Energy and power are often used interchangeably, however, they are not the same. Energy is the product of average power dissipation and latency. Therefore, it is neces-

sary to understand power dissipation and its effect on latency and vice versa in order to better understand energy dissipation.

In this section, we will first briefly describe sources of power dissipation in FPGA based reconfigurable devices. We will then discuss techniques for achieving low energy dissipation by one of the three methods: lowering power dissipation, lowering latency, or lowering the product of the power and latency, at different abstraction levels, from the arithmetic and architectural to the algorithmic level. The result of our design is an energy efficient IP core that can be readily tuned to the requirements of its applications and initiated in any number of wireless devices.

3.5.1 Energy Dissipation in FPGA based Reconfigurable Devices

There are two primary areas of power consumption in FPGAs. Static power comes from transistor leakage, and dynamic power comes from voltage swing, toggle rate, and capacitance. Both are important factors in meeting a power budget and power optimization. Several studies on power dissipation of reconfigurable devices have appeared in recent works [63, 105]. These works review that power dissipation in reconfigurable devices is primarily due to programmable interconnects. For instance, in the Virtex-II, the dynamic power dissipated in the interconnects is about 50% to 70%, while the remaining is being dissipated in logic, clock and I/O blocks. These results are different from ASIC technology, where clock distribution typically dominates power dissipation [1]x. The programmable interconnects consist of multiple pre-fabricated

row and column interconnect wire segments of various lengths, with used and unused routing switches attached to each wire segment.

The second important factor that affects the power dissipation in reconfigurable devices is resource utilization. In typical reconfigurable system designs, a large number of of the resources are not utilized and only static power is dissipated on those unutilized resources.

Switching activity is another important factor that is used to determine the amount of dynamic power dissipation of each hardware resource. It depends not only on the type of the design but on the input stimuli.

To obtain the power consumption information of the target Xilinx chip XC2V3000, we did low-level simulation of the VHDL codings of our design with Mentor Graphics Modelsim and generate simulation results (.vcd file). The input vectors for the simulation was obtained from the high level simulation of the MP algorithm. The design was synthesized using Xilinx Synthesis Technology and the place and route file (.ncd file) was obtained. These two files were fed into Xilinx XPower tool to evaluate the average power dissipation. Energy dissipation was obtained by multiplying the average power by latency.

With good understanding of the sources of the power dissipation and the way to obtain the power dissipation, we can now discuss design and optimization techniques to achieve an energy efficient channel estimator, from the bit level to the algorithmic level design.

3.5.2 Binding Energy Efficient IP Cores

As modern chips are increasingly providing high computational power with the fixed components [110], like microprocessor cores, embedded multipliers and on-chip distributed memory, a very important factor to consider in designing systems is to choose energy efficient bindings and map operations onto available hardware resources. Different bindings affect energy consumption of the reconfigurable device greatly. For example, embedded multipliers, such as those in the Virtex-II and Altera Stratix families, can be more energy efficient than the multipliers implemented with CLBs. Our analysis shows that the energy consumed by a CLB-based multiplier is approximately twice of the energy consumption by an embedded multiplier core.

3.5.3 Bit Level Optimization — Bitwidth Analysis

As we have studied in Section 3.4, floating point functional units take much longer execution time and consume copious amount of power than their fixed-point counterparts. Therefore, we employ the fixed point representation in the rest of our study, which we show can provide reasonably accurate results.

An important consideration for implementing the matching pursuit algorithm is decision on the number of fixed-point bits. The larger the bitwidth, the more accurate the estimation results. Conversely, bigger bitwidths lead to larger and slower functional units, which has obvious negative effects on the latency and power. Therefore, it is imperative to find a good tradeoff between accuracy, latency and power. Referring to Fig-

ure 3.4, it has been shown that 8 bits is sufficient to achieve accurate multipath channel estimation, with averaged square error 3.37%. Fewer number of bits (e.g., 6 bits will result in 4.8% error) can lead to minor improvements in performance and power. This, however, comes at the cost of the large system error. Conversely, large bitwidth (16 bits) should not be used because it takes longer execution and larger power consumption, and only can improve the accuracy by a small degree (about 0.12%). Therefore, a bitwidth of 8 is used in our study, which hits a sweet spot between accuracy, performance and power.

3.5.4 Architectural Level Optimization

Since reconfigurable devices provide the freedom to map various architectures, choosing the appropriate architecture affects the energy dissipation. Based on previous studies, interconnect dissipates a large amount of power. Therefore, minimizing the number of long wires or global communications between building logic blocks is beneficial. In the following section, we evaluate different architectures based on different schemes of partitioning the MP data and computations to meet the needs of low energy consumption.

Figure 3.7 shows the results of performance and power consumption vs. granularity (number of columns or number of rows partitioned into each block RAM for the local scheme and the global scheme, respectively). The figure illustrates that for both schemes as the number of columns/rows of the **S** matrix packed into the same block

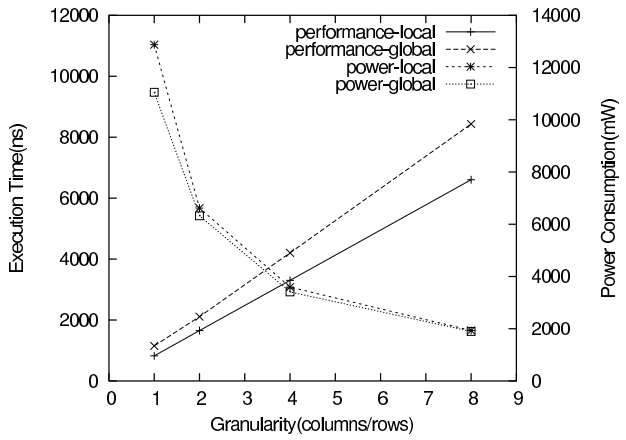


Figure 3.7: The tradeoff of performance vs. power by using both the local and the global scheme.

RAM increases, the execution time increases linearly and the power consumption of hardware resources decreases exponentially. This observation can be explained by understanding that each matched filter is sequentially executed if multiple columns/rows are within one block RAM, which takes less power consumption but longer execution time. Depending on different system requirements, designers can follow the curve to implement the matched filters optimally. In the situation where high data rate is required, the best implementation is to employ the local scheme and distribute every column into each block RAM to achieve the best performance.

Comparing Figure 3.7 and Figure 3.8, we can make the following observations. The results presented are based on the distribution of only one column or row per block RAM, though other similar distributions exhibit similar behavior. First, after placement and routing, the local scheme can achieve the best performance(925.73ns), which

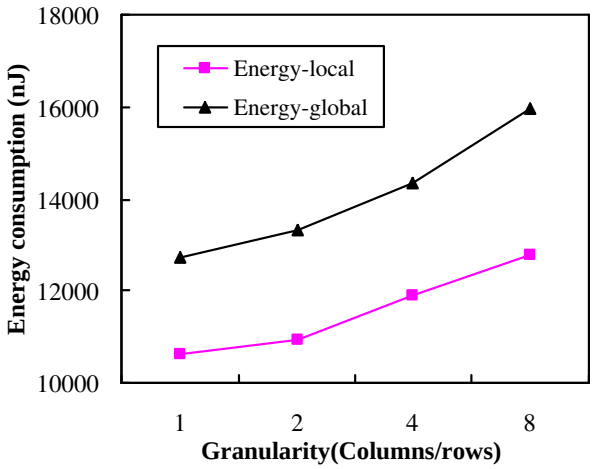


Figure 3.8: The energy consumptions by using both the local and the global schemes.

is about 376ns faster than the global scheme(1311.4ns). Second, the local scheme achieves less energy consumption (4143nJ less) than the global scheme. These can be attributed to the fact that the pipelined adder-tree requires a large amount of global communication across the chip, which degrades the system performance and further leads to large energy consumption. The overriding theme is that architectures using the local scheme takes less time than that of using the global scheme, in terms of RTL synthesis and placement and routing time. This is due to the fact that the local scheme uses functional units and control logic that are distributed locally to the block RAMs, while the global scheme uses centralized control which requires tools taking longer time to figure out the detailed placement and routing. Therefore, the local scheme with one column in each BLOB is employed in our design, which can achieve the least amount of global communications and the lowest energy consumption.

3.5.5 Algorithmic Level Optimization — Module Disabling

At the algorithmic level, a lot of optimizations can be conducted [48], with a strong impact on the system's power consumption. The MP algorithm is designed to be easily parallelized, i.e. the BLOB-level computation of each path can run independently of others. This enables the clock gating technique to disable BLOBs that are not in use or have been detected during the computation. By disabling the unnecessary computation modules, the power dissipation can be reduced. In MP, for instance, after canceling a significant path, the computation of this path is useless but still consumes power. With the BLOB model of the execution of each path, the implementation can exploit clocking gating to disable the computation modules of those detected paths.

As discussed in [56], for radiolocation applications the MP stopping criterion can not only ensure that the most significant paths are accurately detected, but guarantee detection of the direct path, which can be consequently used to measure the line of sight distance from the transmitter to the receiver using the TOA-based method. When the stopping condition is met based on the stopping criterion, the MP core can be simply switched off to further save power.

In FPGAs, clock gating can be realized either by using primitives such as BUFG-MUX to switch from a high frequency clock to a low frequency clock [90] or by introducing a sleep transistor to switch the unnecessary modules off [4].

To study the power consumption of the total on-chip hardware resources, we divide the resources into three types: unused, active, and disabled parts. The unused part is

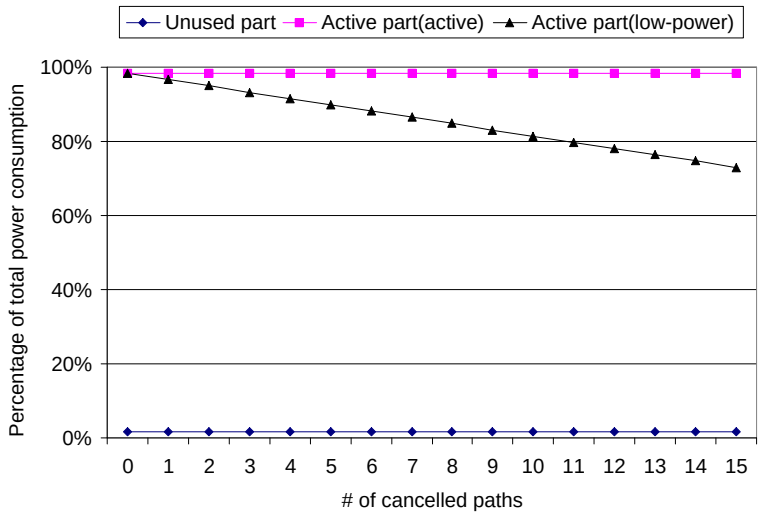


Figure 3.9: Percentage of power savings with the algorithmic level optimization.

the amount of hardware resources that are not contribute to building the MP core, and thus it only consumes static power, which is proportional to the resource utilization. The active part includes the resources that are always actively executing, which contribute to both dynamic and static power dissipation. The disabled part has no dynamic switching and thus only contributes to static power consumption.

Figure 3.9 shows the percentage of the total power consumption as the paths are detected and canceled by disabling the unnecessary modules during the computation. From the figure, we can see that the unused part consumes 1.67% of total chip power consumption, i.e., there is 1.67% of unused on-chip resources. When we apply the algorithmic level optimization for achieving low power, the power consumption decreases linearly as the detected paths are successively canceled, and for each cancellation the power saving is about 1.69% compared with leaving all the canceled paths

active. After detecting 15 paths, the MP terminates, resulting in a total power saving of 25.4%.

3.6 Applications of MP Core for Channel Estimation

With accurate channel estimation, MP core can be used as an alternative to other advanced algorithms [16, 20, 44, 84, 108] in various wireless communications systems. Among them, CDMA base-stations and wireless sensor networks are the most promising systems that researchers have been addressed.

To accommodate the demand for wireless communicaitons services, the next generation wireless communications systems use Code Division Multiple Access (CDMA) as the multiple access protocol for efficient use of limited available frequency spectrum, increased system capacity and speed of communications as well as accuracy. In a CDMA system, uses are allowed to transmit simultaneously and occupy the same bandwidth. CDMA uses spread spectrum signaling so that the overall transimission rate is much higher than the maximum needed to transmit the actual data. Each user's data bits are modulated by a unique signature sequence (spreading code) that can only be detected at the basestation which also knows the coding waveform. To maximally differentiate between users, the spreading codes assigned to the users should be mutually orthogonal. This is very difficult and a large part of research has focused on this issue. The base-station receives a summation of the signals of all the active users after they travel through different paths in the channel. These channel paths include differ-

ent delay, attenuations and phase-shifts to their signals and the mobility of the users causes these parameters to change over time. Moreover, the signals from different users interfere with each other (Multiple Access Interference) adding to the additive white Gaussian noise present in the channel. Multiuser channel estimation refers to the joint estimation of these unknown parameters for all users to mitigate these undesirable effects and accurately detect the received bits of different users. Interference cancellation or multiuser detection requires the use of such highly accurate estimate of the channel for proper detection. The efficient MP core, when combined with the GSIC algorithm [57], is promised to enable high data rate multiuser detection in CDMA base-stations.

Wireless sensor networks are widely deployed as another promising application that may revolutionize the way in which we understand and manage complex physical systems [25], e.g., physiological monitoring, environmental monitoring (air, water, soil chemistry), smart spaces, military, precision agriculture, transportation, factory instrumentation and inventory tracking. A low-power wireless technology called ZigBee [5] has been proposed as a wireless network standard that meets the unique needs of sensors and control devices. ZigBee's success is largely due to its low data rate. ZigBee won't be sending email and large documents, as Wi-Fi does with its 54Mbps 802.11g technology, or documents and audio, as Bluetooth does with a nominal 1Mbps frequency band. For sending sensor and control information, high bandwidth isn't necessary. Due to its low bandwidth requirements, a ZigBee node can sleep most of the

time and save power, which fulfills its goal of low power, low cost, and robustness. A big part of ZigBee's power saving comes from the radio technology of 802.15.4, which uses direct-sequence spread spectrum (DSSS) technology because the alternative frequency-hopping spread spectrum (FHSS) would have used too much power just in keeping its frequency hops synchronized [62, 5]. Using the same spectrum waveform, MP core promises to be employed in ZigBee for accurately estimating parameters to characterize channels.

3.7 Related Work

Much research has been conducted on the topics of wireless channel estimation, design space exploration, and IP reuse.

Channel estimation for wireless applications has attracted considerable attention recently, and a variety of algorithms have been developed [16, 20, 44, 57, 84, 108]. In [84], an approximate ML-based algorithm was targeted for implementation in DSP hardware. In the ML algorithm redesign, a matrix inversion step was approximated via gradient descent for improved efficiency. We choose the MP-based algorithm [57] because of its efficiency and accuracy, and we redesign the algorithm for improved efficiency based on a sufficient statistics interpretation.

Design space has been explored from different perspectives through various techniques to meet the design objectives [76, 106]. In this work, we focus on the bit-width analysis [27, 32, 100] and data distribution [43]. BitWise [100] determines the mini-

mum number of bits by propagating static information in the program data-flow graph, and Bitsize [32] decides the number of bits through sensitivity analysis of outputs. Through careful bit width analysis, hardware costs and energy consumption can be substantially reduced. Huang et al. [43] proposed a method for HLS to distribute data across memory logic blocks for reducing data communications.

IP reuse is a design method for bridging the gap between available chip complexity and design productivity. Quite a few methods and techniques [46, 73] have been proposed for IP creation, assembly and testing. Metacores [73] is similar to our work; it creates parameterized cores for Viterbi decoding and IIR filters.

We want to emphasize that there are numerous examples where reconfigurable devices execute signal processing applications substantially better than processors in both overall performance and energy/power. Wouters *et al.* designed an OFDM modem with adaptive loading on a Xilinx XC2V6000 FPGA [109]. Scrofano *et al.* [96] compared a representative FPGA (Xilinx Virtex-II Pro) with a digital signal processor (DSP) (TI TMS320C6415) and an embedded processor (Intel PXA250). Their results showed that Virtex-II Pro consumes much less energy than any of the other devices. Choi *et al.* [18] studied FFT and matrix multiplication. Their FFT design achieved 56% less energy consumption than a DSP. In terms of performance, the FPGA implementation resulted in a 10 fold improvement over an embedded processor. Liang *et al.* [67] studied a dynamically reconfigurable turbo decoder, which showed 100 times performance improvement over contemporary microprocessors. Swaminathan *et al.* [103]

implemented an adaptive Viterbi algorithm on an FPGA, achieving a speed-up of 29 compared to the DSP implementation. Stitt and Vahid [101] showed that configurable logic provides an average energy savings of 25% to 71% over a series of embedded system benchmarks. Sakiyama *et al.* [92] described 4X performance improvement of a JPEG decoder implemented on FPGA versus its DSP implementation. In [17], the authors compared the implementations of coded-matched filter (MF) and minimum output energy (MOE) detectors on a Xilinx Virtex XCV800 FPGA and a TI 'C6x DSP. They showed that the ratio of the DSP throughput to the FPGA throughput is 1/2 for the MF detector and 1/11 for the MOE detector. In addition, they compared the implementations of Viterbi decoding, and reported that higher data rate can be achieved with the FPGA implementation than its counterpart DSP implementation. Rajagopal and Cavallaro [85] presented an implementation of the pipelined multiuser detection algorithm by integrating both FPGA and DSP, and showed that the FPGA achieves 4X speedup over the DSP implementation.

3.8 Summary

Wireless connectivity is playing an increasingly important role in communication systems. To meet the demands of higher data rate and higher multi-user capacity, channel estimation has been employed as the key to modern communication algorithms. Given the frequency with which new wireless protocols are developed and deployed, an ASIC based approach is a poor option due to it's lack of programma-

bility. On the other side, with the computational demands that the current generation of signal processing algorithms place on a device, a microprocessor simply could not provide enough throughput. A reconfigurable device provides an excellent balance between these two extremes, and also presents a unique opportunity to designing and optimizing the signal processing algorithms in concert with the actual hardware implementation. In this chapter, we described a cross-cutting method to explore the design space to solve the channel estimation problem on reconfigurable devices.

As high performance and energy efficient implementations remain as a design challenge, this chapter is focused on building a high speed and energy efficient matching pursuit IP core. Through exploring the design space, a high speed and energy efficient MP core was developed, specifically, by redesigning an MP algorithm for efficient channel estimation, and by employing low power optimization techniques at all levels of the system design: bit, architecture, and algorithm levels.

At the bit level, we have studied the tradeoff of performance, energy consumption vs. accuracy. At the architectural level, we have investigated many different data and computation partitioning schemes, and found that an effective way of partitioning an application is to treat the reconfigurable device as a collection computational blocks, where each block has a single block of memory and an associated set of computational abilities. We call each of these logical units a BRAM-level operational block, or "BLOB". We have demonstrated that by keeping both the control and the data signals local to each BLOB to the greatest extent possible, the implementation can achieve

the highest performance and the least energy consumption. At the algorithmic level where the decision has the strong impact on the system power consumption, we have employed the clock gating technique to disable the unnecessary computation modules, achieving 25.4% of total power savings. The parameterized MP core was developed and mapped on a Xilinx Virtex-II FPGA, executing over 216 times faster than a high performance desktop machine. With this level of efficiency, in the near future the parameterized MP core could enable high data rate multiuser detection, delivering on decades of promises from communications theory and revolutionizing the state of the art in applied wireless technologies.

Chapter 4

Exploring Generalized Model for Optimizing Leakage Power Dissipation

4.1 Introduction

Power dissipation has become a major concern to those designing processors for high performance desktops, servers, and battery-operated portable devices. Higher energy dissipation requires more expensive packaging and cooling technology, which in turn increases cost and decreases system reliability. There are fundamentally two ways in which power can be dissipated: either dynamically (due to the switching activity of repeated capacitance charge and discharge on the output of the millions of gates), or statically (mainly due to sub-threshold and gate leakage [54, 82]). Dynamic power consumption is proportional to the square of the supply voltage, which reduces as process technology scales. While the scaling down of transistor geometries enables the reduction of the dynamic power, it worsens the leakage problem greatly. If current

technology scaling trends hold [45], leakage will soon become the dominant source of power consumption, and as such new techniques are needed to battle this growing problem.

The problem of leakage becomes more significant as threshold voltage, channel length, and gate oxide thickness are reduced. Furthermore, the sub-threshold leakage, as a major component, stems from the need for a trade-off between dynamic power and performance. Scaling down the transistor supply voltage reduces the dynamic power dissipation. Yet, to maintain high switching speed under reduced voltages, the threshold voltage must also be scaled. As the threshold voltage drops, it is easier for current to leak through the transistor resulting in significant leakage power dissipation. The increases in device speed and chip density exacerbate the leakage problem. New technologies targeted at reducing dynamic power and increasing performance, such as low threshold voltage [69] and gate oxide scaling [60], further increase the relative importance of leakage power [45] (Figure 4.1).

Cache memories have long been used to reduce the ever-growing gap between processors and memory. Modern processors typically provide two levels of on-chip caches (e.g. separate L1 instruction and data caches and a unified L2 cache). In these processors, a large and growing fraction of the total on-chip area, and an even larger fraction of the total number of transistors, is consumed by caches. Because they account for such a significant portion of the total chip real estate, caches provide a healthy-sized target for designers to try circuit and architectural optimizations with the goal of re-

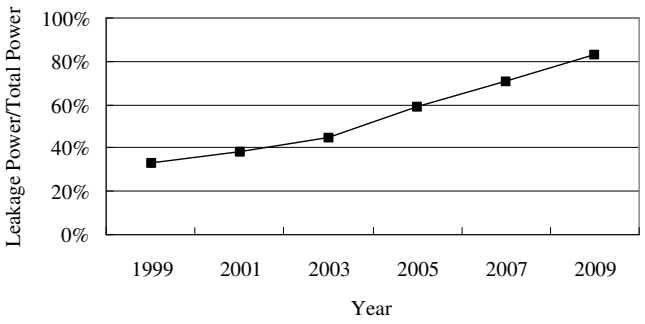


Figure 4.1: Projected leakage power consumption as a fraction of the total power consumption according to the International Technology Roadmap for Semiconductors [ITRS].

ducing leakage power. The central idea behind most of these techniques is to exploit some form of temporal locality. By putting infrequently or unused cache lines into low leakage mode, much of the power will be reduced. By keeping frequently accessed cache lines active, total performance will not be reduced significantly.

Though there are several circuit techniques and management schemes concerning how and *when* to turn on or off individual cache lines, little work has been done to explore the *limit* of how well such techniques can work. What is the best we could hope to do with a given low power technology? The primary goal of this chapter is to explore these limits under different architectural and design assumptions in the hope of guiding research effort on leakage power in much the way that Belady's OPT algorithm [12] helps (and continues to help) in the study of replacement policies.

There has been much work on leakage power reduction already, and any proposed methods for calculating the limits of their effectiveness must be both general enough

to capture a variety of techniques, yet specific enough to provide useful bounds. Our methods capture both state-preserving and state-destroying techniques, and additionally we show how to optimally combine two such techniques into a hybrid scheme. We show that, given perfect knowledge of the future address trace, there exists a break-even point between Drowsy and Gated-V_{dd}. If the same cache line is accessed twice in an interval of time less than or equal to this break-even point then Drowsy mode should be used. If the same cache line is not used again within an amount of time greater than the break-even point then more power can be saved by turning off the cache line using Gated-V_{dd}. If these timings are known, then an optimal policy can be achieved.

Clearly perfect knowledge of the future trace is not always known, but it serves several purposes. First it provides an important bound. No management method will be able to beat our power reduction scheme under the given circuit assumptions. Second, it demonstrates that there is still a great deal of potential for policy decisions (when to turn a cache line on or off) to significantly reduce leakage power. Finally, while perfect knowledge of future references cannot be known, it can often times be approximated by architecture techniques such as address prediction or *prefetching* [75].

In particular, we make the following contributions:

1. We relate the potential savings that can be obtained from Drowsy and Gated-V_{dd} techniques, under various assumptions for both the L1 instruction and data caches, and the unified L2 cache.

2. We show that with oracle knowledge of future accesses, a simple optimal power management scheme can be derived from a small set of circuit parameters.

3. In addition to showing the optimal leakage savings on a set of implementation parameters, we develop a parameterized model to determine the optimal leakage savings while the implementation technologies and architectures change over time.

4. We show that while both Drowsy and Gated-V_{dd} schemes are useful on their own, when combined, we can push the upper bounds of the leakage power savings to 96.4%, 99.1%, and 97.7% for the instruction cache, the data cache and the unified L2 cache, respectively, with the 70nm implementation technology.

5. We also show that the model can be applicable to explore the limits of leakage power savings for different implementation technologies and cache configurations.

6. In addition to the limits study for the L1 instruction and data caches, We study the limits for L2 caches when both sleep and drowsy modes are employed. Taking a large amount of area in modern processors, L2 caches exhibit themselves as another interesting target to battle the leakage problem. Without careful attention to power, L2 caches may overtake the chip's power budget.

7. We conduct the leakage study on different cache configurations to validate our methods.

8. Instead of examining the leakage reduction on only five benchmarks [75], in this study we investigate our methods across all the SPEC2000 benchmark applications.

9. We also empirically study the interval distribution to show how much percentage the short dead intervals contributes to the total leakage reduction.

The rest of the chapter is organized as follows. We review related work and motivate our limit study in Section 4.2. In Section 4.3, we propose our method for combining the Gated-V_{dd} method and the drowsy method. We also explore the limit of leakage power saving that we can potentially achieve using our hybrid scheme. A model which parameterizes all the individual assumptions is also proposed. Section 4.4 describes our simulation setup and the benchmarks in our study, and shows the results of our empirical study in exploring the upper bounds. We also study the generality of the parameterized model on different cache configurations. We offer concluding remarks in Section 4.6.

4.2 Circuits and Architectures of Reduced Cache Leakage Power

In order to derive a useful limit for leakage power reduction in caches, we must first begin with a discussion of those related technologies so that our model will be

grounded in reality. In this section we review several circuit techniques, and develop the general ideas of our method.

Leakage power comes from transistors that are simply left on, and the easiest way to think about reducing the amount of the consumed leakage power is to "turn off" those transistors that are not needed. While this is the easiest to think about, it is by no means the easiest to implement. One such approach, Gated-V_{dd} [80], attempts to solve this problem by reducing leakage through the use of a high threshold sleep transistor (between pull-down NMOS and virtual V_{ss}) to break the connection and thus increases the L1 cache line access time. This transistor is in the read critical path which may impact performance, however we only consider the potential for energy savings in this paper. This leakage reduction technique is often called *sleep mode*, and this is the naming convention that we use here. While efficient in saving leakage, *sleep mode does not preserve the state of the data*. When a cache line is needed again after it has been put to sleep, it must be re-fetched from lower levels of the memory hierarchy. This re-fetch is essentially an extra cache miss, and this process can take many cycles depending on the memory hierarchy, architectural assumptions, etc.

A different way of saving leakage power in the caches is to make use of *multiple supply voltages*. When the cache line is left fully on, it will dissipate too much leakage power. If V_{dd} is fully gated, it will use very little power, but the data is lost. A compromise is to use a *lower* supply voltage when data is not needed for a while. This will reduce the leakage power without losing the data. The trade-off is that, while data will

be preserved at this low supply voltage, it cannot be accessed while in this state. Thus there is a small wakeup time associated with changing from the lower voltage up to V_{dd} (hence the name "drowsy"). If this can be implemented without adding a high-Vth transistor in the read critical path as the initial proposal, drowsy mode has the potential to achieve a smaller L1 cache access time than sleep mode. Drowsy mode does not fully turn off the memory, and thus does not reduce the leakage power as much as Gated-V_{dd}. For a piece of data that is not going to be accessed for a very long time, sleep mode will be better because it reduces more leakage power. For a piece of data that is accessed in a moderate amount of time, drowsy mode will be better because there is not a large re-fetch penalty. This sets up one of the fundamental questions answered in our paper — how long is *long enough* for each mode?

While our paper attempts to address a previously unanswered question, there is a great deal of prior work aimed at reducing leakage power in caches. Azizi et al. [9] introduced asymmetric dual-V_t SRAM cell caches(ACCs). ACCs exploit the fact that in ordinary programs most of the bits in caches are zeros for both the data and instruction streams, and provide significant leakage reduction in the zero state. DRI-cache [80] uses the Gated-V_{dd} technique to dynamically adjust the size of the active portion of the cache by turning off a bank of cache lines based on the miss rates. DRG-cache [3] employs Gated-V_{dd} to reduce leakage power by turning off the gated-Ground transistor, while data is restored when the gated-Ground transistor is turned on. DTSRAM [39] uses body biasing to separately control the V_t of each cache line. To minimize the

energy and delay overhead, a cache line is switched to high V_t when it is not likely to be used anymore. Kaxiras et al. [42, 49] proposed the cache line decay scheme to turn off the cache lines in the dead periods of their cache generations using the Gated-V_{dd} technique. Instead of placing both the tag and the data into the sleep mode, AMC [114] keeps the tag alive and tracks the miss rate with respect to the ideal miss rate. This helps to dynamically adjust the turn-off interval and control the overall performance. Velusamy et al. [107] used formal feedback-control theory to adaptively adjust the cache decay interval and cache lines are turned off accordingly. Another approach to reducing leakage power is called drowsy cache [30, 52, 55], which decreases the supply voltage of idle cache lines. Specifically, all cache lines are periodically placed into drowsy mode. [53] studied techniques for data retention with lower supply voltage. [41] employed drowsy cache to exploit program hot-spots and code sequentiality for instruction cache leakage management. Parikh et al. [66] compared Gated-V_{dd} and drowsy cache at different L2 latencies with HotLeakage and showed Gated-V_{dd} is superior for a set of faster L2 latencies. Heo [38] reduced bitline leakage by leaving bitlines open whose cache banks are not accessed. Hanson [35] found that for L1 caches, MTCMOS, which is a state-preserving technique that operates multiple threshold voltages, outperforms Gated-V_{dd}. In [65], the authors presented several architectural techniques that exploit the data duplication across the different levels of cache hierarchy. They found that the best strategy in terms of energy and energy-delay product is to place the L2 subblock into a state-preserving mode as soon as its contents

are moved to L1 and to reactive it only when it is accessed. Bai et al. [10] investigated the impact of T_{ox} and V_{th} on power performance trade-offs for on-chip caches. In contrast, [94, 112] studied software approaches. [94] decided the decay interval through profiling and showed that the optimal decay intervals can be estimated with a reasonable degree of accuracy using profiling. [112] studied using compiler to insert power mode instructions that control the voltage for the cache lines to control leakage energy.

All of the above approaches strive to develop a scheme for predicting when a section of the cache should be put into a low power mode. They use some heuristics based on either static analysis or run-time behavior to determine what mode each line should be in. One major open question is: what is the best that these approaches could hope to do? Clearly some of the cache lines will have to be left in a high V_{dd} mode so they can be accessed, but how many and for how long? Are these approaches the ultimate in policy leakage power reduction, or is there still room for improvement?

4.3 Calculating Limits of Leakage Power Reduction Techniques

Now that we have reviewed the circuit and architecture techniques employed to reduce leakage power, we describe how to calculate the savings that could be achieved by an optimal method.

```
.....
int i, j, sum, total;
int low(int);
int high(int);
.....
for (total = 0, i = 0; i < 12; i ++)
{
   for (sum = 0, j = low(i); j < high(i); j ++)
      sum += a[j];
   sum *= i;
   add: total += sum;
}
.....
```

Figure 4.2: The access interval example. The interval length of the consecutive accesses to the *add* instructions depends on the range of the inner loop $|high(i) - low(i)|$.

4.3.1 Cache Intervals

Our analysis of the leakage power saving limit relies on the idea of breaking up the life time of each cache line into a series of *intervals*. An interval is the time that a cache line rests between two accesses. If an interval is very long then it would be beneficial to put that cache line in sleep mode for the duration of that interval. If an interval is very short, it should be simply left in a high-V_{dd} mode. If an interval is somewhere in the middle, perhaps drowsy mode would be the best.

To illustrate the above situations, let's take a two-level loop example (Figure 4.2) extracted from a human resource management application. It counts the total number of people employed during a year. In the example, the interval (I_{add}) of the two consecutive accesses to the same instruction *add* depends on the size of the inner loop.

When the range of the inner loop variable j is large, the interval I_{add} is long, which indicates the cache line of add instruction should be put into sleep mode to save leakage power. And when the range is very small, the interval I_{add} is small, which means this cache line should be left in the high-V_{dd} mode for fast accesses. While the range is in the middle, the drowsy mode should be applied to save leakage power without much performance cost. The idea behind our optimal scheme is to determine what the best policy would be for each interval in the program, and then to apply the appropriate leakage technique to that interval.

In an optimal approach, each interval can be thought of as atomic in the eyes of the optimal policy. With oracle knowledge of the future address traces known (as would be for an optimal approach), there should be no reason to perform any new power saving techniques in the middle of an interval. Instead, the same technique should have been applied for the entire duration of the interval as less power would be consumed with the same penalty (for either wakeup or re-fetch).

One thing to note is the notion of live intervals and dead intervals. A live interval starts when a new memory is brought into the cache frame, and ends after the last access. Between the last access to a line of memory and the time it is evicted from the cache, it is regarded as dead. Besides turning off cache lines in dead periods as the cache decay scheme does [49], our method also explores the live period of a cache generation, which demonstrates great potential for leakage reduction. In fact we found that dead periods did not contribute a large amount of leakage savings in

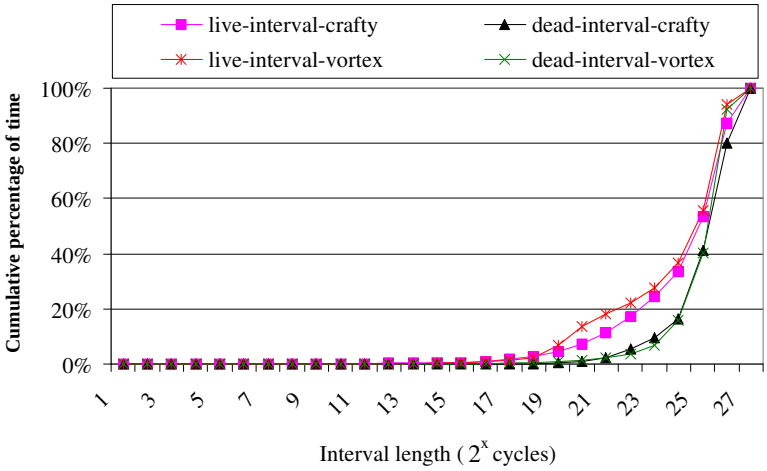


Figure 4.3: Cumulative distribution of live intervals and dead intervals of the L2 cache for *crafty* and *vortex*. The total amount of short dead intervals only contribute little to the leakage power reduction, while the long intervals play a major role.

the optimal case, because *any* long interval would be turned off whether live or dead. The only additional savings that are achieved from considering dead intervals are from short dead intervals, of which there are very few. Figure 4.3 is used to demonstrate such a point. It is drawn based on our experimental setup (see Section 4.4.1). The x-axis shows the interval length and it is log_2-scaled. The y-axis shows the cumulative percentage of the live intervals over the sum of all live intervals and dead intervals over the sum of all dead intervals of the unified L2 cache for *crafty* and *vortex*. As it can be seen, the curves of the dead-interval-crafty and dead-interval-vortex arise when the interval lengths are large (greater than 2^{20} cycles), and the short dead intervals only contribute an insignificant amount (less than 1%) for the sum of all the dead intervals, which indicates that the short dead intervals contribute little to the leakage

power reduction. Thus, for the rest of this paper we ignore the effect of live and dead intervals, and instead concentrate on only the durations of the intervals.

4.3.2 Optimal Method

Our optimal method works as follows. Given an interval distribution of cache accesses, which can be obtained based on a memory configuration, our optimal method first classifies each cache access interval into one of the following three types: sleep-mode optimal, drowsy-mode optimal and active-optimal, and applies the appropriate mode on each interval to obtain the optimal leakage power saving. If the size of an interval is very small (i.e. there are multiple consecutive accesses within a short period of time), then it is best to leave the cache line in a fully active (non-power saving) mode. If the size of an interval is long, then the best policy is to completely turn off that cache line (sleep) and then re-fetch it when it is needed again. The final case is if the interval size is neither very long nor very short. In this case it is best to put the cache line into a drowsy state, which consumes a small amount of power, has a small wakeup cost and has the advantage of retaining the data values.

The key to dividing intervals into these categories is knowing the precise length of an interval that should be put into sleep mode, drowsy mode or left active. The interval length where the power saving mode changes is an *inflection point*. There are two inflection points: one between sleep and drowsy modes and the other between drowsy and active modes.

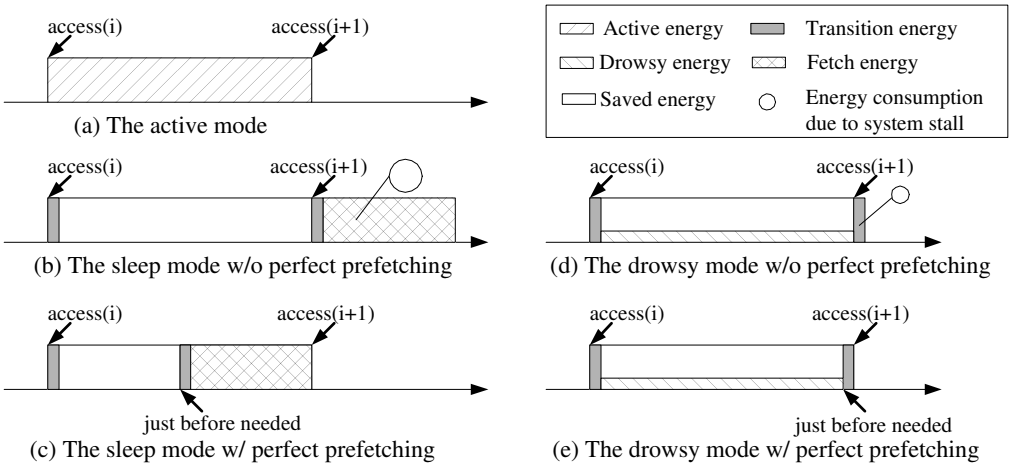
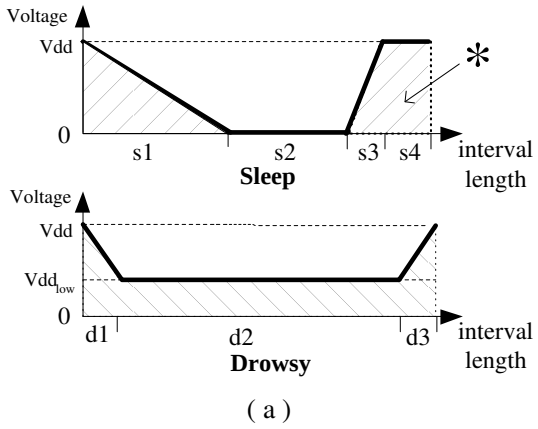


Figure 4.4: Using perfect prefetching to avoid performance degradation. Assuming perfect access pattern knowledge, an optimal method uses perfect prefetching to refetch data just before it is needed and avoids stalling the whole system to reduce energy consumption.

One thing to note is that our optimal method will have *no-effect* on the performance of the machine. Because we assume perfect access pattern knowledge, an optimal method can *re-fetch any needed data just before it is needed* and avoid any performance impact. By exploiting this fact we can separate out the power problems from the performance problems. Even though a just-in-time re-fetch or perfect prefetching will not affect the performance of the machine, it does have a power cost which we do consider in this paper. Figure 4.4 is used to illustrate this point. In the sleep mode, due to turning off the cache line to save leakage power, the data is not preserved. If the data is accessed again, it needs to be refetched, and this refetching process may usually take several cycles. Without just in time refetch (Figure 4.4(b)), the other parts of the whole system will have to stall for that amount of cycles, waiting for the data



Duration	*Voltage*
s1	High to off
s2	Always off
s3	Off to high
s4	High
d1	High to low
d2	Always low
d3	Low to high

* Dynamic power consumption due to an induced miss

(b)

Figure 4.5: Time-voltage diagrams of sleep-mode and drowsy-mode. In Sleep-Mode the cache line is essentially turned completely off and the power consumed drops to nearly zero. While beneficial over a long period of time, there is a more significant overhead due to re-fetch. Drowsy-Mode has a smaller overhead, but the cache line still consumes a measurable amount of power because the voltage has not been completely turned off.

to be ready. The stall will lead to significant energy consumption as the big circle indicates. Similar things happen to the drowsy mode. But the drowsy mode preserves the data and only takes a couple of cycles [52] to wake up the cache line. So, without just-in-time refetch (Figure 4.4(d)), the amount of energy the drowsy mode consumes is less than that of the sleep mode during the system stalling, which is indicated by a small cycle. By contrast, with just-in-time refetch (Figure 4.4(c) and (e)), The data will be ready when it is needed, avoiding stalling the rest parts of the whole system to wait for data to be ready, which consequently saves power. It is worth noting that our scheme calculates the optimal power savings for a given replacement policy, it does not change the replacement policy to further save power.

For the convenience of illustrating how our method works in general, we will use (Figure 4.5(a) and (b)) to show how the inflection points are calculated. Figure 4.5(a) shows that the sleep mode and the drowsy mode require time to reduce the voltage from high V_{dd} to off (s_1) and from high to $V_{dd_{low}}$ (d_1), respectively. Also, there is a similar time overhead in coming out of the mode (s_3 or d_3). For the sleep mode, since the latency D of fetching data from L2 cache is longer than s_3, there is another overhead ($s_4 = D - s_3$) before the next access. We divided the life time of an interval into several durations to illustrate these overheads. Figure 4.5(b) shows the length of each duration s_1, s_2, s_3, s_4, d_1, d_2, d_3 in an access interval of both a sleep mode and a drowsy mode. The total length of the cache access interval using the sleep technique is $s = s_1 + s_2 + s_3 + s_4$, and that of using the drowsy mode is $d = d_1 + d_2 + d_3$.

For the sleep mode, the data has been lost due to an induced miss [49] and must be re-fetched from the memory hierarchy. as such, there is a significant amount of power consumed by the dynamic activity required to fetch the data from the L2 cache, marked with "*" in Figure 4.5(b). This dynamic power cost can be obtained from analytical models, such as the interconnect model based on logical effort [7] or the CACTI [99] model, which has been used in this paper.

The sleep-drowsy inflection point is derived as the access interval length when the sleep and the drowsy modes consume the same amount of energy. If the interval is of a length less than the inflection point then drowsy mode would be optimal. If it is greater than the inflection point then sleep mode would be optimal. We denote the

leakage power consumption of each cache line as P_L, which can be obtained from the HotLeakage tool [113], and the cost of dynamic power due to an induced miss for the sleep mode as C_D. The energy of a sleep mode interval can be calculated as Equation 4.1:

$$E_S = \sum_{i=1}^{4} P_L(s_i) * s_i + C_D. \tag{4.1}$$

Similarly the energy consumption using the drowsy model can be calculated as Equation 4.2:

$$E_D = \sum_{i=1}^{3} P_L(d_i) * d_i. \tag{4.2}$$

When the two modes consume the same amount of energy, we reach Equation 4.3:

$$E_S = E_D. \tag{4.3}$$

Applying the data in Figure 4.5(b) into Equation 4.3, we can calculate the sleep-drowsy inflection point.

The other inflection point is between drowsy and active modes. The drowsy-active inflection point is calculated as the sum of the durations d_1 and d_3, within which the voltage changes either from V_{dd} to $V_{dd_{low}}$ or from $V_{dd_{low}}$ to V_{dd}.

Note that the sleep-drowsy inflection point is the point at which sleep mode has the potential to save power of drowsy mode. Sleep mode does not provide benefit at small interval lengths because of the larger penalty associated with coming out of sleep mode (the power of re-fetch) as opposed to drowsy mode. The only way to save power on small interval lengths is to know exactly when the cache line will be accessed again

so that it can be brought out of sleep mode before the data is needed. This is how an optimal leakage management scheme would take advantage of it's perfect knowledge.

When an interval between two accesses to the same cache line is longer than the sleep-drowsy inflection point, using sleep mode has the potential to save more leakage power. When an interval is less than the sleep-drowsy inflection point but still greater than the active-drowsy inflection point, the drowsy mode saves more leakage. When its interval length is less than the active-drowsy inflection point, the cache line is always active and cannot have its leakage power reduced without causing a delay in delivering the data.

Figure 4.1 details our optimal leakage power saving method. By classifying cache intervals into the three types and applying to them the appropriate leakage saving mode, the maximal leakage power saving can be obtained as the accumulation of the leakage saving over all access intervals, which provides us an upper bound for optimal leakage power savings. It can be proved that based on the perfect knowledge of the lengths of all intervals, the optimal leakage power saving can be achieved by applying the proper operating mode on each interval.

4.3.3 Theorem of Optimal Policy for Leakage Power Saving

In this section, after defining the relevant terms in our study, we provide the theorem of the optimal policy for leakage power saving.

```
ALGORITHM OPTIMAL_LEAKAGE
    Input: A set of intervals I
    Output: Total leakage power saving total_saving
BEGIN
1   total_saving := 0
2   i := 0
3   while (I_i ∈ I) do
4     if (|I_i| > b) then
5       total_saving := total_saving + sleep_saving(|I_i|)
6     else if (|I_i| > a) then
7       total_saving := total_saving + drowsy_saving(|I_i|)
8     else
9       no leakage power saving can be obtained
10      i := i + 1
11    end do
12    return(total_saving)
END
```

Table 4.1: Algorithm to compute the optimal leakage power saving given an interval distribution. Intervals are classified into one of the three categories based on the drowsy-active inflection point a and the sleep-drowsy inflection point b: $(0, a]$, $(a, b]$, and $(b, +\infty)$. The *Sleep mode* is applied on intervals within the range of $(b, +\infty)$; the *Drowsy mode* is applied on intervals within the range of $(a, b]$; and the cache lines are left on for intervals within the range of $(0, a]$.

Definition 4.3.3.1. *We define* **I**=$\{I_i\}$ *as a set of intervals, and the length of interval* I_i *as* $|I_i|$ *(* $|I_i| \in (0, +\infty)$ *).*

Definition 4.3.3.2. *For each interval* $I_i \in$ **I**, *we define three possible operating modes* $T_j \in \mathbf{T}, where \mathbf{T} = \{T_1 = active, T_2 = drowsy, T_3 = sleep\}$, *and the leakage energy saving of the interval* I_i *working in the mode of* T_j *is defined as* $E(I_i, T_j)$.

Definition 4.3.3.3. *We define two inflection points, the active-drowsy inflection point* a *and the sleep-drowsy inflection point* b. *The active-drowsy inflection point* a *is defined as the sum of the durations within which the supply voltage changes either from high*

to low or from low to high, i.e. $a = d_1 + d_3$*. The sleep-drowsy inflection point* b *is defined as the access interval length when the sleep and the drowsy modes consume the same amount of energy, i.e.* $b = s_1 + s_2 + s_3 + s_4 = d_1 + d_2 + d_3$*, where* $s_i \geq 0$*,* $d_j \geq 0$ *(*$i = 1, 2, 3, 4$*;* $j = 1, 2, 3$*).*

Lemma 4.3.3.4. *The active-drowsy inflection point* a *is less than the sleep-drowsy inflection point* b*.*

Proof. Because loads have physical capacities, the discharging process takes less time for the voltage dropping from high to low than from high to off, i.e. $d_1 < s_1$. Similarly, the charging process takes less time for increasing the voltage from low to high than from off to high, i.e. $d_3 < s_3$. Since there is no overlapping time between s_i and s_k ($i \neq k$, $i, k = 1, 2, 3, 4$) and $s_i \geq 0$ ($i = 1, 2, 3, 4$), we can conclude that the sum of the durations $a = d_1 + d_3 < s_1 + s_3$, and the sleep-drowsy inflection point $b = s_1 + s_2 + s_3 + s_4$ is greater than $s_1 + s_3$. So, a is less than b. Our study based on the 70nm technology process also justifies that a (6 cycles) is less than b (1057 cycles) from the experimental perspective. □

Theorem 4.3.3.5. *Under the context of the independent model, where access intervals of a cache block are independent from each other, we assume that for each interval* $I_i \in \mathbf{I}$*, one and only one of the three operating modes* $T_j \in \mathbf{T}$ *can be applied for reducing leakage energy consumption based on the following policy:*

1. *When the interval length* $|I_i| \in (0, a]$*, the active operating mode or non-power saving mode is applied.*

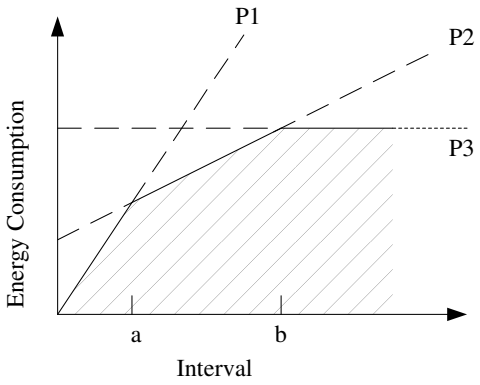


Figure 4.6: Energy consumption for each of the three operating modes and the lower envelope $E(I_i, T_j)$ function for minimal energy consumption.

2. *When the interval length $|I_i| \in (a, b]$, the drowsy mode is applied.*

3. *When the interval length $|I_i| \in (b, +\infty)$, the sleep mode is applied.*

*Then the maximal leakage saving can be obtained as the combination of the power saving over all intervals $I_i \in$ **I**, which gives an upper bound for optimal leakage power saving.*

Proof. We prove the theorem by contradiction. We divide the whole range of the interval length $(0, +\infty)$ into three independent portions based on the active-drowsy inflection point a and the sleep-drowsy inflection point b, i.e. $(0, a] \cup (a, b] \cup (b, +\infty)$ (see Lemma 1 that $a < b$). Suppose the energy saving M based on the above assumptions is not maximal, then there must be another energy saving M' that is greater than M, which indicates that there is at least one interval I_i whose operating mode T_j' is different from T_j.

Figure 4.6 shows the function of interval vs. energy consumption. In the figure, we can have the following derivations:

1. The function is continuous and monotonically increasing.

2. The slopes P_1, P_2 and P_3 indicate the power consumptions within the interval ranges of $(0, a]$, $(a, b]$ and $(b, +\infty)$ respectively.

3. For intervals in the range of $(0, a]$, the minimal energy consumption can be achieved through the active mode T_1. For intervals in the range of $(a, b]$, the minimal energy consumption can be achieved through the drowsy mode T_2. For intervals in the range of $(b, +\infty)$, the minimal energy consumption can be achieved through the sleep mode T_3.

For a set of independent intervals, if at least one interval I_i was applied with T'_j, not the corresponding mode T_j, then $E(I_i, T'_j)$ is greater than $E(I_i, T_j)$ (above the shadow area in Figure 4.6), giving the contradiction. Therefore, the maximal leakage power saving can be obtained by the proposed policy. □

4.3.4 Generalized Model for Optimal Leakage Power Savings

After illustrating our optimal leakage power saving method, we evolve our method to a complete model that can capture the optimal leakage savings as configurations and technologies change.

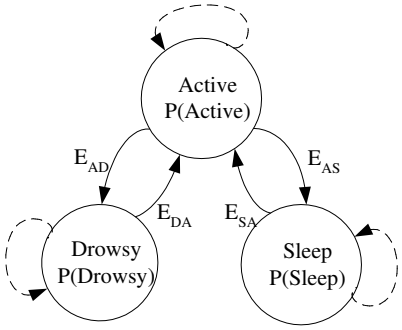


Figure 4.7: The optimal leakage power saving model. The circles indicates states and edges represent transitions between states.

As Figure 4.7 depicts, the model has three states, which are indicated by the circles, representing the three operating modes: Active, Drowsy, and Sleep. It also models the transitions between states, demonstrated by the edges (self edges means that the state remains the same in the next cycle). Each state is associated with its static power consumption (P), and the weights (E_{AD}, E_{AD}, E_{AD}, E_{AD}) on the edges are the transition energy consumptions. For example, E_{AD} is the energy consumption when transiting from the state Active to the state Drowsy.

In the model, all the individual assumptions namely the durations, energy costs of transitions between modes, the leakage power consumption of each mode, and the intervals, are parameterized and used as inputs to the model. And the outputs of the model are the optimal leakage saving percentages of using the optimal sleep, optimal drowsy, and the optimal combining methods.

The model has been designed to explore the optimal leakage savings with parameterized architectural and design considerations, i.e. if the architectural configuration

changes(for example, cache configurations), the model can adapt to the input change, and if a new low power mode is employed, the model can be easily extended by adding a new state.

The model for optimal leakage power savings serves two major functions. First, instead of being an abstract model, it is coded in C language and is publicly available for cache leakage studies[1]. To use the tool, designers only need to feed the following inputs into the C program, such as the transition energies obtained from CACTI [99], the leakage power consumption from HotLeakage [113], the interval distribution from SimpleScalar [23], and the duration parameters $\{s_1, s_2, s_3, s_4\}$. The tool will then find out an optimal mode transition sequence that can achieve the maximal leakage power savings and output the optimal leakage saving percentages of using the optimal sleep, the optimal drowsy, and the optimal combining methods. Second and the most important is that this model was designed to explore the optimal leakage savings under different architectural and design assumptions with the hope of guiding research effort on leakage power study.

Concurrent to our work, similar approaches are being developed to guide policy decisions in the domain of Computer Aided Design [70]. In [70], the optimal energy mode transition sequence for generic devices is calculated under a fixed delay constraint. Because of the complex timing involved in a modern superscalar microprocessor, a simple timing model will not accurately reflect the impact changing cache

[1] http://express.ece.ucsb.edu/software/leakage.html

parameters. In our study on cache leakage we have factored out any performance impact through perfect prefetching, which acts as a guiding bound for those developing leakage reduction schemes. We have built our model around the state of the art in leakage reduction techniques and show how the most common techniques considered today can be mapped onto this simple framework quickly and precisely. In addition, we have given a formal proof of optimality, which has not been offered in their work.

4.4 Empirical Study

In Section 4.3, we discussed the limits of leakage power reduction techniques and how they are calculated. In this section we show limit results gathered from actual benchmarks with parameters extracted from modern processors and prior work. Our objective is to evaluate the limits on some leakage power saving techniques as applied to both the L1 instruction and data caches, and the L2 cache. We show upper bounds on the possible savings using Sleep mode, Drowsy mode, or a potential hybrid of the two. We also evaluate the generality of the parameterized model in deriving the limits of leakage power savings from the perspectives of different implementation technologies and different cache configurations.

4.4.1 Methodology

To test the amount of power that can be saved by using an improved leakage reduction technique, we employed detailed cycle-level simulation. The simulator we use is a version of SimpleScalar closely resembling Compaq Alpha 21264 [50]. The execution core is a 4-wide superscalar pipeline, and the memory hierarchy includes a 64KB, 2-way set associative L1 instruction cache with a single-cycle hit latency, a 64KB, 2-way set associative L1 data cache with a 3-cycle hit latency, and a unified 2MB direct-mapped L2 cache with a 7-cycle hit latency. The main memory system consists of 16 32MB DDR2 SDRAM chips [93], for a total main memory capacity of 512MB, with its access time 40ns, and access power 300mW. Because leakage is exponentially dependent on temperature, we use 85°C in our experiments. LRU is employed as the replacement policy throughout the memory hierarchy. To calculate inflection points, we assumed a 500MHz processor clock.

In order to capture the most important program behaviors while at the same time reducing simulation time to reasonable levels, we used the simulation points that were described and verified in SimPoint [98]. The benchmark suite for this study consists of all the SPEC2000 benchmarks compiled for the Alpha AXP ISA. Because modern processors typically have two levels of on-chip caches (e.g. separate L1 instruction and data caches and a unified L2 cache), in the rest of our empirical study, we will first explore the limits of the leakage power savings for the L1 instruction and data caches. We will then conduct the limits study for the L2 caches.

4.4.2 Limits Study for L1 Instruction and Data Caches

In this section, we explain how the optimal method works from the experimental aspect in exploring limits of leakage power savings for the modern L1 caches by the ways of how we calculate the inflection points, and how we optimally combine the sleep and drowsy modes.

Calculating Inflection Points

Inflection points are the keys to choosing the best power saving mode of a given interval distribution. An intervals with its length greater than the sleep-drowsy inflection point is put into the sleep mode, and an interval with its length less than the active-drowsy inflection point is left in the active mode. For an interval with its length between the two inflection points, it is put into drowsy mode to save power and has little performance impact.

Technology	70nm	100nm	130nm	180nm
Active-Drowsy point	6	6	6	6
Drowsy-Sleep point	1057	5088	10328	103084

Table 4.2: Active-drowsy and drowsy-sleep inflection points depicted in cycles for different technologies.

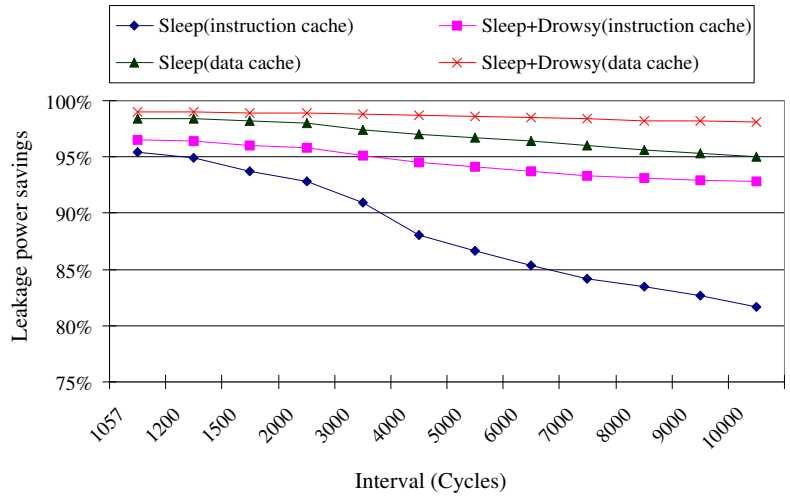
To calculate inflection points with respect to different technologies, we used the durations s_1=30, s_3=d_1=d_3=3 and s_4=4 cycles [66] (s_2 and d_2 are dependent on an interval length). When we applied the parameters into Equation 4.1, 4.2 and 4.3, we obtained the inflection points for the L1 instruction and data caches shown in Table 4.2. The

table shows that the value of the sleep-drowsy point decreases while the technology scales down from 180nm to 70nm (These are the only currently available technologies provided by the Hotleakage [113] tool. If in the future Hotleakage is extended to incorporate more technologies, our method can still be applied to obtain their inflection points). This is due to the fact that the leakage power consumption per cache line increases while the dynamic energy consumption caused by an induced miss decreases with technology scaling down (see Equation 4.3).

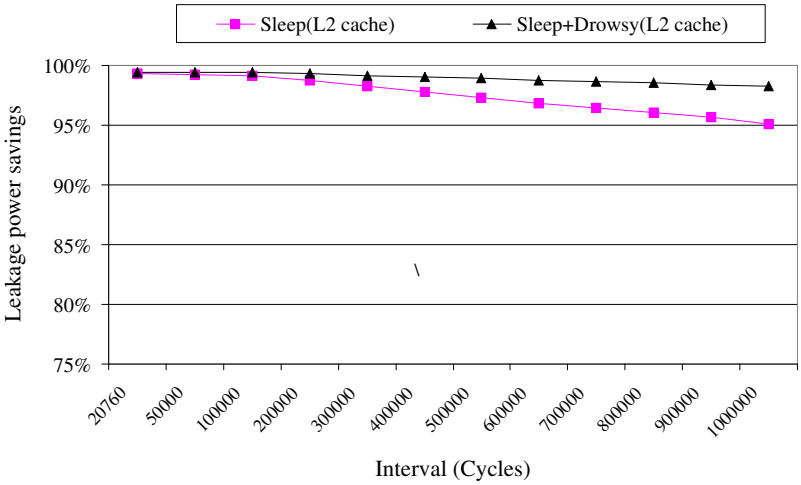
Since 70nm is the most advanced technology that will be reached in a few years according to ITRS [45], we employed it and its corresponding sleep-drowsy inflection point (1057 cycles) in the rest of our study on the L1 caches.

Combining Sleep and Drowsy Modes

With the inflection points calculated, the first question to be answered is how well a hybrid of sleep and drowsy modes can perform versus sleep mode. If, in the optimal case for sleep mode, we can perfectly predict the distances between access to cache lines then we can potentially make use of sleep mode even if the cache line is accessed every 1057 cycles. In this case, there will be little benefit from using drowsy mode for those cache lines that are accessed more frequently than every 1057 cycles. However, if the threshold was different, if the inflection point between drowsy and sleep modes changed dramatically, there would be a point at which using both drowsy mode (for occasionally accessed line) and sleep mode (for rarely accessed lines) would become



(a) L1 Instruction and Data Caches



(b) Unified L2 Cache

Figure 4.8: Comparison of the hybrid method vs. the sleep-mode method for different sleep interval-lengths. The usefulness of applying the drowsy method to save leakage power decreases as the sleep length approaches the sleep-drowsy inflection point. For leakage power saving, the sleep mode plays a more important role in the L2 cache and the data cache than in the instruction cache. The L2 cache has larger sleep intervals than the data cache.

beneficial. The purpose of Figure 4.8 is to demonstrate this point. In our experiments, when a sleep mode is applied, the dynamic power consumption due to an induced miss was removed from the total leakage power savings.

The results in Figure 4.8(a) are derived based on the average leakage power savings for both instruction and data caches across all the given benchmarks. Through this figure, we examine the potential effectiveness of a pure sleep mode versus a hybrid sleep/drowsy method where we change the minimum interval length that can be put into sleep mode from 1057 to 10000. These results indicate that a hybrid method (Sleep+Drowsy) can work consistently better than the sleep or the drowsy method alone, especially if one is very conservative about which lines are put to sleep. However, as the minimum sleep length approaches the sleep-drowsy inflection point (decreases), the usefulness of applying the drowsy method in addition to the sleep mode decreases. Under such conditions, the sleep mode removes most of the leakage power and thus there is not much more for drowsy to save. While clearly an implementable scheme will not have the luxury of perfect future knowledge, for those that we do have knowledge for, sleep mode should be applied very aggressively.

Moreover, the figure depicts that the gap between the hybrid method and the sleep mode for the data cache is much smaller than that for the instruction cache. The reason is that the same cache block in the data cache tends to be less frequently accessed than in the instruction cache, and the interval-lengths between consecutive accesses are

much longer. Hence, the sleep mode plays a much more important role in the data cache for the leakage power saving than in the instruction cache.

Finally, this figure also confirms that the small variances of the sleep-drowsy inflection point will not change our findings significantly.

Exploring the Upper Bound

With the assumption of perfect-prefetching, the upper bound of the leakage power saving was derived in Section 4.3 based on the two inflection points. We now explore the leakage-power-saving limits of the following methods assuming perfect knowledge of the future address trace:

1. *OPT-Drowsy*: An optimal drowsy cache that has no performance penalty for waking up data (although there is a power penalty as discussed in Section 4.3).

2. *OPT-Sleep(10K)*: An optimal cache line sleeping technique that puts to sleep all intervals of a size greater than 10K with no performance penalty.

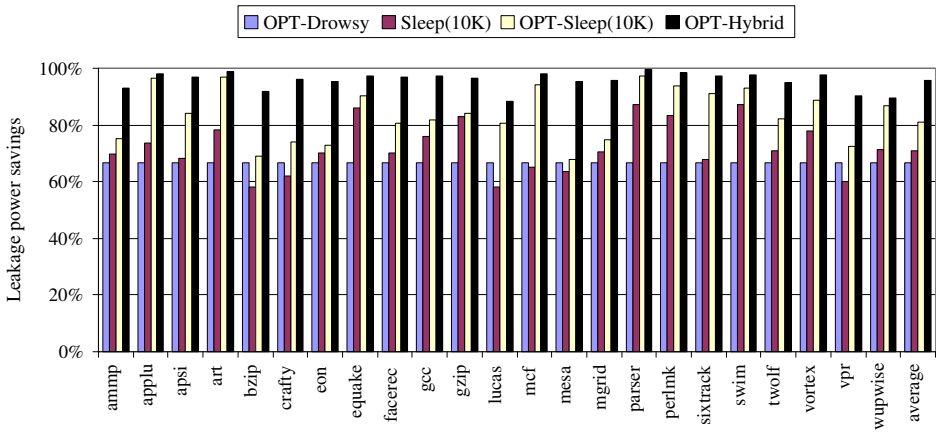
3. *Sleep(10k)*[2]: Similar to the OPT-Sleep(10K) with the exception that instead of optimally turning off any cache line that has an interval larger than 10K, the line must now stay active for 10K and then may be optimally slept.

4. *OPT-Hybrid*: The method that optimally combines drowsy and sleep modes based on the inflection points without any performance penalty.

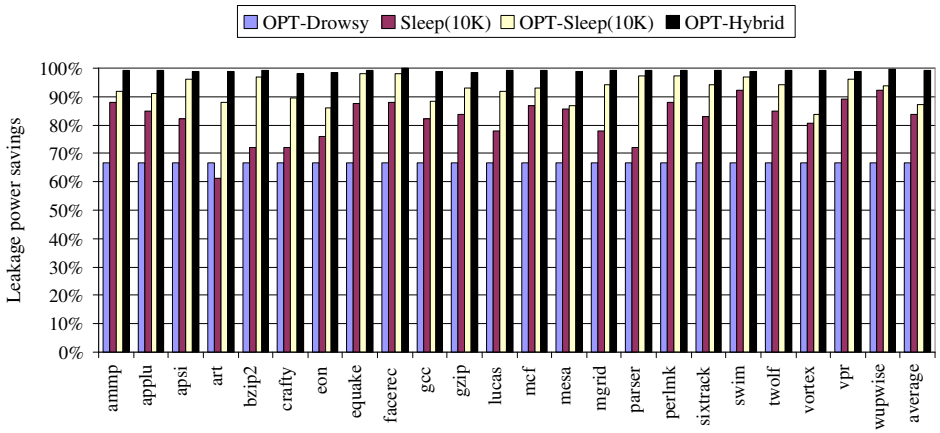
[2] *The sleep(10K) is similar to the cache-decay scheme in [49], in which the decay interval was set to be 10K cycles, and the extra leakage power consumed by the counter per cache line was taken into account.*

For the convenience of discussing the implementation of each technique, we define an access interval of a cache line as T_i. OPT-Drowsy puts the line into the drowsy mode during T_i, if T_i is greater than 6; while OPT-Sleep(10K) puts the cache line into the sleep mode during T_i if T_i is greater than 10K. We also studied Sleep(10K) to simulate the cache-decay scheme, whose decay interval is 10K. In this case, a cache line is put into the sleep mode for (T_i-10K) cycles if T_i is greater than 10K. The OPT-Hybrid is to put a cache line into the sleep mode during T_i if T_i is greater than 1057, and to put it into the drowsy mode if T_i falls into the range of (6, 1057]. When T_i is less than 6, all the above methods keep the cache line active to insure fast access time. When a sleep mode applied, the dynamic power consumption due to an induced miss was removed from the total leakage power savings.

Figure 4.9 depicts the percent of leakage power in comparison with a cache with all its cache lines constantly active for each benchmark application. Figure 4.9(a) shows that for the instruction cache, the limit of leakage power saving that OPT-Hybrid can achieve is 96.4%. It is 26% higher than Sleep(10K), 16% higher than OPT-Sleep(10K), and 30% higher than OPT-Drowsy. For the data cache (Figure 4.9(b)), the leakage power saving limit is 99.1%, which is 15% higher than the Sleep(10K), 12% higher than the OPT-Sleep(10K), and 33% higher than the OPT-Drowsy. The results indicate that while the initial Drowsy and Sleep techniques devised are quite effective, there is still far more potential left in these techniques. Indeed, the leakage power savings for the optimal case are so large that it is fair to say that leakage power would become an



(a) Instruction Cache



(b) Data Cache

Figure 4.9: Comparisons of different leakage power saving schemes.

insignificant portion of the total overall power if these savings could be realized. All these savings could be realized with new policies for cache management. Of course realizing these optimal numbers requires perfect knowledge of the address trace and timing, which is not typically possessed by a management policy.

4.4.3 Limit Study for Unified L2 Cache

In modern processors, a large number of the total on-chip transistors is consumed by caches, particularly L2 caches. Different from L1 caches that are optimized for performance, L2 caches are optimized for density and stability considering both yield and process parameter variation, and its memory-cell size is usually smaller than a L1 SRAM cell. The area overhead of implementing either drowsy or sleep technique in L2 caches is about 5-8%, and sleep mode may cause instability of L2 memory cells[52]. However, as machines and working-sets grow, the L2 caches are becoming increasingly performance critical, and integrated even closer to the processor (especially when a third level of hierarchy is added). In the future it is likely that L2 caches will be heavily accessed by multiple processors, and will be close enough to the logic that they may well be affected by heat dissipation problems. For example even today, the L2 cache covers 37% of the alpha 21364 chip area and contains 85% of the total devices [33]. In addition, L2 caches have much larger miss penalty than L1 caches, because they have to go to main memory to fetch data when L2 misses happen. Thus, without careful attention to power, L2 caches may overwhelm the chip's power bud-

get. In this section, we study the leakage reduction in L2 caches when both sleep and drowsy modes are employed.

For applications where the access frequency of L2 caches is small, using high-Vth transistors in memory cells will significantly reduce leakage power. Yet, for applications with large code and data footprints, more levels of cache hierarchy will be needed to take advantage of locality, e.g. the Intel Madison processor [71] has L2 caches which are frequently accessed. In those applications, solely applying high-Vth transistors in memory cells will reduce leakage but may lead to significant performance impact, and dynamic management polices could be employed to trade off power and performance. In this work, we assume two level caches, while our optimal method can be extended to study general multilevel caches. With the most advanced 70nm implementation technology, we calculated the optimal sleep-drowsy inflection point for the L2 cache as 20760 cycles, and the drowsy-active point as 6 cycles.

Figure 4.8(c) shows the average leakage power savings of all the benchmarks. It demonstrates how well the hybrid of sleep and drowsy modes can perform versus sleep mode. From the figure, we can see that if we can perfectly predict the access intervals of the L2 cache, there will be little benefit from using drowsy mode for those cache lines that are more frequently accessed than 20760 cycles. Figure 4.8(c) looks similar to Figure 4.8(b), however, they are different in scales. The L2 cache has much larger sleep-drowsy inflection point than the data cache(Figure 4.8(b)), since the miss penalty of the L2 cache is much larger than that of the on-chip data cache.

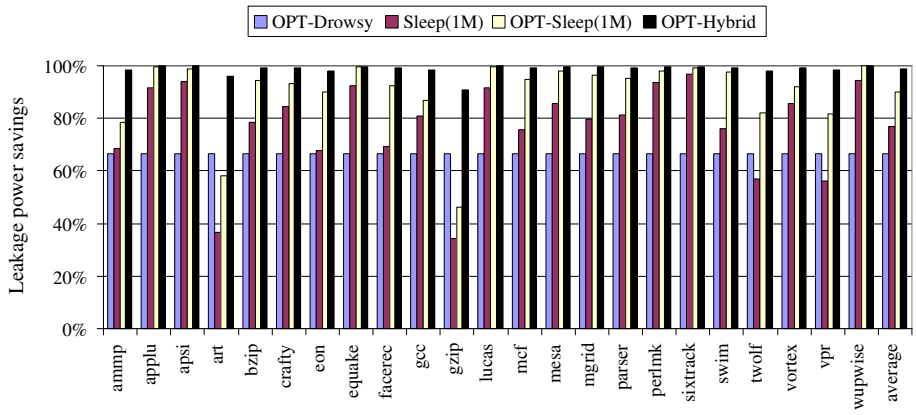


Figure 4.10: Comparisons of different leakage power saving schemes for the direct-mapped L2 cache.

Figure 4.10 shows the comparison results of the four methods across all the benchmarks, *OPT-Drowsy*, *OPT-Sleep(1M)*, *Sleep(1M)*[3], and *OPT-Hybrid*. The figure shows that for the L2 cache, the limit of leakage power saving that OPT-Hybrid can achieve is 97.7%. It is 21% higher than Sleep(1M), 7.6% higher than OPT-Sleep(1M), and 31% higher than OPT-Drowsy, which indicates that there is still far more potential left in the existing techniques.

4.4.4 Empirical Study with Generalized Model

Also, we evaluate the generality of the proposed parameterized model from two perspectives, one with different implementation technologies and the other with different cache configurations.

[3]*The sleep(1M) is similar to the cache-decay scheme in [49], in which the decay interval was set to be 1M cycles, and the extra leakage power consumed by the counter per cache line was taken into account.*

	Technology	70nm	100nm	130nm	180nm
	Vdd (V)	0.9	1.0	1.5	2.0
	Vth (V)	0.1902	0.2607	0.3353	0.3979
I-Cache	OPT-Drowsy (%)	66.4	66.6	66.6	66.7
	OPT-Sleep (%)	95.2	85	80.6	61.5
	OPT-Hybrid (%)	96.4	93.7	91.3	67.1
D-Cache	OPT-Drowsy (%)	66.1	66.6	66.7	66.7
	OPT-Sleep (%)	98.4	96.9	95.3	63.2
	OPT-Hybrid (%)	99.1	98.1	97.3	67.3

Table 4.3: Optimal leakage saving percentages with technology scaling down.

Evaluating the Generality of the Parameterized Model with Different Implementation Technologies

To show the generality of the parameterized model, we also study the L1 instruction and data caches with 100nm, 130nm and 180nm processes. Table 4.3 summarizes the optimal leakage saving percentages we can possibly achieve by using OPT-Drowsy, OPT-Sleep, and OPT-Hybrid methods for each of these technologies. Instead of using OPT-Sleep(10K) on intervals that are greater than 10K cycles, we study OPT-Sleep to figure out what is the best leakage power saving we can achieve by aggressively turning off all intervals that are greater than the sleep-drowsy inflection point. The OPT-Drowsy and OPT-Hybrid methods are the same as before. The results in the table are the average results over all the benchmark applications.

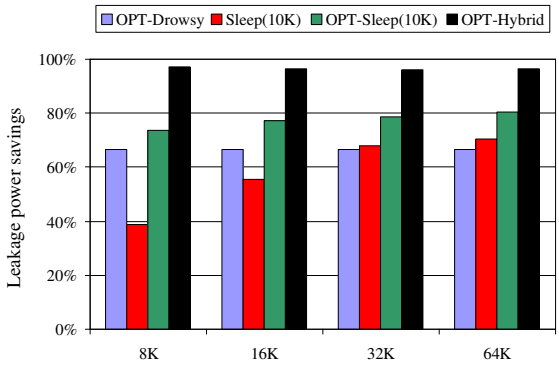
The table illustrates that the leakage savings for both the instruction and the data caches of using OPT-Hybrid increase with the technology scaling down from 180nm to 70nm. The increment of the possible leakage savings is due to the decrement of the

sleep-drowsy inflection point. Moreover, the table shows that for the 180nm technology implementation, the drowsy mode plays a more important role in saving leakage power than the sleep mode does; while for the others, the sleep mode plays a leader role. This can be also attributed to the large difference of the sleep-drowsy inflection points. Finally, the table also reveals that more leakage savings can be possibly achieved with the technology scaling down, which leaves us more space for further improvement on leakage power savings.

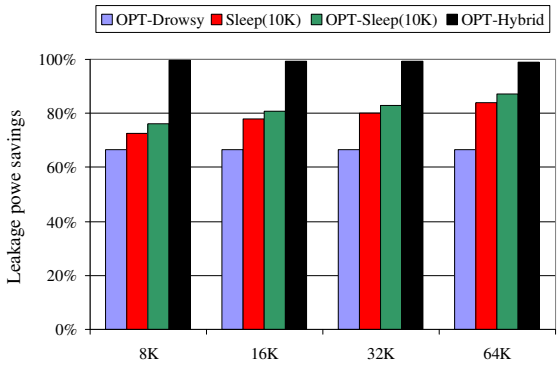
Evaluating the Generality of the Parameterized Model with Different Cache Configurations

To further evaluate the generality of the parameterized model on studying the limits of leakage power reduction, we also conducted experiments on L1 caches with different sizes, while the rest of the configurations remain the same. Specifically, we studied 8KB, 16KB, and 32KB 2-way set associative instruction and data caches with one-cycle latency.

Figures 4.11 shows the on-average results of different L1 caches. The results demonstrate that as the L1 cache sizes grow larger, the percentage of the leakage power saving for each scheme (OPT-Drowsy, Sleep(10K), OPT-Sleep(10K) and OPT-Hybrid) increases, and the drowsy mode plays a more important role for smaller caches than the sleep mode does. This is due to the fact that using smaller caches usually results in more frequent cache misses, which lead to many small intervals (less than 10K cy-



(a) L1 Instruction Caches



(b) L1 Data Cache

Figure 4.11: Comparison of different leakage power saving schemes for L1 caches with different sizes.

cles) due to frequent replacements. Because of the existence of the small intervals, the drowsy mode shares a significant portion in reducing leakage power. In addition to study the leakage problem with different L1 cache sizes, we also experimented with another configuration that has a different L2 cache configuration.

This configuration includes a 32KB 2-way set associative instruction cache with a single-cycle hit latency, a 32KB 2-way set associative data cache with a two-cycle

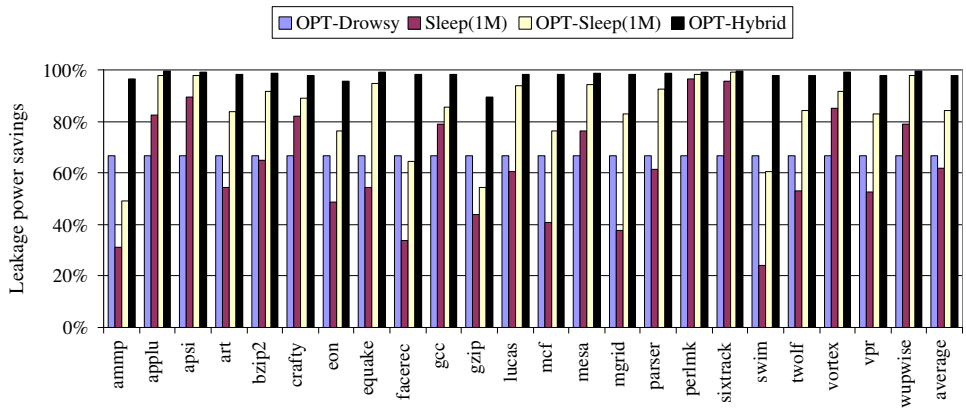


Figure 4.12: Comparisons of different leakage power saving schemes for the new 2-way associative L2 cache.

hit latency, and a unified 1M 2-way set associative L2 cache with five-cycle hit latency. The rest of the configuration, such as the main memory, replacement policy and implementation technology, are the same as the previous configurations.

Figure 4.12 shows the comparison results of the smaller L2 cache for different leakage power saving techniques. For the purpose of comparison with the previous cache configuration, the results here also include the results of all benchmarks and their average. Note that due to the different cache configurations, the baseline of the total leakage power consumption of the new configuration (Figure 4.12) is different from that of the previous configuration (Figure 4.9), even though the y-axis also indicates the percent of total leakage power of which all lines remain constantly active. In our experiments, we found that the total power averaged over all benchmarks for the previous configuration was twice that of the new configuration.

From Figure 4.9 and Figure 4.12, we can make three observations. First, for OPT-Drowsy, Sleep(1M) and OPT-Sleep(1M), there are still much room left for further exploring circuit and architectural techniques to achieve the maximal leakage power savings that OPT-Hybrid provides. Second, the percentages of leakage power savings for Sleep(1M) and OPT-Sleep(1M) on the new L2 cache are smaller than those on the previous L2 cache. Third, the results of *ammp* and *mcf* clearly show that OPT-Drowsy achieves more leakage power savings than Sleep(1M) and OPT-Sleep(1M) with the new configuration, while OPT-Drowsy saves less leakage power than either Sleep(1M) or OPT-Sleep(1M). The explanation for the above observations is that the new cache configuration has smaller L1 caches than the previous configuration, which will generally result in a larger number of accesses to the L2 cache and smaller access intervals for the L2 cache lines. Since most of the power savings are from the long intervals, the new cache configuration consequently only achieves less power savings than the previous configuration. This confirms that the parameterized model can be generalized to provide limits of leakage power reduction for different cache configurations.

We would also like to mention that we had an initial exploration [75] of using a form of prefetching, such as next-line and stride-based techniques, to approximate the optimal. The goal of prefetching is to accurately predict future access patterns so that they can optimistically fetched from memory before their use. We propose that prefetching can optimistically re-fetch data that has been either turned off for sleep mode or put into a drowsy state. In our trial study, we delivered the information of

what is the best those employed prefetching techniques can help us to approach the optimal, and we thus did not take into consideration the overhead of prefetching. Our evaluation of the potential usefulness of next-line and stride-based prefetching toward reducing leakage power on the L1 instruction and data caches shows that if prefetching can be used to guide sleep mode and drowsy mode is used the other times then the leakage power dissipation will be within a factor of 2.5 from the optimal.

4.5 Approximating Perfect Knowledge With Prefetching

Figure 4.9 also reveals that even though OPT-Sleep(10K) can achieve a good amount of leakage power saving, there is still great potential to approximate the optimal leakage saving that OPT-Hybrid can accomplish. For instruction cache, the gap is 14.4%, and for data cache, 12.1%. In this section, we will study how prefetching can be employed to approach the optimal in the real situations where perfect knowledge of the future address is unknown.

4.5.1 Approximation of Perfect Knowledge

In Section 4.3, we provide the upper bound of leakage saving, yet it is derived assuming perfect knowledge of the timing of the address trace. This information is used to insure that no additional latency overhead will be added to the next access

of the cache line for either scheme because performance overhead can be perfectly overlapped with the length of the interval (see Figure 4.4.) As a result, there will be no extra stalls that result in leakage power consumption of the whole system.

While the upper bound on the amount of savings that can be achieved is a useful notion to characterize, an actual implementation will require that we approximate this perfect knowledge. Luckily, such a set of future address predictions is already well studied in a different area of processor research — prefetching. The goal of prefetching is to accurately predict future access patterns so that they can be optimistically fetched from memory before their use. We propose that prefetching can optimistically refetch data that has been either turned off for sleep mode or put into a drowsy state. This would allow a more aggressive sleep mode policy as the penalty that is normally associated with sleeping a line can be fully or partially hidden by the approximated future knowledge that prefetching provides.

Prefetching is one of the primary techniques used in modern processors to tolerate memory latency. Many models have been proposed for prefetching either instructions or data to reduce latency, ranging from compiler-based prefetching [78] to hardware-based prefetching [91]. In this paper, we examine the potential use of two hardware based schemes to capture simple but frequent forms of misses: next-line prefetching and stride-based prefetching.

The idea behind next-line prefetching is that a cache miss will likely be followed by one or more cache misses to consecutive cache blocks. Because programs exhibit a

great deal of spatial locality, this form of prefetching is very effective in hiding latency, and meanwhile is fairly straightforward to implement.

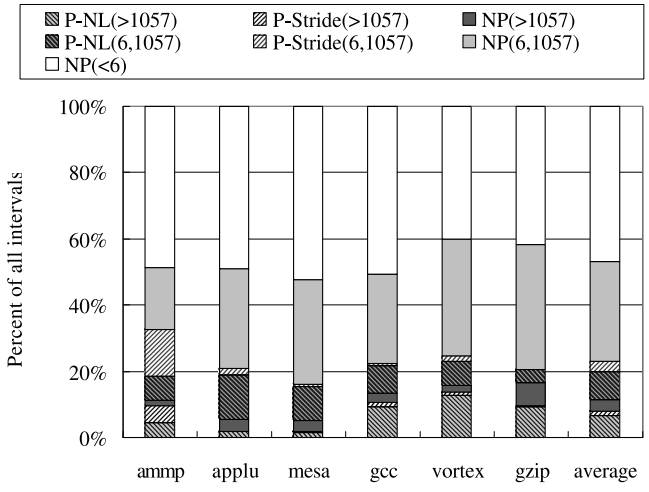
Stride-based prefetching is to eliminate miss patterns that follow a regular pattern but access non-sequential cache blocks. This type of accesses frequently occurs in programs that use multidimensional arrays. To capture this type of misses, Farkas et al [29] showed that the most efficient way is by examining the access patterns on a per static load basis. A miss is taken as stride miss if the same stride has been seen at least twice for the static load accesses.

In this study, we employ the next-line prefetching technique for the instruction cache, and both next-line and stride-based techniques for the data cache, since most of the cache misses can be captured by these schemes based on the study in [91].

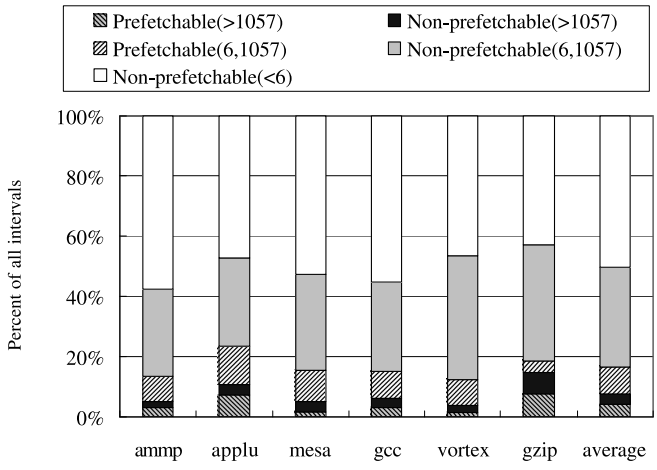
4.5.2 Approaching Limits of Leakage Power Savings

With the attempt to approach the upper bounds, we took the access intervals within which one or more accesses to the previous cache line occurs as prefetchable (P) and the other intervals as non-prefetchable (NP). Since the intervals that are less than 6 are always kept active and there is no need to prefetch them, they are counted as non-prefetchable. The prefetchability is deduced as the number of prefetchable intervals over the total number of intervals.

Figure 4.13 shows the prefetchability of cache accesses. We divided the intervals into three types, $(0, 6]$, $(6, 1057]$, and $(1057, +\infty)$, based on their lengths. The shaded



(a) Instruction Cache



(b) Data Cache

Figure 4.13: Prefetchability of intervals with different lengths.

Prefetch-A		
Interval	Prefetchable	Nonprefetchable
(1057, $+\infty$)	Sleep	Active
(6, 1057]	Drowsy	Active
Prefetch-B		
Interval	Prefetchable	Nonprefetchable
(1057, $+\infty$)	Sleep	Drowsy
(6, 1057]	Drowsy	Drowsy

Table 4.4: The Prefetch-A and Prefetch-B methods.

areas within the range of $(6, 1057] \cup (1057, +\infty)$ cover the percentage of the intervals that are prefetchable, and the other areas cover the intervals that are non-prefetchable. For the instruction cache, the prefetchability of using the next-line prefetching technique (P-NL) is 23%; and for the data cache, the prefetchabilities of using the next-line (P-NL) and the stride-based (P-stride) prefetching techniques are 16.3% and 5.1% respectively. So, the total prefetchability out of all intervals is 23% for the instruction cache and 21.4% for the data cache.

When employing prefetching to approach the upper bound, we designed two methods: Prefetch-A and Prefetch-B based on the understanding of the two fundamental objectives of pursuing high performance and low power.

1. *Prefetch-A* was designed to provide the best performance that the corresponding prefetching techniques could achieve.

2. *Prefetch-B* was designed to provide the best power saving that the corresponding prefetching techniques could achieve.

Table 4.4 details the two methods. For both of them, when an interval was identified as prefetchable, a corresponding low power mode was applied onto it. For the non-prefetchable cases, Prefetch-A keeps the cache lines always on, while Prefetch-B puts them into drowsy mode.

The limit results with prefetching are shown as the two rightmost bars in Figure 4.9. From the figure, we can have the following four observations. Prefetch-B can closely approach the optimal (within 5.3% for the instruction cache and within 6.7% for the data cache). For the instruction cache, Prefetch-A achieved 10% better than Sleep(10K) and Prefetch-B achieved 21% better than Sleep(10K); and for the data cache, Prefetct-B achieved 7% better than Sleep(10K). The leakage power difference between Prefetch-A and Prefetch-B is due to the unprefetchable intervals within the range of $(1057, +\infty)$, and it can be alleviated by employing more sophisticated prefetching techniques. Finally, the best design trade-off of power and performance is somewhere in between of the Prefetch-A and Prefetch-B methods, which will be studied in our future work.

4.6 Summary

Leakage power dissipation is quickly becoming a major concern in designing high performance processors. In this chapter we explore the limits to which known circuit level techniques can be combined and employed to save cache leakage power using new management methods and protocols. In addition, we developed a parameterized

model to determine the optimal leakage savings while the implementation technology changes over time. We find that it is possible with perfect knowledge of the future address trace to reduce the amount of power dissipated by the instruction cache down by a factor of 5.3 from known techniques (2 for the data cache, and 10 for the unified L2 cache). At this level, the leakage power of the cache would become a less serious problem. Through the evaluation of generality of the parameterized model, we found the model is robustly applicable to different caches, which will provide helpful guidance for further researches in cache-level leakage power reduction.

Chapter 5

Co-exploration for Optimizing Leakage Power Dissipation of Embedded Memories on FPGAs

5.1 Introduction

Transistor leakage is a growing problem in reconfigurable devices and will soon become the dominant source of power dissipation. FPGAs are an attractive option when implementing a variety of applications due to their high processing power, flexibility and non recurring engineering (NRE) cost. While there is some preliminary work on leakage power reduction in FPGAs, tackling the leakage problem requires solutions that consider the growing die area consumed by embedded memories, a problem which so far has been left unaddressed. In this chapter, we argue that leakage in embedded memories will be of growing importance, and we propose a leakage-aware design flow with five power saving schemes to initiate the exploration.

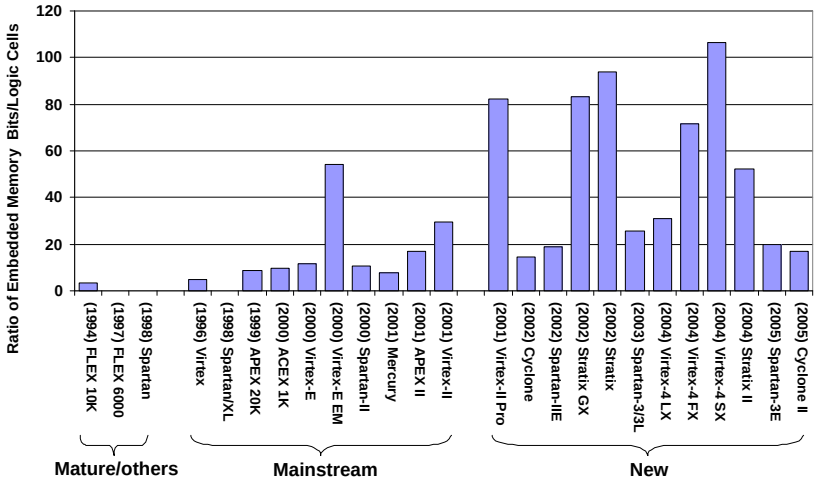


Figure 5.1: Ratio of embedded memory bits/logic cells on modern FPGAs. The number in the parentheses shows the release year of the device. New devices have 20 to 100 times more embedded memory bits than logic cells.

To justify the importance of this research area, we collected information on all the Xilinx and Altera FPGA devices [1, 2] over the past 10+ years and grouped them into three categories - mature, mainstream, and new. Figure 5.1 plots the ratio of embedded memory bits to logic cells of the largest FPGA[1] for each family of devices. It clearly illustrates the growing importance of embedded memory as newer devices have increasingly larger amounts of embedded memory. For example, there are over 100 times more embedded memory bits in Virtex-4 SX than logic cells. This points to a pressing need for optimizations that target the embedded memory of current and future generations of FPGA architectures.

[1]The largest means that the chip has the largest number of logic cells, or logic elements, with each logic cell containing a 4-input LUT and a D-type flip-flop.

As FPGA manufacturers move to advanced technology nodes[2], there are significant increases in leakage current due to the technology scaling of supplied voltage (V_{dd}), threshold voltage (V_{th}), channel length, and gate oxide thickness [52, 88]. These changes are making leakage power the dominant component of total power consumption, and new techniques are needed to address the leakage power concerns of FPGAs.

While dynamic power is dissipated only when transistors are switching, leakage power is consumed even if transistors are idle. Therefore, leakage power is proportional to the number of transistors [52]. An effective method in reducing leakage power is to put transistors into low power states. Since embedded memory blocks occupy an increasingly large area in FPGAs they are an ideal target for reducing the overall power.

A number of low-leakage circuit techniques [60, 88] have been proposed that save power by putting memory bits into a lower power state. Sleep transistors can be employed to shut off the power supply to the circuit and to put transistors into a *sleep* mode. While efficient in saving power, sleep mode does not retain data, and there is a large penalty to restore the data if it needs to be reaccessed [49]. Dual/multi-V_{dd} and dual/multi-V_{th} are other popular techniques that can be effectively used to limit the dynamic power consumption and to reduce leakage power consumption. In these *drowsy* [52] schemes, the data is preserved at a lower supply voltage and a small wakeup time is required to change the voltage from low to high, which is necessary to

[2] 90nm FPGAs are in production and 65nm is on the horizon.

access the data. Since drowsy mode does not fully turn off the transistors, it does not reduce the leakage power as much as sleep mode but does not lose data.

In memory leakage power optimization, the above-illustrated techniques have been employed mainly in caches of microprocessors [49, 52, 112]. Our research is specifically focused on studying leakage reduction control methods of FPGA embedded memories [3]. While the central idea behind all leakage power saving techniques is to exploit temporal information to control the supply voltage of regions of memory, embedded memories have many fundamental differences from caches. First, FPGAs memory accesses are usually *statically scheduled* and cannot easily handle the variable latencies associated with the predictive methods used by processor caches. Second, the data in embedded memories are usually *placed statically* as opposed to the dynamic reshuffling that caches try to do. Finally, embedded memories are not necessarily part of an memory hierarchy with inclusion, and thus more care must be taken not to lose important data.

In this chapter, we explore embedded-memory leakage power optimization in FPGAs and present an embedded memory leakage-aware design flow. We further propose a spectrum of leakage power management schemes for embedded memories. These schemes extract sleep and drowsy schedules for the memory from scheduled memory accesses and further reduce power through careful temporal control of, and data placement in, a given RAM. Through experimental evaluation of the schemes, we found

[3] While configuration SRAMs take a majority of memory resources on FPGAs, leakage from configuration SRAMs could be significantly reduced by using midoxide, high-V_t transistors, because configuration memory is usually off critical paths and doesn't switch during normal operation.

that by simply turning off the unused memory entries, 36.7% of the leakage power can be saved, while by carefully placing data in a leakage-aware manner, 94.7% of the memory leakage power can be eliminated.

The rest of the chapter is organized as follows. We formulate the leakage power problem of embedded memories in Section 5.2. In Section 5.3, we propose different schemes for reducing leakage power. We report our experimental results in Section 5.4. After reviewing related work in Section 5.5, we draw our conclusions in Section 5.6.

5.2 Problem Formulation

Considering that the embedded memory leakage problem is very important, and the current available design flow does not take into account the location of the variables within memory to optimize leakage power, our main contribution is to add two components, *path-traversal* and *location assignment* into the design flow (Figure 5.2) to achieve the minimal leakage power consumption of embedded memory. In our flow, the intermediate representation of an application is first scheduled and its memory accesses intervals are then recorded through the path-traversal component to build an interval graph. The interval graph, as exemplified in Figure 5.3, consists of the temporal relationship of live and dead time of all memory access intervals, with each vertex representing a live interval and each edge representing a dead interval. The location assignment component is added to figure out the best power saving mode on each interval as well as the best placement of the variables within the memory in order to achieve

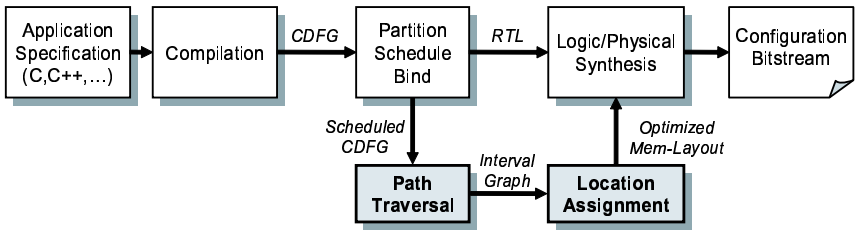


Figure 5.2: Design flow for leakage power reduction of embedded memory on FPGAs. Path traversal and location assignment are introduced components for deciding the best data layout within embedded memory to achieve the maximal power saving.

the minimal leakage power consumption. The whole design flow will be illustrated in detail with a real-world example later in this section.

If an embedded memory has been configured based on the requirement of the bit-width, the number of memory entries, denoted as N, is known. Through traversing the scheduled intermediate representation of an application, a set of memory access intervals I ($|I| = n$) with precedence orders can be derived. Then, the memory leakage power optimizing problem can be formulated as the following.

Problem: Given a memory with N finite number of memory entries, and a set of memory access intervals I with temporal precedence orders, find the best layout of the variables within the memory so that the maximal leakage power saving can be achieved.

In our study, the leakage power saving problem of variables assigned in the bounded size (N) embedded memory is modeled by an Extended Directed Acyclic Graph (Extended DAG) $G(V, E)$, where V is a set of finite v ($v \in \{v_s, v_1, \ldots, v_n, v_e\}$) vertexes and E is a set of finite e directed edges. A vertex v ($v \in V \backslash \{v_s, v_e\}$) in the DAG

indicates that the variable v is in the embedded memory, and the weight on the vertex v shows the leakage power saving during the live time of the variable, which is denoted by $w(v_i)$. Whenever there is a precedence order between two vertices's v_i and v_j, there is an edge, denoted by e_{ij}, representing the precedence order of the two vertices's. The edge is associated with a nonnegative weight $w(e_{ij})$ (the weight of an edge may be zeroed when the two incident vertexes are in the same memory location), which shows the leakage power saving during the time difference between assigning the two vertexes into the memory, or the dead time of the vertex v_i. The number of edges is denoted by e. The source vertex of an edge is called the parent vertex while the sink vertex is called the child vertex. A vertex with no parent is called a starting vertex v_s, and a vertex with no child is called an ending vertex v_e. There is an edge from the starting vertex v_s to every vertex in $V\backslash\{v_s, v_e\}$, and similarly, there is an edge from the vertex v_i in $V\backslash\{v_s, v_e\}$ to the ending vertex v_e. The unused memory spaces, where no variables are assigned into, are represented as edges from the starting vertex v_s to the ending vertex v_e. The length of a path i is the sum of all the weights on the vertexes and edges along the path, which shows the power saving in memory entry i.

Then, the memory leakage power problem is to assign n variables to N memory locations so that the maximal leakage power saving can be achieved by covering the n nodes $V\backslash\{v_s, v_e\}$ with N node-disjoint paths such that every node in $V\backslash\{v_s, v_e\}$ is included in exactly one path. Each path starts from the starting node and ends at the ending node.

According to the definition, the Extended DAG has the following properties:

1. **Property 1.** After path covering, the in-degree and the out-degree of the vertex v_i ($v_i \in V \backslash \{v_s, v_e\}$) are both equal to 1 to ensure that the paths have no duplicated vertexes and edges assigned to the same entry.

2. **Property 2.** The number of edges from the starting vertex v_s to the ending vertex v_e is equal to $N - k$, where k is the number of paths that cover all the n vertexes $\{v_1, \ldots, v_n\}$ and the corresponding edges.

The key to discovering the maximal leakage power saving is to choose the best operating mode on each interval, either active, drowsy, or sleep mode. This can be fulfilled by classifying an interval into one of the three categories: if an interval is very long then it would be beneficial to put that line in sleep mode for the duration of that interval; if an interval is very short, it should be simply put into the active mode and powered with high-V_{dd} mode; if an interval is somewhere in the middle, the drowsy mode would be the best. For live intervals, only the active or drowsy operating modes are allowed. It is because that the sleep mode does not preserves data and once the data is lost, the system will have to go to off-chip memory to refetch the data back, which is too costly to get any gain in terms of power.

To classify intervals into those three categories, two inflection points are introduced in our study: the active-drowsy inflection point and the drowsy-sleep inflection point. Inflection points are defined as the interval length where the operating mode changes.

The active-drowsy inflection point is the point between active and drowsy modes. It can be calculated as the sum of the durations within which the voltage changes either from V_{dd} to $V_{dd_{low}}$ or from $V_{dd_{low}}$ to V_{dd}. The drowsy-sleep inflection point is the point between drowsy and sleep modes. It is derived as the access interval length when the sleep and the drowsy modes consume the same amount of energy. If the interval is of a length less than the inflection point then drowsy mode would be optimal. If it is greater than the inflection point then sleep mode would be optimal. It has been proved that with perfect knowledge of the lengths of all intervals, the optimal leakage power saving can be achieved by applying the proper operating mode on each interval [75, 70].

Let's take radix-2 fft (fft-2) (Figure 5.3) as an example to explain in detail how the memory leakage-aware design flow works. The code segment of fft-2 is first compiled into CDFG intermediate representation, and the CDFG is scheduled. The scheduled CDFG is then traversed through the path traversal component to obtain the acyclic interval graph [4]. Based on the two inflection points, the best power saving mode is then selected on each interval, and its power saving is illustrated as weights of vertexes and edges on the graph. Finally, the location assignment component walks through the weighted interval graph and decides the maximal power saving through careful placement of variables into memory entries. A path $\{start, imag[0](n = 0), imag[1](n = 1), end\}$, for instance, means that the first access interval $(n = 0)$ of variable *imag[0]* and the second access interval $(n = 1)$ of variable *imag[1]* are

[4] Since the path traversal essentially determines the addresses of all the memory accesses and unrolls all the for loops, the interval graph is acyclic [68].

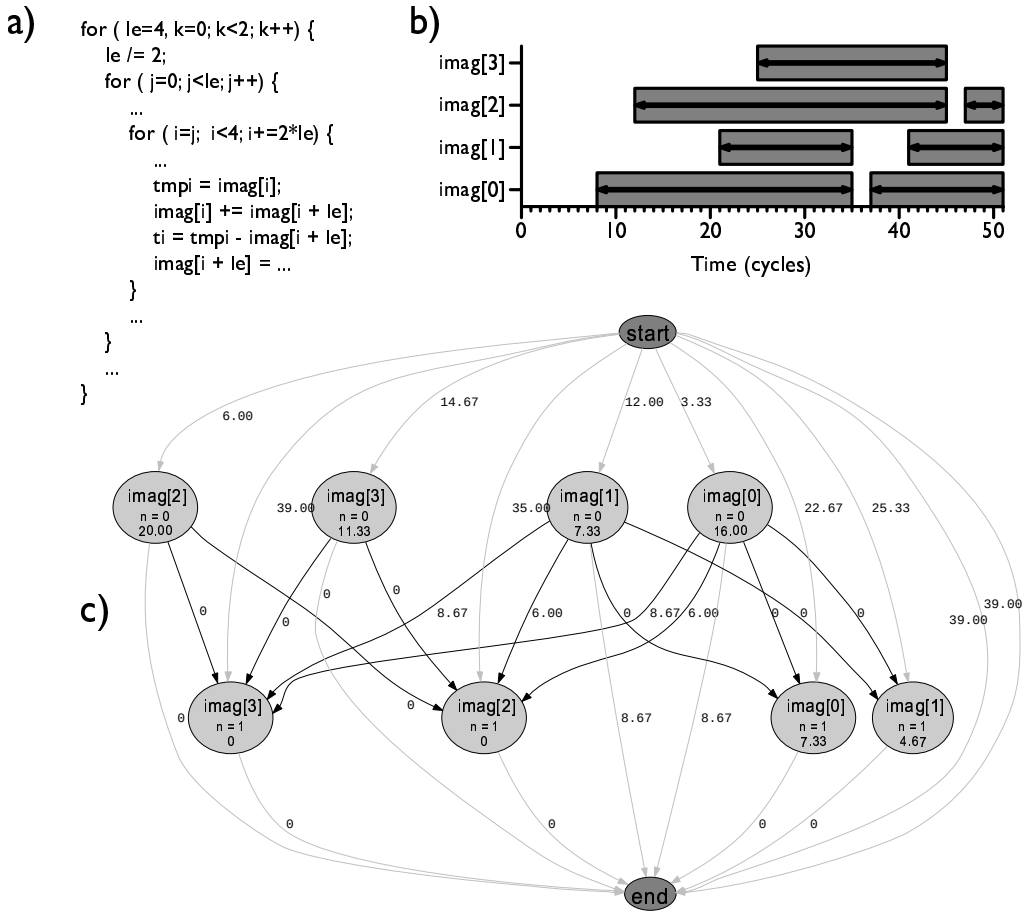


Figure 5.3: Problem formulation illustrated with the radix-2 fft example. The radix-2 fft example (a) is scheduled to extract memory access intervals (b), and the Extended DAG model is built by assigning all intervals to $N = 10$ entries(c). The live intervals are indicated by the gray rectangles in (b) and the gray vertexes in (c), and the dead intervals are depicted by the white space in (b) and edges in (c). A vertex includes the information of a variable name, its access number n and power saving. An edge shows the precedence order and the power saving between the adjacent vertexes. The length of a path i, defined as the sum of all the weights on the vertexes and edges along the path, indicates the leakage power saving of memory entry i.

assigned to the same memory entry, and the power saving in this entry is $Ps = 3.33 + 16.00 + 0 + 0 + 4.67 + 0 = 24.00$.

5.3 Leakage Power Reduction Schemes

With clear understanding of the leakage power problem and the Extended DAG modeling of the problem, let's see how different layouts of the data within memory affect the leakage power savings. We will explore different leakage reduction schemes step-by-step to understand how the maximal leakage power saving can be achieved through carefully assigning the variables into memory entries. We start with keeping every entry active as our baseline, and move forward to the schemes that have different data layouts, as are shown in Figure 5.4.

1. *Full-active.* It assigns one variable per memory entry. All memory entries are kept active, and there is no leakage power saving.

2. *Used-active.* Similar to full-active, it assigns one variable per memory entry and powers on the memory entries that are used. But it turns off the rest of the entries that are not used. The power saving is the percentage of entries that are unused.

3. *Min-entry.* It assigns all variables to the minimal number of memory entries. Those entries that have been used are powered on and the rest of the unused entries are turned off. The power saving is also the percentage of the entries that are unused.

4. *Sleep-dead.* Similar to min-entry, it uses the minimal number of entries. But it also has power savings on the intervals that are dead. So the total power saving consists of two parts: savings in unused entries and savings in dead intervals of the used entries.

5. *Drowsy-long.* Similar to Sleep-dead, it uses the minimal number of entries and saves power on the dead intervals. But it also saves power in live intervals using the drowsy technique. So the total power saving consists of three parts: savings in unused entries, savings in dead intervals, and savings in the live intervals of the used entries.

6. *Path-place.* Different from the above schemes that use the least number of entries, path-place picks the N path-covers that can lead to the maximal power saving.

Figure 5.4 illustrates the above-mentioned schemes. From the figure, we can see that when the precedence orders of all the live and dead intervals are taken into account, different data layouts result in different power savings.

The path-place algorithm (Table 5.1) is a greedy algorithm that can find the N paths to achieve the maximal leakage power saving. It works by first sorting all the vertexes in a topological order. Then a vertex v_i ($v_i \in V \backslash \{v_s, v_e\}$) is picked each time in the sorted list to calculate the maximal power saving from the starting vertex v_s up to v_i, or simply the length of the longest path reaching it. Note that the edges from the

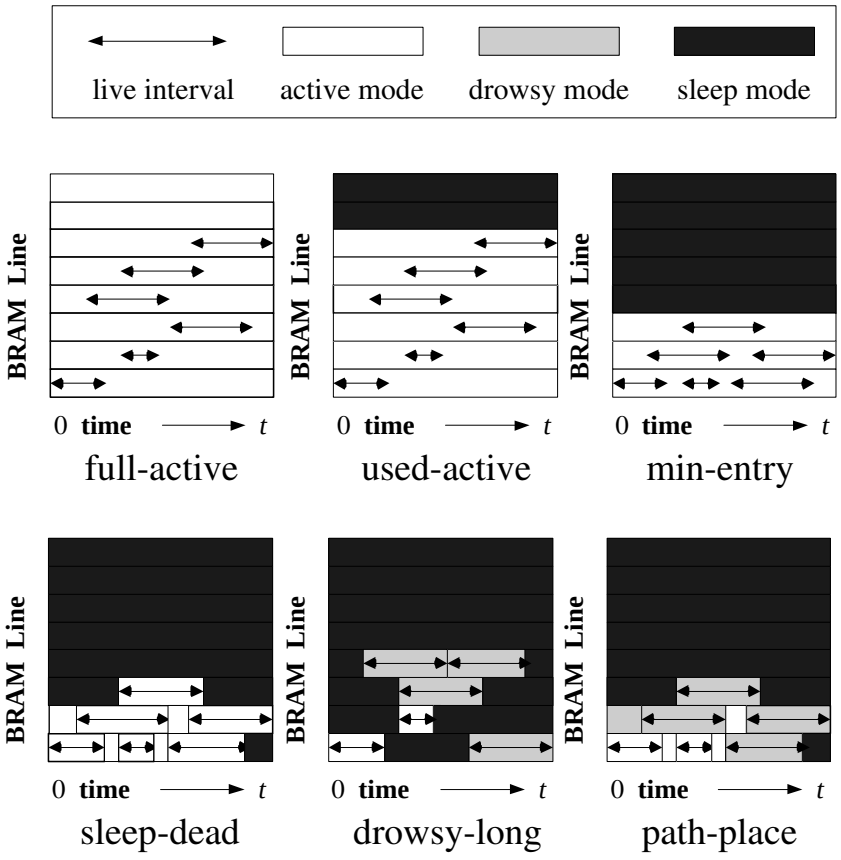


Figure 5.4: Different schemes to save leakage power of embedded memories on FPGAs. Full-active and used-active has one variable per entry. Min-entry, sleep-dead, and drowsy-long use the minimal number of entries, and apply power saving modes on unused entries, dead intervals, and live intervals incrementally. Path-place layouts variables with leakage awareness, and uses power savings on all the unused entries, dead and live intervals.

ALGORITHM **PATH_PLACE**

Input(G, N)
Output($totalSaving$, $path$)
//G: the Extended DAG; N: the number of entries
//$path$: the path for each vertex

Begin
1 Construct a list of all vertexes V in topological order, call it *Toplist*
2 **for** each vertex $v_i \in V \backslash \{v_s, v_e\}$ in *Toplist* **do**
3 $max = 0$
4 **for** each parent $v_p \in V$ of v_i **do**
5 **if** ($saving\_level(v_p) + w(v_i) + w(e_{pi}) > max$) **then**
6 $max = saving\_level(v_p) + w(v_i) + w(e_{pi})$
7 $id = path(v_p)$
8 **endif**
9 **endfor**
10 $path(v_i) = id$
11 $saving\_level(v_i) = max$
12 **endfor**
13 $totalSaving = 0$
15 **for** each parent $v_p \in V$ of v_e **do**
16 $totalSaving$ += $saving\_level(v_p) + w(e_{pe})$
17 **endfor**
End

Table 5.1: The path-place algorithm.

starting vertex v_s to the ending vertex v_e are the edges with the lowest priority to pick. In the end, the total power saving is computed as the sum of three components: the weights of all the final level vertexes that have no child except the ending vertex v_e, the weights of their edges that connect to v_e, and the weights of the $(N - k)$ edges from the starting vertex v_s to the ending vertex v_e if k is less than N. The path(v_i) function is used to calculate the path ID of the vertex v_i. Each time it sets the path ID of the vertex v_i as the path ID of its parent that can lead to the largest power saving of the vertex v_i. The complexity of the algorithm is $O((n + e) * N)$.

There are pros and cons with employing a leakage control technique at the entry level of embedded memory. The disadvantage is due to two types of the overheads: one caused by the employed low-leakage circuit techniques, and the other caused by the controller for choosing the best operating mode. However, as leakage problem gets exacerbated, adding control bits to the embedded memory becomes beneficial not only because of the decrease of cooling cost in packaging, but also due to the increase of circuit reliability [88]. The most important advantage is that adding the components of path-traversal and location assignment does not affect the current design flow for placement and routing in any way. It only gains additionally leakage power saving on embedded memory, which is a dominant portion on FPGAs.

5.4 Empirical Study

In section 5.3, we have discussed different schemes for reducing leakage power of embedded memory. In this section, we report our experimental results gathered from several DSP applications [24]: *dft, idft, radix-2 fft (fft-2), radix-4 fft (fft-4), filter*, and mp, a real design for efficient wireless channel estimation [74]. We first derive inflection points for different configurations of the memory block. We then show the comparison results of applying different schemes on different applications.

Due to the lack of the detailed information of commercial embedded memories, in our study we use the configuration schemes similar to dedicated blocks of on-chip memory, Block SelectRAM [2], of Xilinx Virtex family devices. That is to say, our targeted embedded memory is a true dual-read/write port synchronous RAM[5] with 18Kb memory bits. Each port can be independently configured as a read/write port, a read port, or a write port. Each port can also be configured to have different bit-widths: 1 bit, 2 bits, 4 bits, 9 bits (including 1 parity bit), 18 bits (including 2 parity bits), and 36 bits (including 4 parity bits). A read or a write operation requires only one clock edge. Both ports can read the same memory cell simultaneously, but can not write to the same memory cell at the same time. Therefore, there is no write conflict. In our experiments, the bit-width of each entry is set to be 18 bits, which is reasonable in those DSP applications. So the number of entries N is equal to 1K.

[5]Embedded memory blocks in contemporary FPGAs are typically implemented with synchronous SRAM [1, 110] to improve design performance.

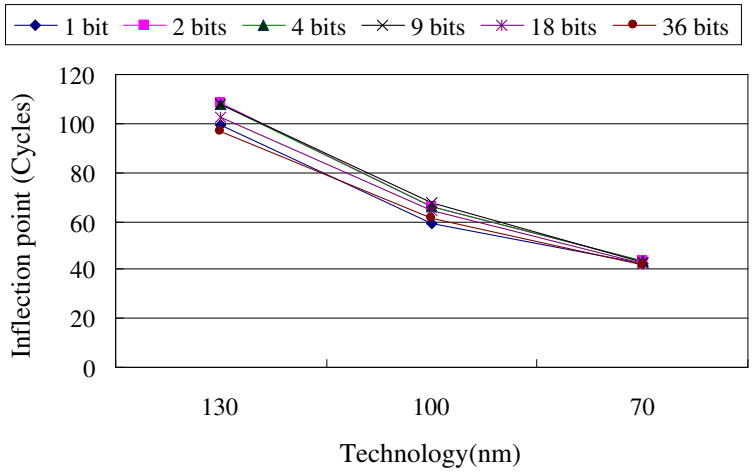


Figure 5.5: The drowsy-sleep inflection points are derived for different bit-width configurations of the embedded memory.

5.4.1 Deriving Inflection Points

The active-drowsy and drowsy-sleep inflection points are used to categorize all the live and dead access intervals. They are also used to select the best operating mode on each interval. In our study, we use the parameters in [75] to calculate inflection points, and assume that 3 cycles is needed to change the supply voltage from high to low and vice versa, and 30 cycles from high to off, and 3 cycles from off to high. So the active-drowsy inflection point can be calculated as 6 cycles. When calculating the drowsy-sleep inflection point, we simulated our target memory using modified eCACTI [72] to get both the dynamic power and leakage power consumptions, and derived the point where the drowsy and sleep modes consume the same amount of energy [75].

Figure 5.5 shows the inflection points for different configurations under different technologies. From the figure, we can see that under the same technology, the drowsy-

sleep inflection points for different configurations are the same; and when the technology scales down from 130nm to 70nm, the drowsy-sleep inflection point decreases from 102 to 43 cycles. Since 70nm is the most advanced technology available in the eCACTI tool, and features the next generation FPGA design technology, we used the 70nm technology and picked 43 cycles as the drowsy-sleep inflection point in the rest of our study.

Note that we also varied the drowsy-sleep inflection point from 43 to 640 cycles, and found the total leakage power savings are about the same. It is because that those intervals that contribute to most of the saving are very long, and small changes of the drowsy-sleep inflection point will not limit the power saving from those long intervals.

5.4.2 Comparing Different Schemes

We have proposed five different schemes to reduce memory leakage power: used-active, min-entry, sleep-dead, drowsy-long and path-place. We now study the power savings of the five schemes are on DSP applications. To assign the variables to the minimal number of entries (for min-entry, sleep-dead, and drowsy-long), the left-edge algorithm [58] was implemented in our experiments, which colors the graph in $O(nlog(n))$ time. To evaluate the different schemes we compared the five schemes against the full-active scheme, which is used as the baseline and has no power saving. To evaluate how close the path-place algorithm can bound to the optimal results from the experimental perspective, we also designed an OPT scheme, which assumes that

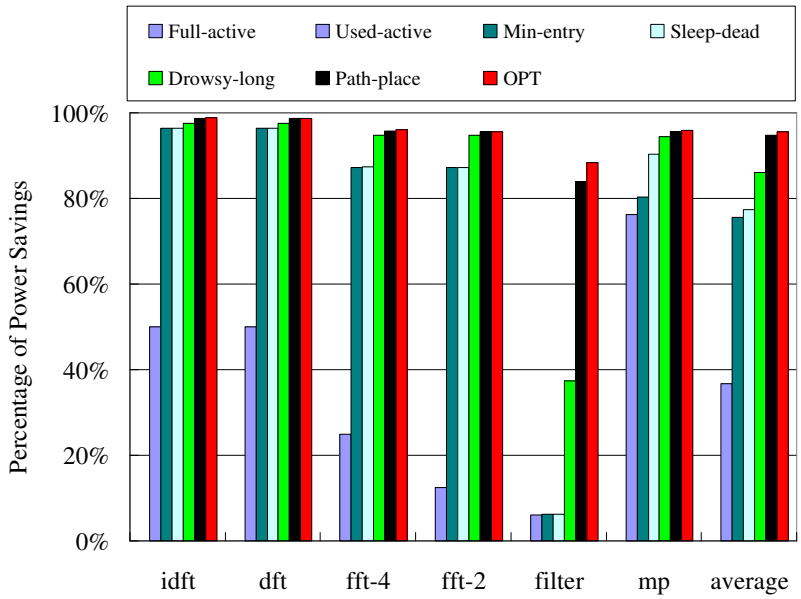


Figure 5.6: Comparison of the leakage power savings for different schemes.

no energy will be consumed during transitions between states. Figure 5.6 shows the comparison results for all the applications. From the figure, we can make the following observations:

1. By simply putting a single interval per entry and turning off the rest, as used-active does, 36.7% of leakage power can be saved on-average.

2. If the minimal number of entries are used, the leakage power savings is 75.6%, 77.3% and 86.0% for min-entry, sleep-dead, and drowsy-long, respectively, and the savings are increasing because more intervals are put into power saving modes. The reason that min-entry does well is that it packs the data very tightly (see Figure 5.4), and more entries that could be completely turned off to save power are left unused. Moreover, due to the effect of compact packing, the dead

intervals that are in the used entries are very short, which leaves very little space for sleep-dead to save power. Consequently, sleep dead performs at a similar level in efficiency as min-entry.

3. Among all, path-place achieves the best leakage power saving, 94.7%, which is about 8.7% better than the drowsy-long scheme. It is because path-place layouts the data in a way that the sleep mode can be explored to the largest extent on all the intervals, and among all three operating modes: active, drowsy, and sleep, the sleep mode can provide the maximal power saving. Also, on average, path-place can approach the bound of the OPT scheme very closely (less than 1.2%), which assumes that there is no transition energy consumption.

4. In terms of best schemes, both min-entry and path-place are favorable. Min-entry is very simple but effective. It only needs to use sleep techniques to turn off the unused entries after interval packing and can achieve a good amount of power saving. By contrast, path-place is very effective but a bit more costly in terms of running time to discover the best layout. If drowsy techniques are incorporated along with sleep techniques, path-place could accomplish the best leakage power saving.

5. For $filter$, the simple used-active scheme does not save too much power. It is because that different from other applications, the number of its variables is close to the total number of entries, and only few entries that are unused can

be put into sleep to save power. But when the variables are layout carefully, significant power saving can be achieved, as path-place does.

6. For the case study of the MP core, 76% of power saving can be achieved by simple turning off the unused part by using the used-active scheme. This is due to the fact that only one third of the entries of an embedded memory is used. Through leakage-aware placement of the data, the path-place algorithm can achieve 94% of leakage power saving.

These provide us the answer that the layout of the data within the memory entries has a significant impact on the leakage power optimization. Moreover, with the available circuit techniques, careful placement of the intervals within the memory can reduce the leakage power by a large magnitude.

5.5 Related Work

We have reviewed the related work on low-leakage circuit techniques in Section 5.1, which have been studied extensively for optimizing leakage power of ASICs and microprocessors [11, 95, 34, 47, 49, 51, 52, 59, 88]. Now we see what are the different techniques that have been proposed to reduce leakage power for FPGAs, which has been in focus only recently [8, 4, 64, 97, 105]. Leakage tolerant FPGA architectures were analyzed and discussed in [14, 15, 4], in which the whole fabric is divided into regions called buckets. Each bucket contains a pre-determined number of CLBs and

the bucket is powered on/off by a sleep transistor. The sleep transistor is controlled by a power state controller to intelligently shut down buckets whenever they are idle. [36] addressed the problem of automatically packing a given design onto the logic block based on their temporal profile. Shang et al. [97] analyzed dynamic power consumption in Virtex-II FPGA family based on measurement and simulation. Tuan and Lai studied the leakage power of Xilinx architecture. And Li et al [63] proposed fpgaEVA-LP for power efficiency analysis of LUT based FPGA architectures. Several techniques for reducing leakage power on FPGAs have been proposed. Gayasen et al. [4] studied region constraint placement to disable unused portions by employing sleep transistors. Anderson et al. [8] considered selecting polarities for signals at the inputs of LUTs so they spend the majority of time in low leakage states. Li et al. [64] proposed to use pre-defined dual-V_{dd} and dual-V_t fabrics to reduce FPGA power. Rahman et al. [83] evaluated the trade-offs of different low-leakage design techniques for FPGAs. While there has been work in low power FPGAs and other work in architectural-level policies for controlling memory leakage, we believe this to be the first work to address embedded memory leakage power in FPGAs.

5.6 Summary

In this study we argue that embedded memory leakage power will be a large and growing concern for FPGAs and that design flows can be effective in reducing this power. We further present a leakage-aware design flow and proposed five schemes

for reducing leakage power of embedded memory on FPGAs. The new flow takes into account the leakage-aware location assignment of variables within memory. The proposed five schemes employ the sleep and drowsy techniques, and exploit the live and dead interval information of memory accesses to save power. They function by choosing the best operating mode, active, drowsy or sleep, on each interval. Through the experimental evaluation, we found that the simple scheme like used-active can provide a good amount of benefits, and by carefully placing data into memory entries, a significant amount of leakage power saving can be further achieved.

Chapter 6

Conclusions

6.1 Research Summary

Personal wireless communications applications are playing a significant role in communications systems in recent years, and advanced communications algorithms have been developed to mitigate multi-path and multi-user interference, as well as to achieve increased capacity and spectral efficiency which is becoming more and more important for future wireless systems. At the same time, digital integrated circuit technology have been growing continuously following the Moore's law, and many architecture and circuit design techniques have been developed for portable devices which have stringent energy consumption, performance and cost constraints. However, there is still a big gap between these two fields, neither algorithm nor architecture characteristics are being fully exploited. In order to reach a more optimized system design in terms of power, performance, and area, etc., algorithms and architectures need to be considered together to achieve the best matching between them.

This dissertation is on the co-exploration of algorithms and architectures for the custom implementation of digital signal processing in wireless channel estimation. The interaction between the two are exploited to derive efficient solution and to bridge the gap between system and circuit level design. A cross-cutting method is proposed to assist the algorithm/architecture co-exploration of dedicated hardware design, resulting in novel optimizations at every level of the system design, from the theory and algorithms to the arithmetic and placement.

The co-exploration enables us to study the performance of various algorithms at the system level and at the same time in architecture that matches the computational requirements of the algorithm to optimize the power consumption and implementation cost. The advantages of bring together system and algorithm design, architecture design, and low power design techniques are demonstrated by the design example of matching pursuit (MP) core, an efficient wireless channel estimator.

MP core demonstrates that multiple orders of magnitude gain in both energy and performance efficiency can be obtained by using dedicated hardwired and highly parallel architectures: achieving 216x speedup compared to running the MP algorithm on a 2.17GHz high performance AMD microprocessor, and 25% energy saving through clock gating the nonactive parts.

6.2 Future Work

There are a number of avenues that could extend this research.

The FPGAs has progressed into a new model of computation. First, re-programmability of the architecture has been recognized as a major evolution, which made it possible to delay the programming of the FPGA to the last minute, and even change it if an error were detected in the implementation. Second, FPGAs are widely adopted for prototyping. The fact that the architecture can be programmed over and over again to try out actual hardware implementations of the design ideas was a desired capability. However, designers have to face the problem of high power consumption that holds up FPGAs from penetrating the vast majority of applications. Low standby power is thus crucial, particularly for mobile applications. One direction to extend for reducing leakage power on embedded memories on FPGAs will be incorporating physical layout information in the design flow to better estimate the power consumption, since adding control signals for reducing power may lead to routing congestion and layout change.

Second, it would be interesting to extend the path-place algorithm based on both temporal and spatial information to reduce power consumption in registers on ASICS. Since registers are large in size on ASICS, they consume a significant amount of leakage power. With static scheduling information, registers could be put into low power mode to save power. So does functional units. Functional units consume a large portion of power. Based on the temporal information, we could decide when to put them into low power mode to save power and when to wake them up for fast execution. Using the spatial information, we could further decide that out of a number of functional

units of the same type which functional unit the operations can be binded to to save power.

Another interesting direction to pursue is to estimate the interval behavior based on statistical data analysis. In this study, we have shown that based on the oracle knowledge of an address trace, we can achieve maximal power saving on a given cache architecture with the generalized model. Instead of using the trace information of a specific application, we could employ the sub-set sampling technique to explore the statistical information of a set of applications to estimate the interval behavior and provide a more general model, which does not restrict to a particular application.

Bibliography

[1] Altera press releases and device data sheets. http://www.altera.com.

[2] Xilinx press releases and device data sheets. http://www.xilinx.com.

[3] A. Agarwal, H. Li, and K. Roy. DRG-cache: a data retention gated-ground cache for low power. In *Proceedings of Design Automation Conference*, June 2002.

[4] A.Gayasen, Y.Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and T. Tuan. Reducing leakage energy in fpgas using region-constrained placement. In *ACM International Symposium on Field-Programmable Gate Arrays*, 2004.

[5] ZigBee Alliance. ZigBee Specification Version 1.0. 2005.

[6] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk. Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[7] B. S. Amrutur and M. A. Horowitz. Fast low-power decoders for RAMs. *IEEE Journal of Solid-State Circuits*, 36(10):1506–1515, 2001.

[8] J.H. Anderson, F.N. Najm, and T. Tuan. Active leakage power optimization for fpgas. In *ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2004.

[9] N. Azizi, F. N. Najm, and A. Moshovos. Low-leakage asymmetric-cell SRAM. *IEEE Transaction on Very Large Scale Integration Systems*, 11(4), Aug. 2003.

[10] R. Bai, N. S. Kim, T. Mudge, and D. Sylvester. Power-performance trade-offs in nanometer-scale multi-level caches considering total leakage. In *Proceedings of Design, Automation and Test in Europe*, Mar. 2005.

[11] S. Bakshi and D. Gajski. A memory selection algorithm for high-performance pipelines. In *Proceedings of the European Design Automation Conference*, Sept. 1995.

[12] L. Belady. A study of replacement of algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[13] P. Belanovic and M. Leeser. A library of parameterized floating-point modules and their use. In *International Conference on Field Programmable Logic and Applications (FPL 2002)*, 2002.

[14] R. Bharadwaj, R. Konar, P.T. Balsara, and D. Bhatia. Exploiting temporal idleness to reduce leakage power in programmable architectures. In *Proceedings of IEEE Asia-South Pacific Design Automation Conference*, Jan. 2005.

[15] R. Bharadwaj, R. Konar, D. Bhatia, , and P. Balsara. Fpga architecture for standby power management. In *IEEE International Conference on Field Programmable Technology*, Dec. 2005.

[16] A. D. Blackowiak and S. D. Rajan. Multi-path estimates via optimizing highly oscillatory cost functions. In *IEEE Journal on Oceanic Engineering*, volume 30, July 1998.

[17] J. Cavallaro. Architectures for Heterogeneous Multi-Tier Networks. *Kluwer Journal on Wireless Personal Communications*, 2002.

[18] S. Choi, R. Scrofano, V. K. Prasanna, and J. Jang. Energy-efficient signal processing using fpgas. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2003.

[19] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, June 2002.

[20] S. F. Cotter and B. D. Rao. Sparse channel estimation via matching pursuit with application to equalization. *IEEE Transaction on Communications*, 50:374–377, Mar. 2002.

[21] M. Cummings and S. Heath. Mode switching and software download for software defined radio: the SDR Forum approach. *IEEE Communications Magazine*, Aug. 1999.

[22] A. DeHon and J. Wawrzynek. Reconfigurable computing: What, why, and implications for design automation. In *Proceedings of the 36th Design Automation Conference*, 1999.

[23] R. Desikan, D. Burger, S. W. Keckler, and T. M. Austin. Sim-alpha: a validated execution driven alpha 21264 simulator. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.

[24] P. M. Embree and B. Kimble. *C Language Algorithms for Digital Signal Processing*. Prentice Hall, Englewood Cliffs, 1991.

[25] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the World with Wireless Sensor Networks. In *International Conference on Acoustics, Speech, and Signal Processing*, Salt Lake City, Utah, May 2001.

[26] L. Jian et al. A dynamically-reconfigurable power-efficient turbo decoder. *FCCM*, 2004.

[27] S. Mahlke et al. Bit-width cognizant architecture synthesis of custom hardware accelerators. In *IEEE Transaction on Computer-Aided Design*, Nov. 2001.

[28] V. Agarwal et al. Clock rate versus ipc: the end of the road for conventional microarchitectures. *proceedings of International Symposium on Computer Architecture*, 2000.

[29] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. Memory-system design considerations for dynamically-scheduled processors. In *Proceedings of Intl. Symposium on Computer Architecture*, June 1997.

[30] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of Intl. Symposium on Computer Architecture*, Anchorage, AK, May 2002.

[31] J. J. Fuchs. Multipath time-delay detection and estimation. *IEEE Transaction on Signal Processing*, 47:237–243, 1999.

[32] A. A. Gaffarl, O. Mencerl, W. Luk, and P. Y. K. Cheung. Unifying bit-width optimization for fixed-point and floating-point designs. In *IEEE Transaction on Computer-Aided Design*, Nov. 2001.

[33] J. Grodstein, M. Gowan, R. Lane, B. Lilly, K. Nagalla, R. Shah, E. Shriver, S. Yin, S. Morton, R. Rayess, T. Truex, L. Shattuck, S. Lowell, D. Bailey, D. Bertucci, G. Bischoff, and D. Dever. Power and cad considerations for the 1.75mbyte, 1.2ghz l2 cache on the alpha 21364 cpu. In *Proceedings of the 12th ACM Great Lakes symposium on VLSI*, New York, NY, Apr. 2002.

[34] P. Gupta, A. B. Kahng, P. Sharma, and D. Sylvester. Selective gate-length biasing for cost-effective runtime leakage control. In *Design Automation Conference*, 2004.

[35] H. Hanson, Agarwal, M.S. Hrishikesh, S.W. Keckler, and D.C. Burger. Static energy reduction techniques for microprocessor caches. In *Proceedings of International Conference on Computer Design*, Sep. 2001.

[36] H. Hassan, M. Anis, A. Daher, and M. Elmasry. Activity packing in fpgas for leakage reduction. In *IEEE Design Automation and Test*, Mar. 2005.

[37] J. R. Hauser and J. Wawrzynek. Garp: a mips processor with a reconfigurable coprocessor. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 12–21, 1997.

[38] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *Proceedings of International Symposium on Computer Architecture*, Anchorage, Alaska, May 2002.

[39] H.Kim and K. Roy. Dynamic vt SRAM's for low leakage. In *Proceedings of ACM International Symposium on Low Power Design*, Aug. 2002.

[40] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proc. of the IEEE*, 89(4):490–504, 2001.

[41] J. S. Hu, A. Nadgir, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Exploiting program hotspots and code sequentiality for instruction cache leakage management. In *Proceedings of International Symposium on Low Power Electronics and Design*, Seoul, Korea, Aug. 2003.

[42] Z. Hu, S. Kaxiras, and M. Martonosi. Let caches decay: Reducing leakage energy via exploitation of cache generational behavior. *ACM Transaction on Computer Systems*, May 2002.

[43] C. Huang, S. Ravi, A. Raghunathan, and N. K. Jha. High-level synthesis of distributed logic-memory architectures. In *Proceeding of International Conference on Computer-Aided Design*, Nov. 2002.

[44] R. A. Iltis and S. Kim. Geometric derivation of expectation-maximization and generalized successive interference cancellation algorithms with CDMA channel estimation. *IEEE Transaction on Signal Processing*, 51(5):1367–1377, May 2003.

[45] International technology roadmap for semiconductors. http://public.itrs.net.

[46] M. F. Jacome and H. P. Peixoto. A survey of digital design reuse. In *IEEE Design and Test of Computers*, May 2001.

[47] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu. Banked scratch-pad memory management for reducing leakage energy consumption. In *International Conference on Computer Aided Design*, San Jose, CA, 2004.

[48] R. Kastner, A. Kaplan, and M. Sarrafzadeh. *Synthesis Techniques and Optimizations for Reconfigurable Systems*. Kluwer Academic, Boston, 2004.

[49] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *International Symposium on Computer Architecture*, Gőteborg, Sweden, June 2001.

[50] R. Kessler. The alpha 21264 microprocessor'. In *IEEE Micro*, pages 24–36, Mar. 1999.

[51] K. S. Khouri and N. K. Jha. Leakage power analysis and reduction during behavioral synthesis. *IEEE Transaction on Very Large Scale Integration Systems*, 10(6), Dec. 2002.

[52] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and micro-architectural techniques for reducing cache leakage power. *IEEE Transaction on Very Large Scale Integration Systems*, 12(2):167–184, Feb. 2004.

[53] N. Kim and T. Mudge. Single vdd and single vt super-drowsy techniques for low-leakage high-performance instruction caches. In *Proc. of the International Symposium on Low Power Electronics and Design*, Newport Beach, CA, Aug. 2004.

[54] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemi, and N. Vijaykrishnan. Leakage current: Moore's law meets static power. *Computer*, 36(12), Dec. 2003.

[55] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction cache: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of ACM/IEEE International Symposium on Microarchitecture*, Istanbul, Turkey, 2002.

[56] S. Kim. *DS-CDMA Multiuser Channel Estimation Algorithms with Applications to Radiolocation*. PhD thesis, Univ. of California, Santa Barbara, March 2005.

[57] S. Kim and R. A. Iltis. A matching pursuit/GSIC-based algorithm for DS-CDMA sparse channel estimation. *IEEE Signal Processing Letters*, 11:12–15, Jan. 2004.

[58] F. J. Kurdahi and A. C. Parker. Real: A program for register allocation. In *Design Automation Conference*, 1987.

[59] E. Kusse and J. Rabaey. Low-energy embedded fpga structures. In *International Symposium on Low Power Electronics and Design*, 1998.

[60] D. Lee, D. Blaauw, and D. Sylvester. Gate oxide leakage current analysis and reduction for VLSI circuits. *IEEE Transaction on Very Large Scale Integration Systems*, 12(2), Feb. 2004.

[61] D. U. Lee, W. Luk, C. Wang, and C. Jones. A flexible hardware encoder for low-density parity-check codes. *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004.

[62] Gary Legg. ZigBee: Wireless Technology for Low-power Sensor Networks. In *TechOnLine Publication*, May 2004.

[63] F. Li and L. He. Power modeling and characteristics of field programmable gate arrays. *IEEE Transaction on Computer-aided design*, 24(11):1712–1724, Nov. 2005.

[64] F. Li, Y. Lin, L. He, and J. Cong. Low-power fpga using pre-defined dual-vdd/dual-vt fabrics. In *ACM International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 2004.

[65] L. Li, I. Kadayif, Y. Tsai, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and A. Sivasubramaniam. Managing leakage energy in cache hierarchies. *Journal of Instruction-level Parallelism*, 5, Feb. 2003.

[66] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, K. Skadron, and M. Stan. State-preserving vs. non-state-preserving leakage control in caches. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference*, Feb. 2004.

[67] J. Liang, R. Tessier, and D. Goeckel. A dynamically-reconfigurable, power-efficient turbo decoder. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2004.

[68] Y. D. Liang and G. K. Manacher. An $O(nlogn)$ algorithm for finding a minimal path cover in circular-arc graph. In *ACM Conference on Computer Science*, pages 390–397, 1993.

[69] D. Liu and C. Svensson. Trading speed for low power by choice of supply and threshold voltages. *IEEE Journal of Solid State Circuits*, 28(1), Jan. 1993.

[70] J. Liu and P.H. Chou. Optimizing mode transition sequences in idle intervals for component-level and system-level energy minimization. In *Proceedings of International Conference on Computer Aided Design*, San Jose, CA, Nov. 2004.

[71] Intel madison processor. http://www.intel.com.

[72] M. Mamidipaka and N. Dutt. ecacti: An enhanced power estimation model for on-chip caches. Technical Report Tech. Report TR-04-28, UC. Irvine, Sept. 2004.

[73] S. Meguerdichian, F. Koushanfer, A. Mogre, D. Petranovic, and M. Potkonjak. Metacores: design and optimization techniques. In *Proceeding of the Design automation conference*, Las Vegas, Nevada, June 2001.

[74] Y. Meng, A. Brown, R. Iltis, T. Sherwood, H. Lee, and R. Kastner. Mp core: Algorithm and design techniques for efficient channel estimation in wireless applications. In *Design Automation Conference*, 2005.

[75] Y. Meng, T. Sherwood, and R. Kastner. On the limits of leakage power reduction in caches. In *International Symposium on High-Performance Computer Architecture*, 2005.

[76] G. D Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[77] E. Mirsky and A. DeHon. Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 157–66, 1996.

[78] Todd C. Mowry, Monica S. Lam, and Anoop Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62–73, Oct. 1992.

[79] J. Oliver, R. Rao, P. Sultana, J. Crandall, E. Czernikowski, L. W. Jones, D. Franklin, V. Akella, and F. T. Chong. Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. *International Symposium on Computer Architecture*, 2004.

[80] M.D. Powell, S. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Reducing leakage in a high-performance deep-submicron instruction cache. *IEEE Transaction on Very Large Scale Integration Systems*, 9(1), Feb. 2001.

[81] J. Proakis. *Digital Communications*. McGraw-Hill, New York, NY, 1995.

[82] J.M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits A Design Perspective(2nd Edition)*. Prentice-Hall, 2002.

[83] A. Rahman and V. Polavarapuv. Evaluation of low-leakage design techniques for field programmable gate arrays. In *ACM International Symposium on Field-Programmable Gate Arrays*, 2004.

[84] S. Rajagopal, S. Bhashyam, and J. R. Cavallaro. Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers. In *IEEE Transaction on Wireless Communications*, volume 1, pages 374–377, July 2002.

[85] S. Rajagopal and J. Cavallaro. A bit-streaming pipelined multiuser detector for wireless communications. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, Sydney, Australia, May 2001.

[86] S. Rajagopal, B. Jones, and J. R. Cavallaro. Task partitioning wireless base-station receiver algorithms on multiple dsps and fpgas. In *International Conference on Signal Processing Applications and Technology*, 2000.

[87] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. *International Symposium on Microarchitecture*, pages 172–80, 1994.

[88] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, 91(2), Feb. 2003.

[89] C. R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. M. Arnold, and M. Gokhale. The napa adaptive processing architecture. *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 28–37, 1998.

[90] V.K. Prasanna S. Choi, R. Scrofano and J. Jang. Energy-efficient signal processing using fpgas. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2003.

[91] S. Sair, T. Sherwood, and B. Calder. Quantifying load stream behavior. In *Proceedings of International Symposium on High-Performance Computer Architecture*, Feb. 2002.

[92] K. Sakiyama, P. Schaumont, and I. Verbauwhede. Finding the best system design flow for a high-speed JPEG encoder. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, KitakYushu, Japan, Jan. 2003.

[93] Samsung Electronics. DDR2SDRAM datasheet MR16R1622, July 2002.

[94] K. Sankaranarayanan and K. Skadron. Profile-based adaptation for cache decay. *ACM Transaction on Architecture and Code Optimization*, 1(3), Sep. 2004.

[95] H. Schmit and D. Thomas. Address generation for memories containing multiple arrays. *IEEE Transaction on VLSI Systems*, 17:377–388, May 1998.

[96] R. Scrofano, S. Choi, and V. K. Prasanna. Energy efficiency of fpgas and programmable processors for matrix multiplication. In *International Conference on Field-Programmable Technology (FPT)*, 2004.

[97] L. Shang, A. S. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. In *ACM International Symposium on Field-Programmable Gate Arrays*, 2002.

[98] T. Sherwood, E. Perelman, G. Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 2002.

[99] P. Shivakumar and N.P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical Report WRL-2001-2, HP Labs Technical Reports, Dec. 2001.

[100] M. Stephenson, J.Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *IEEE Transaction on Computer-Aided Design*, Nov. 2001.

[101] G. Stitt and F. Vahid. Energy advantage of microprocessor platforms with on-chip configurable logic. *IEEE Design and Test of Computers*, 2002.

[102] R. A. Sutton, V. P. Srini, and J. M. Rabaey. A multiprocessor dsp system using paddi-2. In *Design Automation Conference*, 1998.

[103] S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson. A dynamically reconfigurable adaptive viterbi decoder. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2002.

[104] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J-W Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A Agarwal. The raw microprocessor: a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

[105] T. Tuan and B. Lai. Leakage power analysis of a 90nm FPGA. In *IEEE Custom Integrated Circuits Conference*, 2003.

[106] L. Vanzago, B. Bhattacharya, J. Cambonie, and L. Lavagno. Design space exploration for a wireless protocol on a reconfigurable platform. In *Design, Automation and Test in Europe*, Munich, Germany, Mar. 2003.

[107] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proceedings of 2002 Workshop on Memory Performance Issues in conjunction with ISCA-29*, Anchorage, Alaska, May 2002.

[108] S. Verdú. *Multiuser detection*. Cambridge University Press, New York, Oct. 1998.

[109] M. Wouters, G. Vanwijnsberghe, P. V. Wesemael, T. Huybrechts, and S. Thoen. Real time implementation on fpga of an ofdm based wireless lan modem extended with adaptive loading. In *European Solid-State Circuits Conference(ESSCIRC)*, 2002.

[110] Xilinx, INC. *Virtex-II 1.5V Field Programmable Gated Arrays Datasheet*, Oct. 2001.

[111] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. Chimaera: a high-performance architecture with a tightly-coupled reconfigurable functional unit. *International Symposium on Computer Architecture*, pages 225–35, 2000.

[112] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Complier-directed instruction cache leakage optimization. In *International Symposium on Microarchitecture*, Istanbul, Turkey, Nov. 2002.

[113] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. R. Stan. Hotleakage: An architectural, temperature-aware model of subthreshold and gate leakage. Technical Report Tech. Report CS-2003-05, Department of Computer Sciences, University of Virginia, Mar. 2003.

[114] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: a static power-efficient cache design. *ACM Transactions on Embedded Computing Systems*, 2(3), Aug. 2003.