Maximizing Energy Efficiency through Vertically Integrated Dynamic Reconfigurability

---

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Saad Arrabi

May

2014

APPROVAL SHEET

The dissertation

is submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Saad Arrabi

AUTHOR

The dissertation has been read and approved by the examining committee:

John Lach

Advisor

Benton Calhoun

Mircea Stan

Kevin Skadron

Gabriel Robins

Accepted for the School of Engineering and Applied Science:

James H. Aylor

Dean, School of Engineering and Applied Science

May

2014

Abstract

Power and energy consumption have become some of the main hurdles for performance advancement in modern digital systems. High transistor densities and clock speeds have created significant thermal and power delivery issues, and increasingly pervasive portable electronic devices have limited energy sources but high performance expectations. Power affects and is affected by almost every aspect of digital system design, enticing designers and researchers to approach this problem from every possible aspect.

This work explores two such aspects: dynamic reconfigurability and vertical integration. Dynamic reconfigurability is the ability to change the configuration of the system dynamically during runtime, and vertical integration is using low-level design information in high-level decisions (and vice-versa) as well as co-optimizing multiple design levels simultaneously. Both of these concepts can be targeted to various system metrics; in this work, they are used to reduce the total energy consumption of the system with the goal of increasing system battery lifetime while still providing the necessary performance and functional capabilities.

This work explores dynamic reconfigurability and vertical integration through the use of three practical systems. We go through the steps and analysis in designing these systems that will show the potential benefits of our methods. Through the steps of designing the systems, we show the impact of several significant aspects of vertically integrated dynamically configurable systems, thus providing a framework for identifying potential benefits of reconfigurability and co-optimization and for general guidelines of how to implement them on other systems. We also look at the granularity of configurability and the overhead that it incurs, as well as the circuit-level details and information that might affect and be affected by architectural or system-level decisions.

The first system is a wireless body sensor node that consumes most of its energy wirelessly transmitting data. We investigate how adaptive software can compress the data before sending without sacrificing information using only the limited processing and memory resources that typically reside on body

sensors. We go through the various aspects that affect our solution and the different design levels we have to take into account when applying the compression.

The second example system is a custom digital signal-processing system utilizing Panoptic Dynamic Voltage Scaling (PDVS), in which we investigate adding fine-grained spatial and temporal granularities of voltage scaling to the processor. We use a vertically integrated approach to explore the trade-offs of the ability to switch the voltage of a single arithmetic component (spatial) for a single operation (temporal), as well as the different variables that need to be considered for scheduling such systems.

The third system is a Field Programmable Core Array (FPCA), in which we investigate various methods for creating a configurable processor capable of switching between SISD, SIMD, and MIMD configurations. In this system, we also characterize the trade-offs of different levels of configurability and analyze the effect of the circuit overheads on the architectural design of the system.

By studying the various design aspects of these three systems and their effects on system-level tradeoffs through simulations and physical measurements, we present some of the main variables as well as simple guidelines to consider when adding dynamic configurability and vertical integration to system designs.

Acknowledgments

Before I convey my thanks to the many, many people I have to thank, I would like to say how it is impossible for me to account for everyone who helped me on the way to reach this point, and I hope this fact will be taken into consideration when reading this section.

The number of people who have helped me produce this work is just uncountable. First and foremost, I would like to acknowledge the tremendous help I got from my family throughout my whole education path. They were there whenever I needed them, and I honestly could not think of anything I needed that they did not provide.

Second, I would like to mention the great gratitude I have for my adviser, Dr. John Lach, to whom I owe the super majority of thanks for my whole Ph. D. work from start to finish. I am really thankful to him for choosing me to join his wonderful team and do research and work under his supervision and guidance. Thank you for giving me this opportunity, and I truly think that without it, I would not have completed my Ph.D. degree. Thank you.

I would also like to thank my Ph.D. committee—Dr. Benton Calhoun, Dr. Kevin Skadron, Dr. Mircea Stan, and Dr. Gabriel Robins—for their great help and continuous guidance in writing and creating this work.

Through my work at the University of Virginia, I have encountered a multitude of wonderful people who have helped me in many aspects of my work and my life. First, I would like to mention the students whom I have worked with the most and built real friendship with, Yousef Shakhsheer, Kyle Craig, and Sudhanshu Khanna. I have worked very closely with these students for several years on several projects, and we had our share of happy times, awkward times, and long nights to meet deadlines in our tiny, windowless, but fun office.

I am also grateful for the friendships I have formed in the office during the years, which helped me to stay sane and to enjoy my work days although they included working long nights. Mark Hanson, Adam Barth,

Harry Powell, and Mateja Putic were the first students whom I had the pleasure to work with. They were closer to mentors, for they have been here a while before me. Alicia Klinefelter, Yanqing Zhang, Peter Beshay, Liang Wang, Runjie Zhang, Wei Zhang, and many other great friends who, although they were not here when I first started, had great positive influence on me and my well-being through my Ph.D.

Although I did not work with the following people in the office, they still influenced my time here significantly: Tamim Sookoor, Zain Syed, Ali Khan, Haroon Ahmad, Thomas Quattlebaum, Alla Hassan, Muhaimeinul Khan, Mahmood Gharavi, Hadi Anwar, Muhibullah Tora, Mouadh Benammar, and all the great members of the Muslim Students Association who have welcomed me immediately when I first came here and all the other great people I have met through them during my six years here. Those guys and the club in general were my secret way of fully rejuvenating after long stressful hours of deadlines. While I have only known those people for a few years, I believe that I have found lifelong friends in them.

I wish that I were able to mention every single person I know for certain who helped me be where I am. Sadly, this is not possible, for the rest of the dissertation would be dwarfed in comparison. I am really very grateful for all the people in my life, and although I did not mention them all, I still recognize the great impact they had on me and on my life and how I would not be here without them.

Contents

List of Figures

Chapter 1 Introduction

With the accelerating advancement of electronics, numerous problems and obstacles are emerging that present bottlenecks for improvements. One of those main bottlenecks is power consumption and its adverse effects. Power affects many aspects of digital system design, from thermal production to accelerated aging, power delivery to battery lifetime, and many others.

Given the many dimensions and impacts of power consumption, there are many approaches, tradeoffs, and target metrics for system optimization. In this work, the focus is on energy efficiency, where energy consumption is minimized given requirements for performance and/or other metrics. This helps increase battery lifetime with minimum impact on performance and functional capabilities. However, maximizing energy efficiency does not necessarily alleviate thermal or power delivery issues.

Energy efficiency is particularly important in emerging portable computing devices, which contain a limited power supply yet often have a performance demand comparable to plugged-in devices. Improving energy efficiency for those devices can help increase their usability significantly and make them applicable to new scenarios. Many of those new devices have extremely varying workloads, which can be exploited to increase their battery lifetime.

Energy consumption research has generated many partial solutions that target many different aspects of the problem. Many of those partial solutions focus on specific architectures or applications, such as creating specialized hardware like the highly used Graphics Processor Unit (GPU). Specialized hardware is very effective for the range of applications they cater to, but it can only be used when a big percentage of the system workload matches its target, which prevents the use of specialized hardware in many more general-purpose systems. What are needed are more generic solutions and design methods that can be broadly applied and benefit a much wider range of systems.

1.1 Background

There are many design methods that have been used in a variety of ways to maximize energy efficiency. This research explores three: finding the right balance of specialization and flexibility, applying dynamic voltage and frequency scaling to tradeoff energy and performance, and using a vertically integrated approach to co-optimize design decisions across system abstraction levels.

1.1.1 Specialization vs. Flexibility

Many designers have identified ways to design specialized analog and digital circuits to provide more energy-efficient implementations of specific functions. An example of a specialized block is the multiply and accumulate (MAC) [10], which many general processors and most signal-processing processors use. Another example is the streaming processor [11], where the whole processor configuration is optimized for specific types of applications, thus increasing the system's efficiency.

Another form of specialization is allocating resources based on known frequency of use. This is useful for applications that execute specific types of operations frequently, thus straining some components of a processor more than others. This is exemplified in GPUs [23], where a single fetching and decoding unit might cater to tens or hundreds of executing components. This way, the energy consumed for fetching and decoding is not repeated for the extra data points because they share the same instruction. This technique saves energy without sacrificing performance for data-parallel applications. This is why for graphics applications, this kind of specialization has made progress and speeds up at rates sometimes exceeding Moore's law. However, if an application's parallelism type is at thread level, then it requires a different kind of specialization. A better design to improve thread-level parallel applications is to have multiple cores sharing a cache to handle each thread separately.

As expected, those specialized designs are useful and energy efficient for a specific and narrow range of applications. However, once they are executing other types of applications, those specialized designs become less efficient than typical generic ones. Therefore, adding reconfigurability to switch between various specialized configurations is beneficial to a larger range of applications.

As part of this work's contributions, we aim to produce some general guidelines to help optimize the reconfigurability granularity.

1.1.2 Dynamic Voltage and Frequency Scaling

Reducing the voltage of circuits and devices remains one of the main methods for trading off energy per task and delay. Currently, almost all portable and many non-portable devices have low-power modes where the voltage supply, along with the performance, is reduced. The effectiveness of this technique is due to the quadratic relation between voltage and power and the linear relation between voltage and performance. Many devices use this technique mainly to reduce power consumption and curb thermal production, but it can also be used to increase battery lifetime by reducing the energy consumed per task.

Many challenges accompany the change of voltage supply. For example, the overhead of the voltage change can be significant, thus limiting the opportunities for the system to lower its voltage to save energy. Implementing this ability in systems affects many other aspects, like the need to have level converters and mechanisms for switching the voltage. Additionally, to determine the energy overhead of the voltage switching, numerous design decisions have to be considered.

Voltage scaling can vary in spatial and temporal granularity. Some techniques lower the voltage of the whole chip when running in low-power mode [12]. This technique has significant switching overhead and pretty coarse spatial and temporal granularity but saves area on control signals and has a better voltage supply grid since the whole chip is supplied by one big grid. Other techniques look at the component-level granularity where copies of the same component have different voltage sources, thus producing multiple performance-energy profiles of each component [13]. The system can take advantage of such multiple operating options depending on the performance required of the individual components.

As a contribution in this area, this work explores the various trade-offs of adding finer temporal and spatial granularity and identifies the sensitivity of several variables in those trade-offs.

1.1.3 Vertical Integration

Since system complexity is increasing steadily, it is convenient to define abstraction levels with complete design independence between each level. This method of designing is useful to scale to large systems. It is known, though, that many design decisions at any specific design level affect multiple other levels of the design. It is therefore not surprising that optimizing multiple design levels produces better results. The difficult part is identifying whether the extra complexity of co-optimization outweighs the potential benefits.

Vertical integration can take many forms, like synthesizing and placing multiple blocks together rather than each block independently. Another aspect of vertical integration is determining multiple design decisions related to different design levels at the same time, like the transistor width and the processor configuration. Even simply exposing low-level information to the system level is considered vertical integration.

This work contributes to the needed exploration of the challenges and benefits of such co-optimization and to identify some of the key information that should be exposed to different design levels to achieve maximum energy efficiency.

1.2 Problem Statement

This work is focused on developing design methods to maximize energy efficiency in digital systems using dynamic configurability and vertical integration and evaluating their effectiveness through case study designs.

1.3 Related Work

The idea of incorporating dynamic configurability and vertical integration for energy savings is not new. Wan et al. [6] presented some methods for designing reconfigurable DSP processors, showing that energy efficiency can be obtained. Similarly, Sen et al. [7] added dynamic configurability for the radio frequency transceiver to show that changing the performance rate according to workload demand provides energy savings and a more efficient system overall. Ahmadinia et al. [14] explored the interconnect

configurations compatible with dynamically reconfigurable devices. Other researchers have looked into dynamic configurability according to the dynamic workload. Yang et al. [15] tried to use configurability through workload prediction for energy savings. Others investigated application-specific voltage scaling; for example, Zhao et al. [16] where the authors created a specific reconfigurable architecture to deal with variable size FFT transformations. This variability increased around 14% increase in area, but it achieved energy savings reaching to ~94% relative to general purpose architecture execution. Pouwelse et al. [19] on the other hand focused on increased battery lifetime in wearable devices by dynamically changing the voltage level of the microprocessor. Wearable devices in general are very limited in energy consumption, but their performance requirements vary drastically from doing nothing to bursts of high performance. For such systems, varying performance to save energy is an ideal method to increase battery life. As expected, dynamically changing the voltage source is not simple to implement, Martin et al. [20] explains how, due to leakage power consumption in newer generations, voltage scaling is not achieving the hoped power savings. The authors suggest adjusting the body biasing as well as the voltage to mitigate the leakage issues related to voltage scaling. Zhang et al. [21] also tries to help in problems associated with introducing dynamic reconfigurability in processors. In their work, the authors present some interface and programming wrappers to simplify the use of dynamic heterogeneous processors like the Pleiades processor. Rana et al. [22] presents a general overview of some of the dynamically reconfigurable embedded systems. The authors also present the different models and architectures used to enable the dynamic reconfigurability.

Many researchers have identified the potential of dynamic reconfigurability and have begun to address the associated difficulties, but this work differs by focusing on the methodology for adding the configurability and employing vertical integration to investigate the main variables affecting the efficiency of the reconfigurable system. We are not suggesting a specific configurability; instead, we are attempting to help clarify the main aspects to investigate before adding dynamic configurability to a

system. In particular, we are trying to provide the main aspects to investigate in order to determine the suitable level of granularity.

In vertical integration, Jain and Panda [9] optimize the power consumption of the discrete wavelet transform, common in JPEG image compression formats, by co-optimizing the memory banks and architecture with the application memory access patterns. This co-optimization between software and circuit level hardware can increase overall efficiency significantly. Zea et al. [8] suggested co-optimization of the pipeline stages and an error-resilient mechanism to produce an overall more energy-efficient model. Jhaveri et al. [17] focused on commercial outcome optimization by co-optimizing the circuit alongside the layout and the lithography to minimize the cost of commercial production of the circuits. Iyengar et al. [18] also looked into co-optimization focusing mainly on high-level System on Chip (SoC) design.

We see that vertical integration and co-optimization is known to researchers as a beneficial tool for increasing energy and performance efficiency. Though our work is similar in nature to those works, as stated before, our focus is a vertically integrated approach to dynamic reconfigurability to achieve higher energy efficiency. We achieve this mainly by exposing circuit-level information to system-level tools.

1.4 Approach Overview

As mentioned above, work on energy saving techniques has been approached from almost every possible aspect. In this work, we focus on improving energy efficiency – for example, reducing energy per task within performance constraints. We aim to achieve that through targeting two generic aspects that can be applied to a very large range of applications and many other energy-saving techniques. The first aspect is identifying the optimal granularity for adding dynamic configurability to systems. The second aspect is vertical integration through exposing information to various hierarchical design levels, namely, circuit level and system level. The exploration of these two aspects is done through designing systems that utilize them.

To investigate the process of designing dynamically reconfigurable systems, we walk through the steps of designing and adjusting a few such systems. In the process of the design, we explored several aspects that significantly affect the energy efficiency of systems. The design analysis and results were mainly obtained through low-level accurate simulations; some of those results were validated and confirmed through physical measurements.

The analysis and results were obtained from three main projects that addressed one or more aspects of reconfigurability: adaptive compression algorithm for body sensors, panoptic dynamic voltage scaling (PDVS), and field programmable core array (FPCA). In the following sections, we will describe the overview of each project and the research questions we aimed to answer.

1.4.1 Adaptive Lossless Compression for Body Sensors

This project studies the benefit of adding configurability to compression techniques aimed to reduce the data transferred through the wireless components in body sensor devices. Those devices are typically small in size and very limited in energy source. Wireless communication is typically their main energy consumption source. In this project, we create a lossless compression technique that can adapt to the nature of the data collected through body sensors while using specific metrics that span over several design levels.

Project Overview

To try our compression technique and compare it with others, we implemented this technique in an MSP430 to show its simplicity. The analysis and results from this project were obtained by applying different compression techniques and algorithms on actual physical acceleration measurements taken through TEMPO [24]. The comparison was done against several techniques that provide similar features.

In our project, we focused on what other non-traditional metrics that we should consider that due to abstraction are usually disregarded. Those metrics include several specific characteristics typically found in resource constrained body sensor nodes.

Research Questions

The research questions here focused mainly on reconfigurability. This project was implemented to test whether adding configurability to low-power, low-performance electronic devices such as body sensors is significantly beneficial. Another question we wanted to answer whether other non-traditional aspects of the compression algorithms (e.g., implementability of the algorithm, size of cache etc.) need to be considered for small embedded systems.

1.4.2 Panoptic Dynamic Voltage Scaling

This project explores the spatial and temporal granularity of dynamic voltage and frequency scaling (DVFS), including evaluating the benefits and challenges of PDVS, which can change the voltage of a single arithmetic component for a single operation without affecting the performance of other components.

Project Overview

To explore the ability of PDVS voltage switching, we created a digital signal processor that is capable of running simple signal-processing applications. To enable the fast voltage switching for the components, we added switching headers for each of the arithmetic components in the processor shown in Figure 1-1. The headers allow us to switch the voltage of any arithmetic component with low time and energy overhead without any significant effects on the other components.

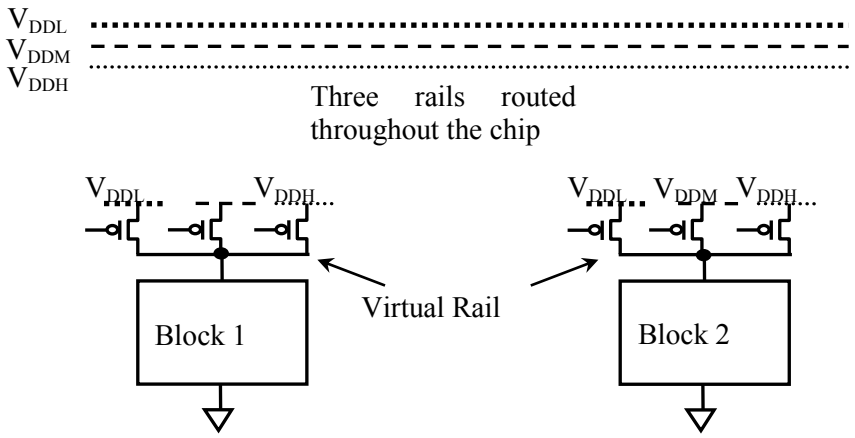


Figure 1-1: PDVS implementation using header switches and a set of shared V_{DD}s routed throughout the chip. The headers can be toggled dynamically during execution to switch voltages. [1]

Because we are interested in the effects of such techniques on a system level, we designed a chip capable of performing simple DSP applications. In our design, we had several adders and multipliers, each capable of switching to one of the three voltage rails. The other components, such as the registers and the multiplexers (muxes), are statically connected to the highest voltage rail. We added voltage converters in front of the arithmetic components to make sure that the data voltage level is compatible with the rest of the processor. In later sections we will describe the system in more depth.

Research Questions

This project covers several aspects of both dynamic reconfigurability and vertical integration. We aimed to learn more about the potential benefits of adding configurability at fine granularity as well as its potential complications. We explored what aspects to consider when adding this reconfigurability and how to resolve some of the additional decisions we had to make because of the configurability.

On the vertical integration front, we wanted to know how the header transistors' sizes and the voltages of the rail can affect the system scheduler and whether the energy consumed in those headers during the

[1] The graph has been taken from [4]

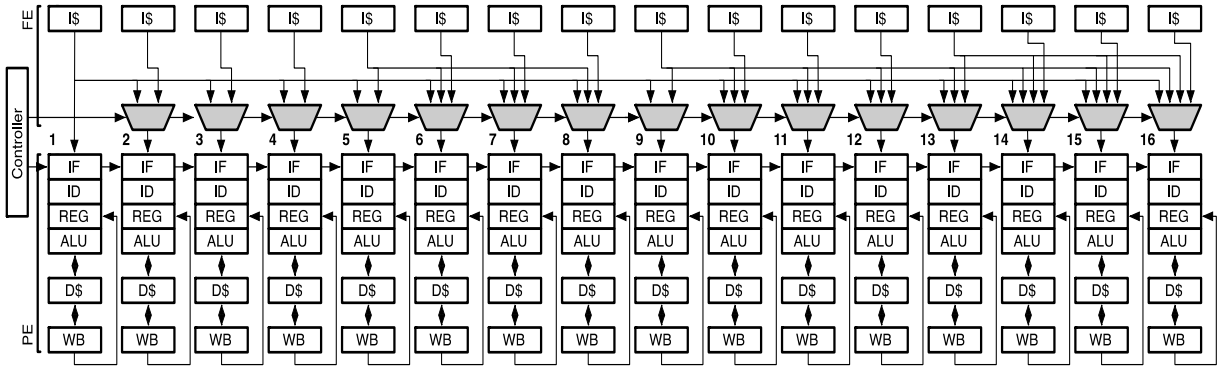


Figure 1-2: Block diagram of our designed reconfigurable SIMD/MIMD system. The reconfigurability is achieved at the interface between the I$ and the cores.

switching between voltage rails significantly affects the design decisions we had to make at the scheduling and binding level.

1.4.3 Field Programmable Core Array

In this project, we designed a dynamically configurable processor that can morph into several architectural configurations to maximize energy efficiency according to the application running. To achieve this, we created a multicore processor using multiple in-order processors divided into front ends (FEs) and processing elements (PEs). By having a configurable interconnect between those components, we can morph the design into different SIMD and MIMD architectures. Because different applications can vary in energy efficiency depending on the architecture, a configurable architecture can achieve better average energy consumption.

Project Overview

Designing the FPCA processor is not a trivial task. Therefore, in this work, we opted to create a dynamically configurable interconnect processor by using a simple off-the-shelf in-order processor and a simple reconfigurable interconnect. We chose OpenRISC as the in-order processor to use and a simple array of different sizes of multiplexers to achieve reconfigurability. The processor is a simple pipelined in-order processor that enables us to divide the processor into FE and PE.

As a final design, we opted to create a 16-core system capable of dynamically switching to four different SIMD/MIMD configurations illustrated in Figure 1-2. The analysis in this project was performed using

various simulators from various levels; circuit-level simulations were used for various components and wires energy consumption estimations, RTL tools were used for interface and connections between components information, and architectural simulators were used to estimate the utilization and performance of the different stages of the processor across different benchmarks.

Research Questions

Our aim for this project included learning the benefits that processors can gain from introducing this coarse grain granularity and how much flexibility such a system should have to achieve maximum energy efficiency.

Another aspect that was investigated through this project is how circuit-level information and simulations can affect high-level design decisions. The use of many simulators across the broad use of design hierarchy showed interesting effects and complexities in optimizing such systems. We analyzed the impact of energy consumed in wires on the final system architecture, as well as the multiplexers overhead from the system configurability.

1.5 Document Structure

The remainder of the document is organized as follows:

Chapter 2 talks about adding configurability at the software level for low power embedded devices like body sensors. In that chapter, we will explain the other parameters needed to be considered when adding configurability while exploring the potential benefits of adding it.

Chapter 3 explains the various tradeoffs we encountered when introducing voltage reconfigurability at a single component's level in processors. We describe the various affected design levels and what kind of optimization we had to go through to maximize the energy efficiency of the overall system.

Chapter 4 describes the design decisions and the analysis of the parameters we undertook while creating a reconfigurable processor that is capable of dynamically switching between various SIMD and MIMD configurations.

Finally, Chapter 5 describes briefly the summary of results and the contributions of the projects in this work. We present simple guidelines and recommendations learned through this work as well as research questions and extensions to this work that might be the target of future work.

Chapter 2 Reconfigurability at Software Level[2]

In most wireless body area sensor network (BASN) applications, the vast majority of the total energy is consumed by the wireless transmission of sensed data. Reducing this transmission energy—even at the expense of increasing another component's energy—is essential to meeting the battery life and form factor (i.e., small battery) requirements of many BASN applications. While improved wireless communication and networking techniques can help do just that, simply compressing the sensed data to reduce the number of transmitted bits can provide significant savings. However, BASN platforms and applications impose many constraints on compression techniques, including fidelity (focus on lossless techniques, as required for many medical BASN applications), programmability (enable ease of code development and deployment), adaptability (achieve high compression ratio regardless of location, subject, activity, etc.), and implementability (require low processing and memory resources). This chapter analyzes variations of two known real-time lossless compression algorithms—Huffman encoding and delta encoding—within the context of these BASN constraints. Experimental results on a multi node accelerometer–based BASN show the strengths and weaknesses of each algorithm and ultimately reveal the superiority of dynamic delta encoding for BASNs, including an average of 35% energy savings across a range of activities, sensor locations, and sensor axes.

2.1 Motivation and Background

Wireless BASNs are emerging as a technology with tremendous potential for a variety of applications, including health care, clinical medicine (including telemedicine), biomedical research, emergency medicine, first responder safety, homeland security, athletics, and others. Although significant efforts have been made to develop and deploy BASNs, there are numerous technical challenges that remain in order for BASNs to be practical for the applications for which they are envisioned. In particular, BASN nodes must be smaller to minimize invasiveness and maximize wearability and must have longer battery

[2] Most text and data in this chapter are taken from [1]

lifetimes to enable significant data collection. However, these are competing metrics because the size (and therefore capacity) of the battery is the primary factor in determining the dimensions of a BASN node. Although improved energy harvesting and storage provide promise for the future [47], the options immediately available to BASN developers involve improving energy efficiency—especially energy related to the wireless transmission of sensed data, which is the largest energy consumer in most BASN systems.

Although work is being done to improve the energy efficiency of wireless transceivers and to explore the energy versus quality-of-service trade-offs in communication coding and wireless networking protocols, the most direct way to reduce energy consumption due to wireless transmission is to simply reduce the number of bits that need to be transmitted. On-node signal-processing algorithms, such as feature detection and pattern recognition, can be used to convert some of the raw sensed data into application information that can be coded using fewer bits, but most of these algorithms are application specific. It is desirable from a programmability and deployment perspective for BASN devices to be general enough that a single device executing the same code can be efficient for a wide range of applications and even sensor locations and activities within applications. Even within a particular application, the critical data/information cannot always be reliably determined/extracted, so any such preprocessing may be problematic for fidelity-critical applications, such as the many medical applications for which BASNs are envisioned. Therefore, lossless compression techniques are the most general and reliable tools for reducing wireless transmission energy in BASNs.

However, given the severe resource constraints of most BASN nodes, any compression algorithm must have low processing and memory requirements while still providing real-time throughput (i.e., the processing must keep up with the data sampling rate) and overall energy savings (i.e., the additional energy consumed by the processor running the compression algorithm cannot exceed the wireless transmission energy saved). Resource-aware compression techniques have been developed for traditional wireless sensor networks (WSNs). In [43] the authors show how wirelessly transmitting 1 bit can

consume about 1000 times the energy of a single computation, which motivates the use of compression techniques on the nodes before transmission. In [45][51] the authors present different ways to save energy in wireless nodes, including data routing techniques combined with compression algorithms to reduce the transmission energy consumption. In dealing with wireless BASN nodes, we have to be aware of the severe constraints. The embedded processors on BASN nodes typically operate in the tens of megahertz and have on-chip memories of only several kilobytes. Efforts to increase total memory by including off-chip memories have been proposed [50], but off-chip access time can be quite high, and the additional resources (particularly area and power) may not make this approach worthwhile, especially in resource-scarce BASNs. Finally, the characteristics and requirements of many BASN applications are significantly different from most Wireless Sensor Network (WSN) applications, including both static parameters (e.g., higher data rates and fidelity requirements) and dynamic variables (e.g., rapidly varying data properties and channel conditions).

This chapter provides an analysis of resource-aware lossless compression techniques specifically targeting BASN characteristics and requirements. The following metrics are considered in the analysis:

- Compression ratio: number of bits before compression divided by the number of bits after compression
- Processor cycles: number of cycles that the processor needs to run the compression algorithm for each sample, which is proportional to the processing energy consumption of the algorithm
- Average energy savings: combination of the previous two metrics that is estimated from the data sheets of common BASN transceivers and embedded processors
- Memory requirement: approximate amount of memory required to implement the algorithm
- Adaptability: ability of the compression algorithm to adapt to static parameters (application, wearer, sensor type, sensor location) and dynamic variables (activities, time), continuing to provide high performance without negatively affecting the other metrics of interest

- Programmability/deployment: related to the previous metric, the ability to adapt to static parameters enables the developer to program and deploy all nodes the same way, regardless of the application, wearer, sensor type, and sensor location

Background work considered a number of lossless compression techniques, but this chapter focuses on two of the most promising with respect to the metrics of interest—delta encoding and Huffman encoding. Delta encoding transmits the difference between each reading rather than the full reading. If the number of bits required to encode the delta (referred to as delta bits) is regularly less than what is required to encode the full reading, significant compression ratios can be achieved. The compression can be made lossless by including a special code after a reading that exceeds the representable delta range. The number of delta bits can be determined statically or dynamically, and both approaches are considered here. Huffman encoding depends on some readings occurring more than others, so by assigning frequent reading codes with fewer bits, the total number of transmitted bits will decrease. This chapter compares both algorithms and some of their derivatives within the context of BASNs and with respect to the above metrics.

The work detailed in this chapter included the consideration of a number of lossless compression algorithms. The two families that were identified as the most promising given the BASN metrics detailed above are Huffman encoding and delta encoding. Several variations of each are evaluated in Section 2.4.

2.1.1 Huffman Encoding

Huffman encoding leverages the uneven distribution of readings in data sets, using fewer bits to encode more common readings to achieve an overall compression. An imbalanced tree structure is generated based on the presumed frequencies of each reading, with high-frequency readings at shallower leaf nodes than those occurring less often. Each reading's code is determined by traversing the tree from root to leaf, with each branch node providing one bit to the code. The length of each code is therefore determined by the depth of its leaf.

A number of practical issues make the use of Huffman encoding a challenge for BASNs. First, Section 2.1 reveals that many of the reading frequency distributions (using the accelerometer-based BASN platform and multiple sensor locations and activities) are relatively flat, thus limiting the compression capabilities of Huffman encoding. Second, although the computational complexity of the Huffman algorithm, as it is running with an existing tree, is small, the tree itself can be quite large and potentially exceed the memory constraints of many BASN-embedded processors. This is especially problematic in lossless compression when every possible reading must be encoded and must therefore have a leaf in the tree, even if that reading is extremely rare. Finally, traditional static Huffman encoding depends on the existence of a reading frequency distribution to generate the tree, and the compression performance achieved depends on how well the dynamic data conform to that frequency distribution. It is therefore essential that a BASN developer wanting to use Huffman encoding performs extensive data collections to profile the reading frequency distribution. As discussed, this can be extremely difficult given the numerous static and dynamic variables. Therefore, this static technique's ability to perform well across many applications, wearers, sensor locations, sensor axes, and activities is limited.

It is therefore desirable to also consider an adaptive Huffman encoding technique that dynamically generates and alters its tree based on the actual reading frequency distribution as it occurs and changes. This does not require a previously constructed tree or any reading profiling. In the tree, each reading will carry its value along with the frequency of its use. This way, the tree can update itself, keeping the most frequent readings on the shallower levels to minimize their code lengths [46]. This adaptability can potentially provide higher compression performance across all of the static and dynamic variables without additional programming and deployment effort. However, as discussed previously, the performance of adaptive Huffman encoding in a sample BASN application is limited because of a number of factors.

Although this adaptive technique is significantly more computationally complex than static Huffman encoding, it can potentially be implemented in real time on BASN-embedded processors, although the number of processor cycles (*cycles* in Equation 3.2) required per reading may be relatively high.

Although adaptive Huffman encoding includes the storage of reading frequencies in addition to the coding tree, its total memory requirements are often smaller than those of the static technique. The static tree remains a fixed size in memory throughout execution regardless of the occurrence (or lack thereof) of certain readings. The adaptive technique can choose to discard certain readings that have not occurred for some time, keeping the tree size to some maximum memory requirement and reinserting a discarded reading should it reoccur in the future [54].

2.1.2 Delta Encoding

Delta encoding achieves compression by sending the difference between a reading and its predecessor rather than sending the full reading. This technique has been used effectively for a variety of applications, from images to Web pages [48]. The compression rate is determined by the difference between the number of bits designated for conveying the difference (delta bits) and the number of bits required for the full reading. This technique is often used in lossy compression, as differences that exceed the range that can be represented by the number of delta bits may occur. However, delta encoding can be made lossless by including a special overflow code in place of the difference, followed by the full reading. This lossless algorithm has been used here for BASNs.

One of the challenges for both the lossy and lossless versions is the selection of the number of delta bits to be used. If too few bits are used, the lossy delta encoding will become extremely lossy (too many readings will be beyond the encoding range), and the lossless encoding may actually have a compression ratio that is less than one (many readings will both be transmitted in full and include the special overflow code). If too many are used, the compression ratio will be lower than what is possible. Like as is required to determine frequency distributions for the static Huffman tree, the BASN application must be profiled, and the collected data must be analyzed to determine the optimal number of delta bits. This again adds significantly to the programming and deployment effort and is still limited by dynamic variables, such as different activities over time.

Delta encoding can also be made capable of adapting to static and dynamic variables, much in the same way as adaptive Huffman encoding, but with significantly lower complexity. For a given interval of time (or number of samples), dynamic delta encoding determines whether it would have been better to use a different number of delta bits, and it sets the number of delta bits for the next interval accordingly, including a special code to indicate to the receiver that the number of delta bits has been changed. Equation 1 calculates the number of bits needed for encoding an interval of samples:

$$Bits = I * DB + OF * FR, \quad (3.1)$$

Where I is the number of samples in each interval, DB is the candidate number of delta bits, OF is the number of samples that results in overflow given DB, and FR is the number of bits in a full reading. Using this equation, the processor reevaluates DB at every interval, always adjusting to maximize the compression ratio.

The computational and memory requirements of this technique are slightly higher than static delta encoding but are significantly lower than both static and adaptive Huffman encoding. The complexity depends on the range considered for delta bit alteration. According to our results, this study determined that ±1 delta was sufficient to provide high performance.

2.2 Metrics

Normally, to evaluate a compression technique, only the compression ratio is taken into consideration. However, like in WSNs, compression in BASN devices must consider several other metrics [53]. In this section, we detail a mix of metrics that provides an overall performance evaluation for compression techniques within the context of BASNs.

2.2.1 Energy Savings

Because the main point of implementing a compression algorithm directly on BASN sensor nodes is to reduce energy consumption, a formal energy savings equation must be introduced. Given that the embedded microprocessor on the sensor node may be in a sleep mode if it were not being used for

compression, the energy equation must include both the reduction in transmission energy due to the reduced number of transmitted bits and the increase in processing energy. Given that compression does not affect other sources of energy consumption (e.g., the energy drawn from the sensors), only those two sources are considered here.

The average energy savings per sample is simply the difference between the energy before and after compression:

$$E_{Before} = RES * E_{Bit} + cycles * E_{Idle} \quad (3.2)$$

$$E_{After} = \frac{RES}{CR} * E_{Bit} + cycles * E_{Active}, \quad (3.3)$$

where E_{Before} and E_{After} are the energy consumption per sample per sensor before and after the compression, respectively; *RES* is the number of bits for each reading (i.e., bits per sample per sensor); *CR* is the compression ratio; E_{Bit} is the energy required to transmit one bit wirelessly; *cycles* is the number of active processor cycles that the compression algorithm needs to compress one reading; and E_{Idle} and E_{Active} are the energy per idle and active processor cycle, respectively. *RES*, E_{Bit}, E_{Idle}, and E_{Active} are specific to the BASN platform—specifically the transceiver and the embedded microprocessor; *cycles* is a function of both the compression algorithm and the microprocessor, and *CR* is a function of the compression algorithm and the actual data. This chapter uses a custom accelerometer-based BASN platform in a motion-capture application to obtain all of these values.

Even within a given BASN platform, E_{Bit} may vary based on a number of factors. Wireless devices typically stay connected all the time, consuming energy constantly, and the energy consumption surges during transmission. In Equations 3.2 and 3.3, E_{Bit} represents the difference between the "stay connected" energy and the "transmit" energy. Many factors and overheads affect the transmission energy; namely, for Bluetooth burst transmission, the transmission rate and packet size affects greatly the energy consumed per burst, which affects the energy consumed per bit. This chapter assumes a transmit rate of 115.2 kbps and the maximum packet length, which minimizes the per-bit transmission energy in our system. This

maximum packet length will be maintained regardless of the compression ratio, so E_{Bit} is reduced because of the lower packet rate.

Many embedded microprocessors are programmed to go into a sleep mode (i.e., a very low-power mode) when no processing is required, so the additional active cycles required—and processing energy consumed—by the compression algorithms are considered in the energy savings as $E_{Active} >> E_{Idle}$. However, in many systems, the overhead of going to sleep and waking up does not justify the often short amount of time spent in the sleep mode. In such cases, the processor will perform "No Operations" (NOPs) until functional processing is again required. NOPs often consume almost as much energy as a functional cycle, so E_{Idle} almost equals E_{Active}. In such systems, the additional processing energy consumed by the compression algorithm is negligible.

In most BASN systems, $E_{Bit} >> E_{Active}$, often by several orders of magnitude. However, if the processing load imposed by a compression algorithm is significant, the additional processing energy can be significant. For example, Algorithm A may provide a higher compression ratio than Algorithm B, but if the complexity of Algorithm A is significantly higher, Algorithm B may actually provide greater total energy savings.

2.2.2 Resource Requirements

Because any compression algorithm will be implemented on the embedded processor, the algorithm will be constrained by the processor's limited resources. The processors used on BASN platforms are typically significantly smaller and less capable than those used in WSNs, making algorithm implementability a significant constraint.

The processor's memory imposes one of the key constraints because many embedded processors that are appropriate for BASNs have only a few kilobytes of memory. Algorithms that use tables, trees, or dictionaries could easily exceed this memory restriction and therefore be excluded from use in BASNs.

Algorithms must also operate under the limited throughput capabilities of the processor, which typically has a relatively simple datapath and runs at tens of megahertz. Depending on the sampling rate of the BASN platform (the accelerometer-based node used in this study samples three sensor axes at 120 Hz, resulting in 360 readings per second), it may be difficult for a compression algorithm to be executed in real time, which is a hard requirement.

2.2.3 Adaptability

It is highly desirable to have a compression algorithm for a BASN platform that will perform well for any application, wearer, sensor location, activity, and so on. Although different compression algorithms can be programmed onto each BASN node based on these factors, this requires significant additional programming and deployment effort, not to mention the challenge of profiling that would be required to determine the appropriate compression algorithm for each scenario. In addition, the data being collected by a BASN is rarely static, as the wearer is often performing different activities that change the compression capabilities of the implemented algorithms.

Instead of using such static techniques, BASN compression algorithms should have the ability to adapt and perform well across different

- applications,
- test subjects (wearers),
- sensor locations and orientations,
- axes of the same sensor, and
- activities over time.

As shown in Section 2.4, the performance of compression algorithms across these static and dynamic variables can vary greatly. Although traditional WSNs also suffer from this problem because of node location and dynamic data, the effects are typically more extreme in BASNs. The overall energy efficiency of a BASN, therefore, depends on identifying the algorithm that performs the best on average across an entire data collection.

2.3 Experimental Setup

To compare between the identified compression techniques, we used the TEMPO BASN [52] shown in Figure 2-1, which measures linear acceleration in three axes. The sampling frequency is 120 Hz, and the resolution of each sample is 12 bits per channel. Without any on-node compression, each node sends its 4320 bits per second over a Bluetooth wireless channel. The compression algorithms were implemented on the resident MSP430F1611-embedded microprocessor. (The code is available online at http://www.ece.virginia.edu/inertia/embedded.php.)

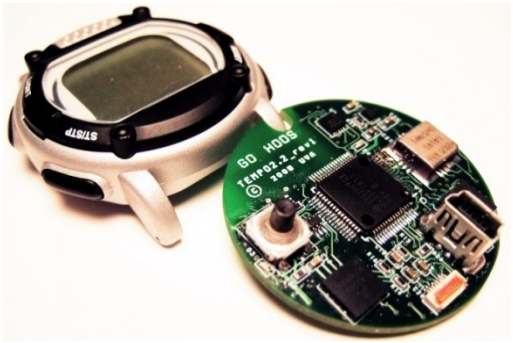


Figure 2-1: TEMPO BASN node showing the small form and the limited energy source

TEMPO nodes were attached at multiple points on the body (including the wrists, ankles, hip, and forehead) of a healthy 22-year-old male. The results in Section 2.4 are for both a 45-minute recording of various movements and activities and shorter recordings of specific activities. The compression techniques described previously were implemented and evaluated with respect to the metrics described in Section 2.2.

2.4 Sweeping Variables

In this section, we vary each of the variables we identified to be significant in determining the energy efficiency of the systems and investigate the impact and the breakeven points between them.

Through simple extrapolation and sweeping of the ratio between CPU active energy per cycle and idle energy per cycle and sweeping the energy of transmitting 1 bit to 1 idle CPU cycle, we were able to calculate the compression ratio that different algorithms need to achieve to break even energy

consumption. The figure below shows the compression necessary to achieve energy breakeven according to different systems. Each line represents a specific hypothetical system with specific active to idle CPU energy consumption ratio and 1 bit transmission to 1 CPU cycle energy ratio. For example we are working on a system that has "active to idle" cycle energy ratio of 2 and transmitting 1 bit consumes energy equivalent to 100 idle CPU cycles, and the compression algorithm takes 200 cycles to compress 8 bits, then it should be able to compress the data to at least 70% of its original size to breakeven with sending the raw data without compression. In general we want to be under the line that most closely represents the characteristics of our system, namely, the "Active to Idle" cycle energy ratio and "1 bit Transmission to 1 Idle CPU cycle" energy ratio.

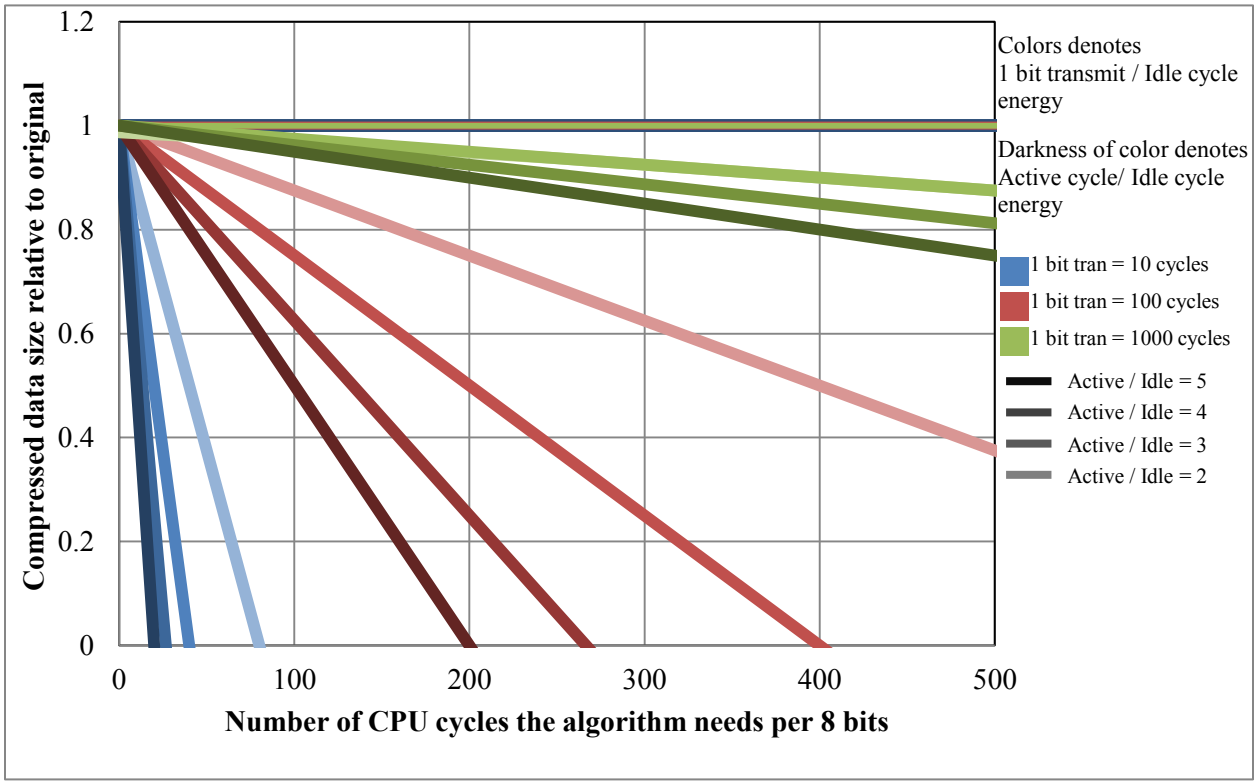


Figure 2-2: Graph showing the compression that should be achieved by the algorithm to breakeven in energy consumption.

2.5 Results

Figure 2-3 shows the average compression ratios for each axis of selected sensor locations across the entire 45-minute data set for adaptive Huffman, static Huffman, and dynamic delta encoding. The tree for

the static Huffman was generated based on the probability distribution of the readings over the entire 45-minute data set. Figure 2-4 shows the compression ratios for selected sensor locations for a 15-second window of healthy symmetric gait. The static Huffman tree was generated from the probability distribution over this specific 15-second window. We also show in the figure the results from the compression algorithm we concluded to use, dynamic delta encoding, which will be described later.

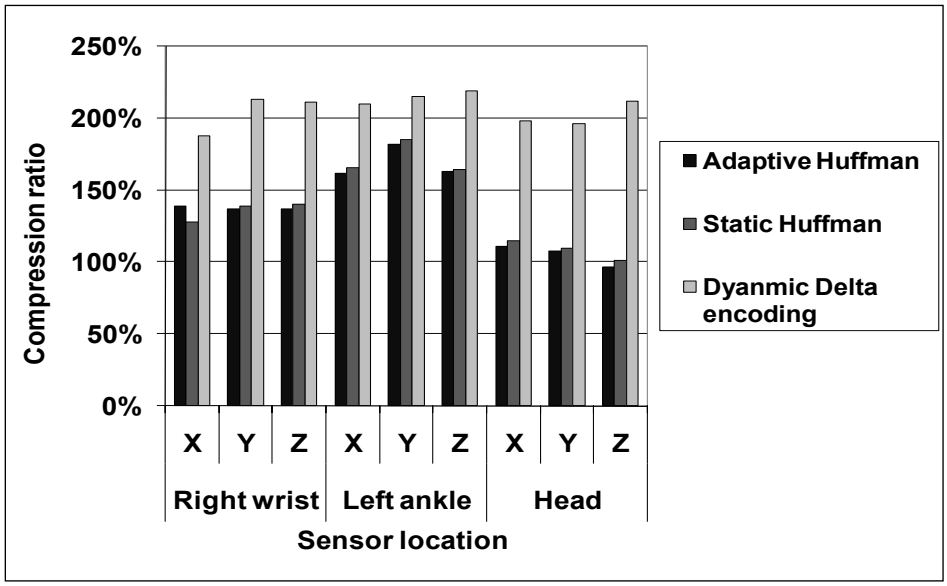


Figure 2-3: Compression ratios for the entire 45-minute recording

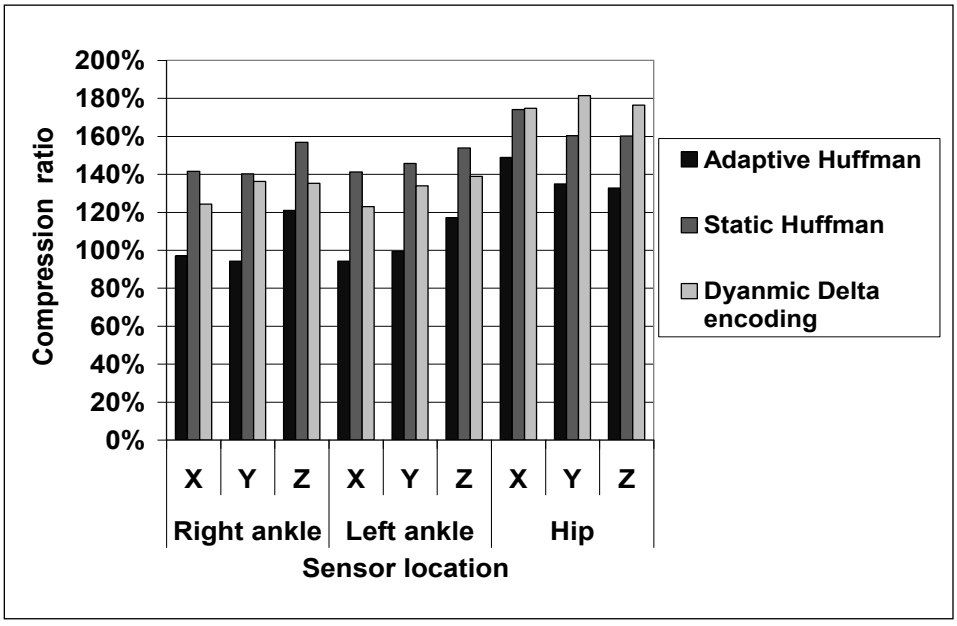


Figure 2-4: Compression ratios for 15 seconds of normal gait

2.5.1 Huffman Encoding

Huffman encoding is an old compression technique that has been adjusted and used numerous times. In this work, we investigate two widely used variations of Huffman encoding: static and adaptive Huffman.

2.5.1.1 Static Huffman

As shown in Figure 2-3, across the entire collected data set, sensor locations, and axes, static Huffman encoding was able to achieve an average compression ratio of approximately 135%. Although the Huffman tree was constructed based on the actual data set, it was not tailored to each of the individual activities and movements over the 45-minute period. Therefore, the frequency distributions of readings may have matched well during certain periods with the probability distribution used to generate the static tree (as was the case in Figure 2-4, when an average compression ratio of almost 160% is achieved). But that is not always the case, and the overall compression ratio suffers from this lack of adaptability. As expected, it was difficult to identify a single static probability distribution that was appropriate across a wide range of activities, axes, and locations. So the overall frequency distribution was used. It is possible to generate a number of Huffman trees from profiled probability distributions and invoke them at the appropriate time and for the appropriate sensor location and axis, but this is problematic with respect to programmability and deployment.

It is interesting to note in Figure 2-4 that Huffman encoding did better than dynamic delta encoding for the ankle sensors, but not for the hip sensor. The hip acceleration is much less than that of the ankles and therefore requires few delta bits for encoding.

Another major issue regarding static Huffman is that it requires a relatively large amount of memory. For lossless compression, each reading value must have its own dictionary index, so a 12-bit resolution system like TEMPO has 4096 indexes. However, some applications may assume that only a subset of the reading values actually occurs and simply assign one more index for all other values. The data collected during this study revealed that less than 1024 of the possible readings occurred, but that still requires that a large number of leaves and the accompanying tree structure be stored in memory. Some techniques can

be used to minimize the memory requirements for such structures, but this is still likely to be problematic for most BASN platforms. In addition, the complexity of the tree search is high, which increases the number of active processor cycles (*cycles* in Equation 2), and can cause the throughput constraints of the system to be violated.

2.5.1.2 Adaptive Huffman

The adaptive Huffman technique is designed to address the static technique's lack of adaptability with the goal of dynamically tailoring the Huffman tree to the current reading frequency distribution. However, the compression ratios provided by adaptive Huffman encoding were mixed based on (1) the amount of time it took for a tree to be adapted versus the amount of time the new tree provided good performance and (2) the distribution of the readings to be encoded, with more even distributions providing lower compression ratios.

Both of these factors come into play when examining the performance of adaptive Huffman in Figure 2-3. Given the adaptability of this technique and the various activities that were performed over the 45-minute data collection (and the resulting various reading frequency distributions), one might expect that adaptive Huffman would perform significantly better than the static version. However, it is clear that the activities and resulting distributions were not held constant long enough to enable the Huffman tree to adapt to them and provide an extended compression ratio benefit. This is revealed in Figure 2-4, as the 15 seconds of walking does not provide enough time for the adaptive technique to settle on the appropriate tree and ultimately benefit from it. In fact, the compression ratio is <1 for some sensor axes. It is likely that longer-term activities would result in better adaptive Huffman performance. In addition, some of the activities had relatively even reading frequency distributions on some axes and locations, resulting in lower compression rations regardless of adaptability. Finally, the memory and processing requirements of adaptive Huffman encoding also pose challenges, as described in Metrics section before.

2.5.2 Delta Encoding

It is clear from Figure 2-3 that dynamic delta encoding provides by far the best compression ratio for every sensor and axis over a long data collection that includes a variety of activities. Figure 2-5 shows how the number of delta bits changes for different activities over the 45-minute data collection (data from the right wrist), demonstrating the algorithm's ability to adapt to the current activity and achieve the highest compression ratio without having to change the algorithm. Figure 2-4 shows that it also does well over short periods of a single activity, even performing better than the optimized static Huffman for the hip sensor for the reason mentioned above.

In addition to providing the best compression ratio, it also requires little memory and the fewest processor cycles, further adding to its energy-saving capabilities relative to static and dynamic Huffman. Finally, it uses the same simple algorithm regardless of the application, test subject (i.e., BASN wearer), sensor location, sensor axis, or activity, making it extremely flexible and adaptable and easing the programming and deployment. It is therefore the conclusion of this chapter that delta encoding is the best algorithm for lossless compression in BASNs. The question, therefore, becomes one of optimizing dynamic delta encoding for the metrics detailed in Section 2.2.

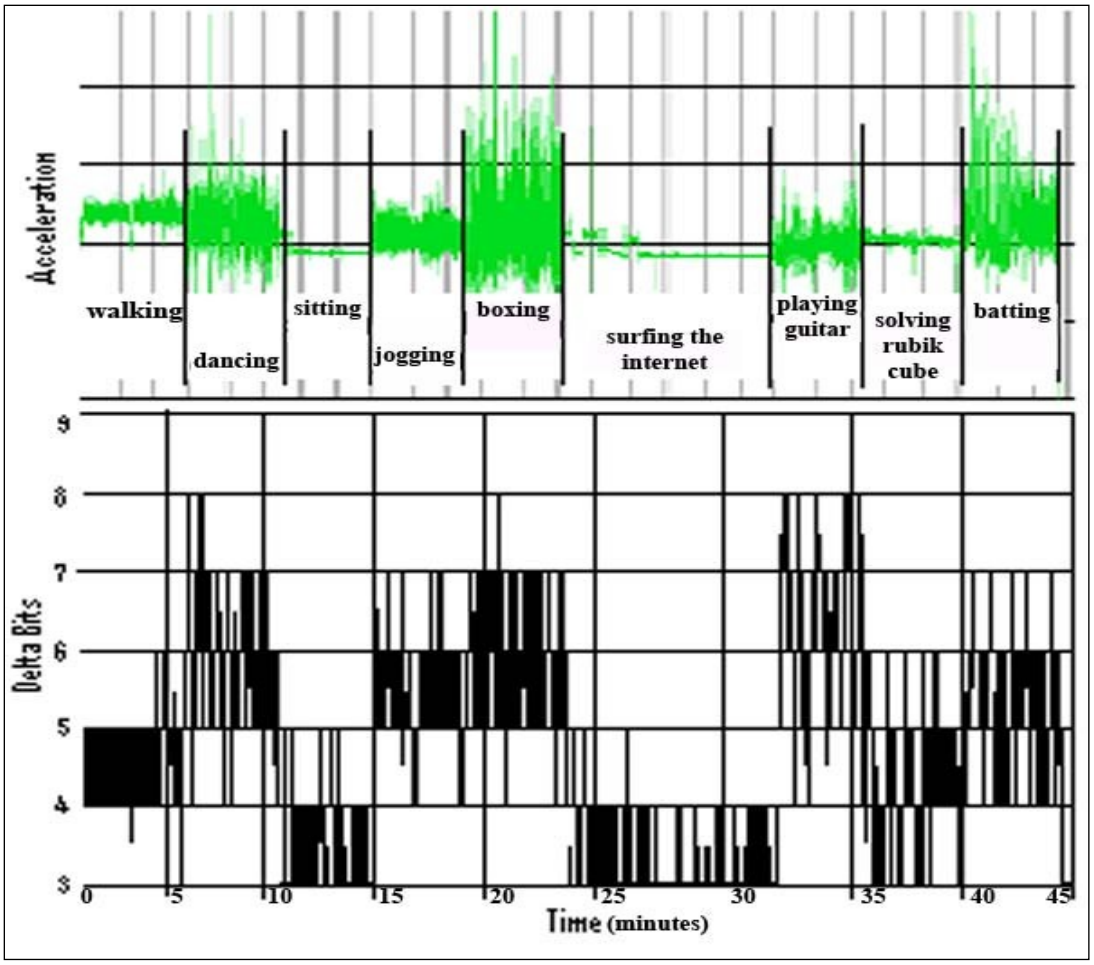


Figure 2-5: Number of delta bits across different activities

The algorithm changes the number of delta bits at every predetermined interval and does so by inserting a special code, which incurs an overhead. Therefore, changing the number of delta bits too often may reduce the compression ratio or cause rapid oscillations between settings. However, not changing often enough may reduce the algorithm's ability to adjust to rapid changes in the data characteristics. Figure 2-6 shows the compression ratio for the three axes on the right wrist over the 45-minute data collection period as a function of the delta bit update interval. It is clear that the optimal interval for all three axes lies between 0.25 and 1 second, and 0.5 second was selected for the rest of the results in this chapter, including those in Figure 2-3 and Figure 2-4. However, this optimal interval is data (and therefore BASN system, application, wearer, sensor, and axis) dependent, and the proper selection of the update interval is essential to compression performance.

The fact that the number of bits to represent each piece of data changes categorizes this algorithm under variable length coding. It is important to consider this characteristic when designing the transmission system to use. A known downside of variable length coding protocols that if one bit was wrongly transmitted (1 bit flip) then the rest of the streaming bits will be parsed wrongly. In our case, if a bit was erroneously flipped during sending the special code the indicates the change of delta bits, then the receiver will think the delta bits per sample did not change and it will continue parsing the stream of bits according the old number of delta bits. The Huffman algorithm faces the same problem since data can be represented by different length of bits. We do think Delta encoding is slightly more robust than Huffman in this aspect because it only changes code length during special occasions (i.e. changing number of delta bits or overflow) unlike Huffman where each data might be of a different length than the one before it.

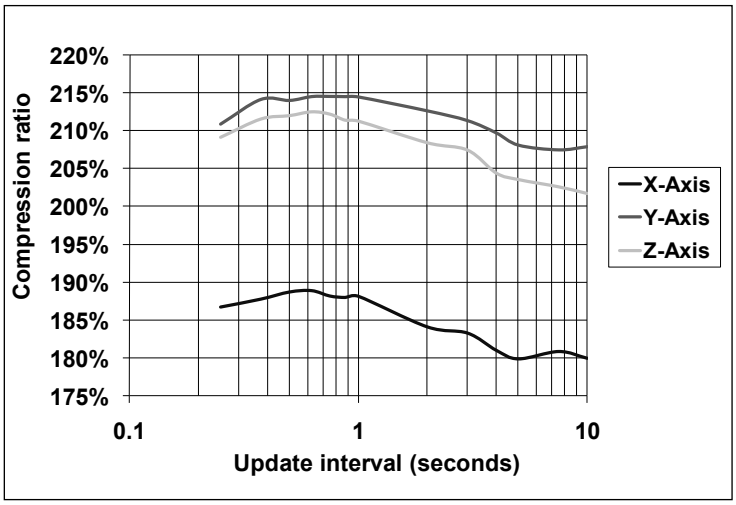


Figure 2-6: Compression ratio across different update intervals

Another interval-related issue is how large changes in optimal delta bits are handled. Consider, for example, the situation when the current number of delta bits is set to four and the data suddenly changes to include large deltas that require six or more delta bits for representation. Using Equation 3 and the ±1 delta bit options, it may be determined that the number of delta bits should be reduced by one rather than increased by one because the new deltas will overflow any of the available choices—three, four, or five delta bits. Given that every reading will result in overflow, the highest compression ratio will be provided by the fewest number of delta bits. Three approaches were considered to address this issue. First, the

maximum change in the number of delta bits could be increased to ±2 or ±3, but that dramatically increases the computational complexity of the algorithm and is unlikely to provide a significant benefit for real BASN data streams. Second, the update interval could be increased in the hope that any dramatic change in delta sizes is not long lasting, but this cannot be guaranteed. Third, Equation 3 could be altered to only make changes in the number of delta bits when the benefits of the change exceed a defined threshold, but that does not guarantee that the algorithm will converge on the optimal number of delta bits over time. Combinations of these three methods were investigated, but none improved the compression ratios more than 2%.

Figure 2-7 compares the performance of dynamic delta encoding with two other variations of delta encoding. The first, passive delta encoding, always uses seven delta bits. Seven was selected based on an analysis of the deltas for all of the sensor locations and axes from several data sets. The second, optimized delta encoding, also uses a constant number of delta bits, but that number was determined for each individual sensor location and axis based on the deltas in the 45-minute data set. Neither technique benefits from dynamic adaptation nor therefore achieves the compression ratios provided by dynamic delta encoding. Optimized delta encoding approaches that of the dynamic algorithm, but it has issues related to programmability and deployment. It is impractical to individually characterize and program every sensor location and every axis, especially because that process will likely have to be performed for every BASN platform, application, and wearer. Dynamic delta encoding provides better compression while enabling the same algorithm to be pervasively implemented.

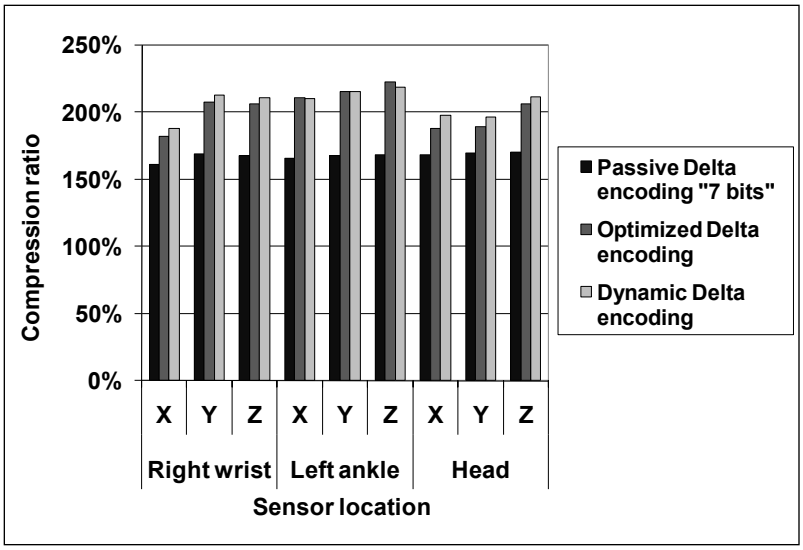


Figure 2-7: Compression ratios for delta encoding

While neither of these alternative delta encoding techniques benefit from dynamic adaptation to activities, they are both simpler algorithms (no dynamic decision making and no need to keep track of the number of overflows) and require fewer processing cycles and less memory as a result. The average number of processor cycles per sample on the MSP430F1611 to execute dynamic delta encoding was about 95, whereas the two static techniques (passive and optimized) required only 75.

Figure 2-8 shows the percent energy savings provided by the three delta encoding techniques over the uncompressed baseline. As specified in Equations 1 and 2, these results take both transmission energy and processor energy into account. The transmission energy per bit and processor energy per active cycle were taken from the data sheets of the Bluetooth module [44]and the MSP430F1611 [49], respectively. It is interesting to note that when processor energy is taken into account, the optimized delta encoding sometimes provides slightly higher energy savings than the dynamic algorithm. However, the programmability and deployment issues remain and prevent optimized delta encoding from being practical for most BASN applications. Dynamic delta encoding provides nearly the same energy savings while being much easier to program and deploy.

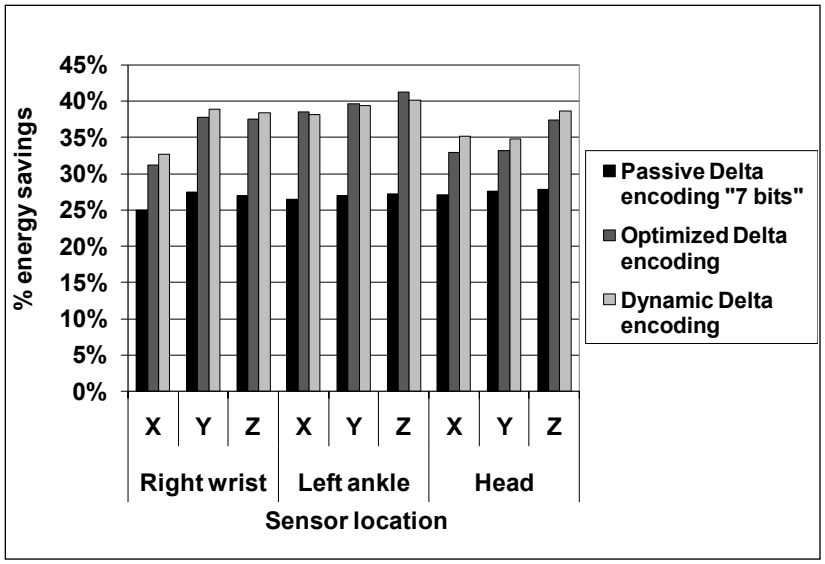


Figure 2-8: Percent energy savings for delta encoding

2.6 Conclusion about Software Reconfigurability

Given that the vast majority of energy consumption in most BASN platforms is due to the wireless transmission of sensed data, pre-transmission data compression is one of the most direct and high-impact ways to increase the energy efficiency of BASNs. However, the use of compression comes with trade-offs and constraints, especially given the extreme resource limitations on BASN nodes and the many static and dynamic variables associated with various BASN applications, wearers, sensor locations, sensor axes, dynamic activities, etc. First, the compression algorithm must fit within the limited memory of a BASN embedded processor while providing real-time performance. A typical instruction cache size for several commonly used platforms ranges from only 8KB to 32KB [67]. Second, given the difficulty of differentiating important from unimportant data and the criticality of many target BASN applications (e.g., those in the medical domain), lossless compression is desirable and can be made application independent. Finally, in order to provide significant energy savings, the algorithm must maintain a high compression ratio regardless of the static and dynamic variable settings and without significant additional programming and deployment effort. That is, it is highly desirable to program every sensor node the same way without profiling and without consideration of application, sensor location, etc.

This chapter evaluated two families of lossless compression techniques—Huffman encoding and delta encoding—within the context of these BASN requirements using a custom accelerometer-based BASN platform within a motion-capture application. Both static and adaptive/dynamic variations of these techniques were considered. Results revealed that dynamic delta encoding provided the best combination of energy savings (including both reduced transmission energy and increased processing energy), low memory requirements, high performance across a range of activities and sensor locations/axes, and low programming and deployment effort. The approximately 35% average energy savings provided by dynamic delta encoding can go directly toward extending a BASN platform's battery life and/or reducing the required battery (and total BASN node) size.

While dynamic delta encoding showed strong adaptability, future work will evaluate its performance on other BASN platforms and applications and in combination with other on-node signal-processing techniques, such as feature detection and pattern classification algorithms.

Chapter 3 Reconfigurability at Components Voltage Level[3]

This chapter talks about the work done in adding voltage scaling at a fine-grained granularity to processors and what benefits/complications this configurability adds. We will go over the concept this project was based on and its main research goals. For ease of reference, the technique of adding fine-grained voltage-switching abilities to arithmetic components is referred to as Panoptic Dynamic Voltage Scaling (PDVS).

To improve energy efficiency, PDVS uses headers and several global V_{DD} rails instead of dedicated block-level DC-DC converters. This architecture speeds up virtual-V_{DD} switching, allowing PDVS to lower energy even for brief changes in workload, which cannot be realized in conventional DVS implementations. The delay of switching the virtual V_{DD} depends on the header size, but for our processor, this time is less than our target clock period.

3.1 Concept and Design Overview

The idea of reducing performance to save energy when applicable is a very widely used concept. Almost every circuit can operate at a lower voltage than what it is typically running on. This reduction of voltage will increase the delay and hurt the performance of that circuit, however; it will also decrease the energy and power consumed in that circuit. The most famous technique used to utilize this concept is Dynamic Voltage and Frequency Scaling (DVFS), where the voltage is reduced along with frequency, which saves a significant amount of energy at the expense of performance.

In this project, we try to explore the various tradeoffs of adding this fine granular voltage scaling and we try to maximize the efficiency by finding the right amount of configurability voltage scaling should be implemented at.

[3] Most text and data in this chapter are taken from [2][3][4][5]. The work in this chapter was done in collaboration with 3 other Ph.D. candidates. The focus in this dissertation is on the individual contribution I had in this project.

To increase the granularity of the voltage scaling, we added headers to arithmetic components. Each of those headers is linked to a different voltage rail, with a different voltage value. Having those multiple voltage rails enables each arithmetic component to independently switch its voltage to one of the discrete voltages available, as illustrated in Figure 3-1. This fine granularity voltage scaling will definitely increase the potential benefits of voltage scaling; however, it also adds a significant amount of complexities on various aspects of the design. In this work, we explore the range of the options that affect the overhead of the configurability and the efficiency of the system. During this work, we aimed to find the best point for maximizing the energy savings across several variables.

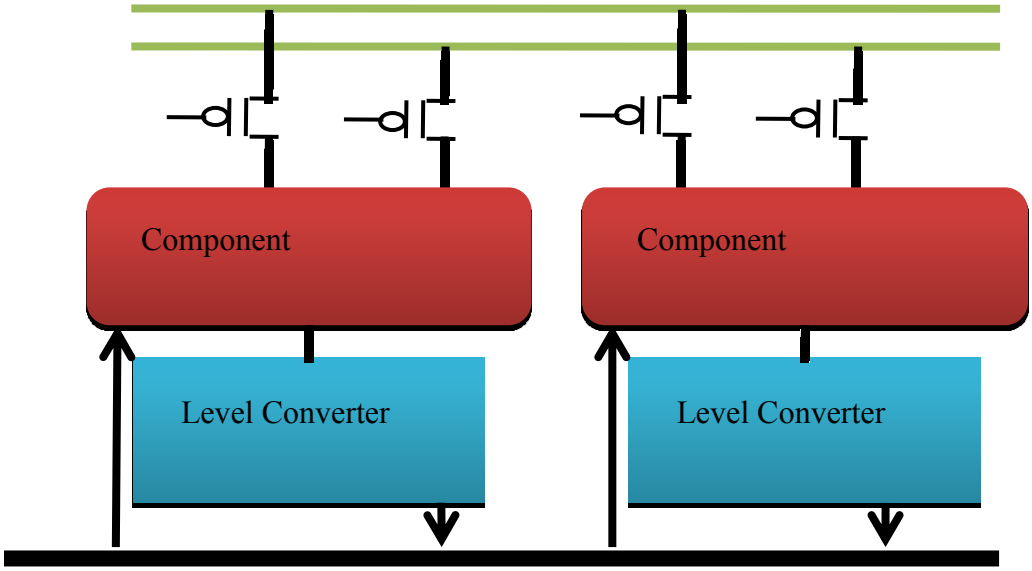


Figure 3-1: Block diagram of the concept of a header-based voltage-switching system

To explore the full benefits of PDVS, we designed a 32-bit data flow processor, illustrated in Figure 3-2, capable of executing arbitrary data flow graphs (DFGs) at 1 GHz at 1.2V. We used the PDVS architecture to implement the datapath of the processor. The datapath consists of four Baugh-Wooley multipliers and four Kogge-Stone adders. Each of these components uses three PMOS header switches tied to the three V_{DD}s (V_{DDH}, V_{DDM}, V_{DDL}) that are common throughout the processor. The processor includes a programmable crossbar that feeds input registers of the datapath components either from the datapath, the register bank, or the memory. To prevent short-circuit current from blocks operating below the nominal

V_{DD}, level converters (LCs) are used at the output of each multiplier and added to up-convert their outputs to the V_{DDH} level that is used at the register file.

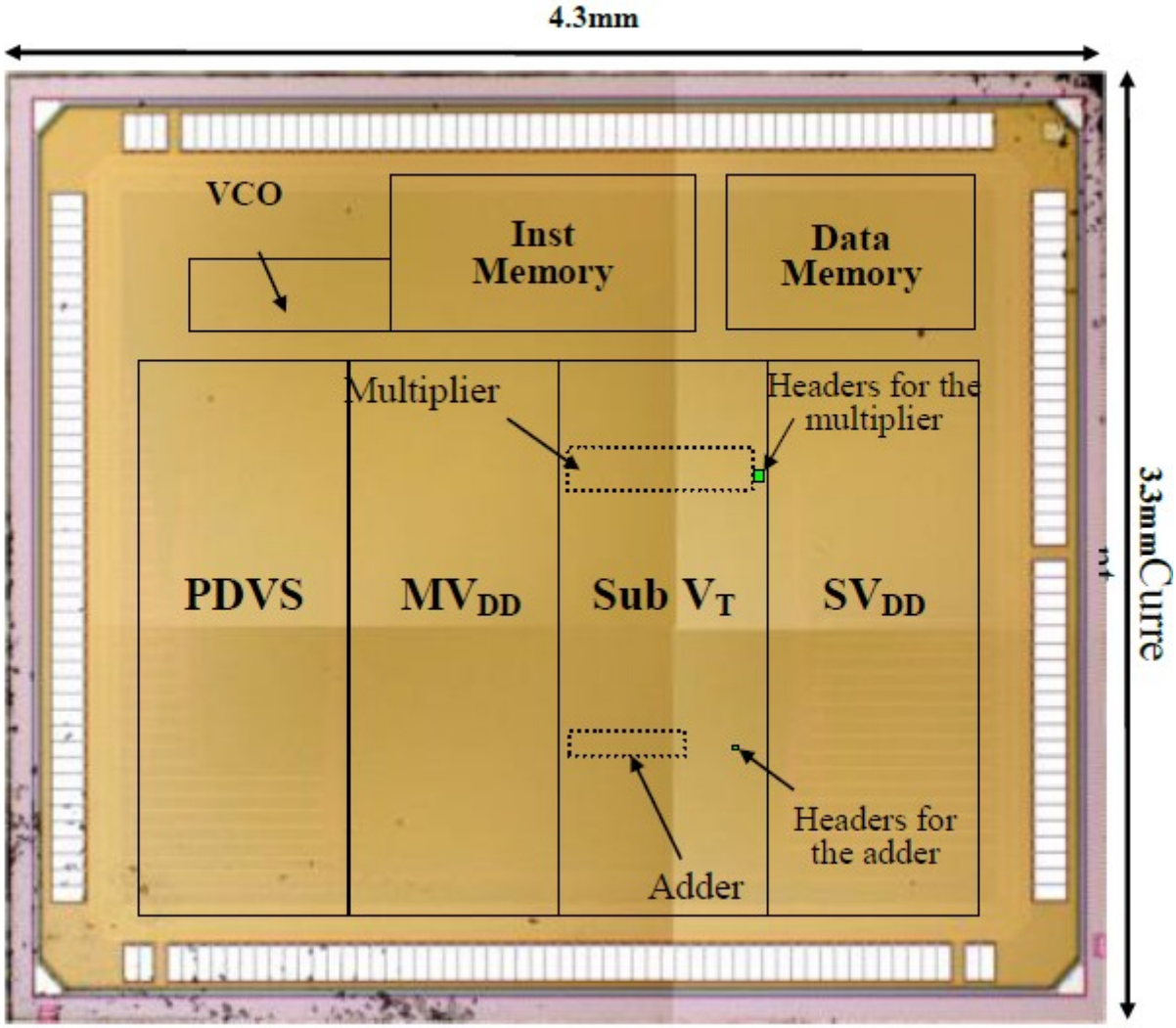


Figure 3-2: Die photo of the custom digital signal processor used for PDVS testing showing some of the main components

In order to provide a fair hardware comparison to PDVS, we included three additional datapaths on the chip that are functionally identical but that use different power management options: single V_{DD} (SV_{DD}), multi-V_{DD} (MV_{DD}), and a sub-V_T optimized PDVS datapath. In the SV_{DD} datapath, the four multipliers and adders all share the same V_{DD}. In the MV_{DD} datapath, the four multipliers and adders are permanently

tied to either V_{DDH}, V_{DDM}, or V_{DDL}, and operations can be scheduled for execution on any of these components based on the timing requirements. The processor has a 32kb data memory and a 40kb instruction memory that are shared for all of the datapaths. The control word for controlling the data flow (and header control where applicable) of the various datapaths is 160b for this test chip.

The PDVS processor, shown in Figure 3-3 contains a 40kb data memory and a 32kb instruction memory. Since the focus of the processor was to evaluate the benefits of the PDVS datapath compared to the other datapaths, the memories operate at a nominal voltage of 1.2V and use 6T SRAM. These SRAMs are designed to run at the maximum processor frequency of 1 GHz. The memory design is a main contribution done by a different researcher. The instruction memory is in the critical path of the processor; the 160b control word must be read every clock cycle. However, the data memory is not on the critical path since it is only read and written at the start or the end of a DFG iteration.

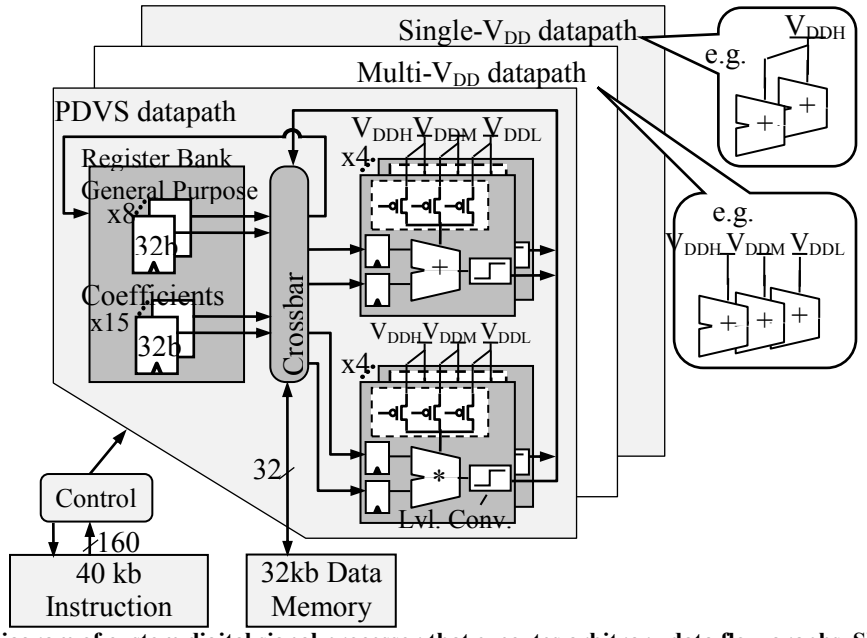


Figure 3-3: Block diagram of custom digital signal processor that executes arbitrary data flow graphs. SRAMs and control serve three similar copies for direct comparison of PDVS, SVDD and MVDD. [4]

We were able to acquire a large amount of data from this chip. My individual work was focused on the system-level data and the tools performance. In the following sections, we will present some of the main data collected in this project.

While the premise of voltage scaling is sacrificing performance for energy, sometimes there is an extra slack for some of the components created due to dependency on other components. This has been utilized in many aspects, like when there is a cache miss, processors will typically try to do something else rather than just wait for the cache to come back with the correct value. Similar thing with voltage scaling; in some cases, some arithmetic components are waiting on other components to finish. In this case, the component can lower its energy without affecting the overall performance.

Of course the energy saved from those small slacks is limited; however, this was not viable with the traditional voltage scaling, which required several components to reduce performance at the same time. Error! Reference source not found. shows the observed energy savings in our system in comparison to traditional methods. More detailed explanation of the results will be presented in the following sections.

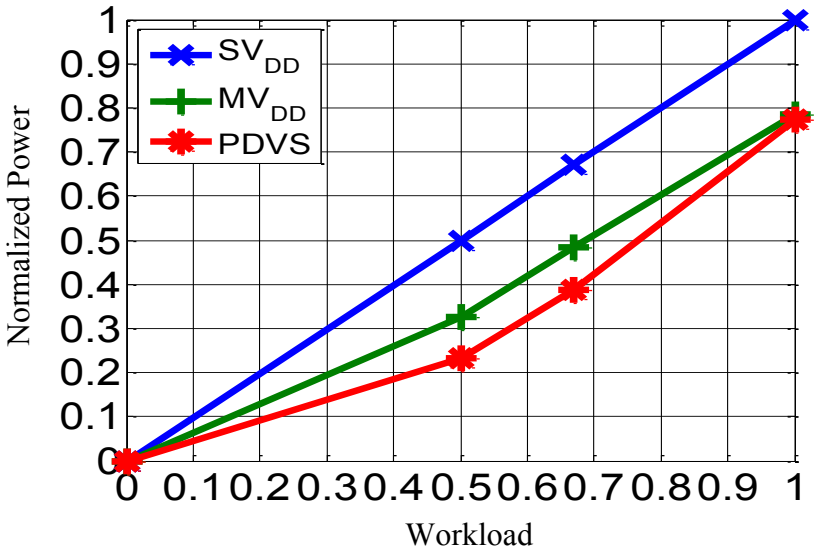


Figure 3-4: Average measured power of seven benchmarks running on PDVS and two other traditional voltage scaling methods

Designing for configurability introduced several variables that need to be considered. In our case, the overhead of switching is directly influenced by the headers and the additional scheduling task of assigning which rail each operation should execute on. Another aspect that affects the energy efficiency of the system indirectly is the voltages of the rail. Since the number of voltage rails is limited, the voltages of those rails determine the energy-delay profile the arithmetic can operate on, which affects the efficiency of the scheduler at different rates. Those aspects will be discussed in the following sections.

3.2 Approach and Tool Creation

Energy-savings benefits can become apparent once the benchmark is scheduled to use the lower-voltage rails. The scheduler has to be redesigned and adjusted to include this voltage-selection ability. In this work, we created the algorithm and then a program that is capable of scheduling the operations to arithmetic components and lowering the voltages whenever possible.

To test the energy savings in PDVS, we created a series of tools that can convert a simple data flow graph (DFG) into microcode, illustrated in Figure 3-5, which can be downloaded to our processor and executed on the three different schemes (a fourth scheme using subthreshold existed; however, it is a mainly individual contribution of another student). The series of tools to achieve this task is used extensively during this project to explore and analyze the effects on the system level. The tools are responsible for scheduling the application according to a user-specific time constraint and for optimizing the schedule to reduce the energy consumption without violating the timing constraints. The microcode output of those tools contains all the scheduling, switching, and binding information required to run the PDVS system efficiently**.** Figure 3-5 below describes the stages that the DFG has to go through before it is executed on the chip.

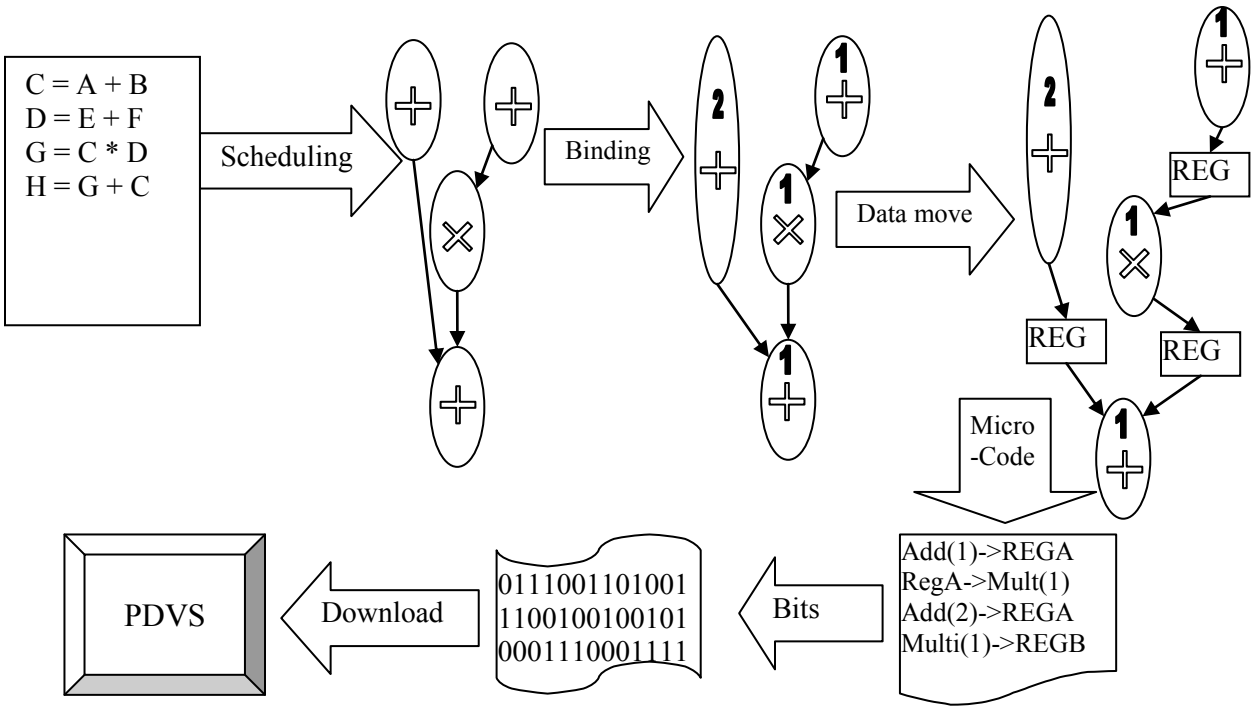


Figure 3-5: Flow diagram for tools chain for PDVS showing the different stages the application has to go through before it can be executed by the processor

To be able to find the right schedule, we had to obtain the energy-delay graph for the adder and the multiplier units, showing our results of this graph in Figure 3-6. Using this data as an input to the scheduling tools, we were able to compare the benefits of using various methods in choosing the voltage of the rails and the effects of using different voltages for the rails.

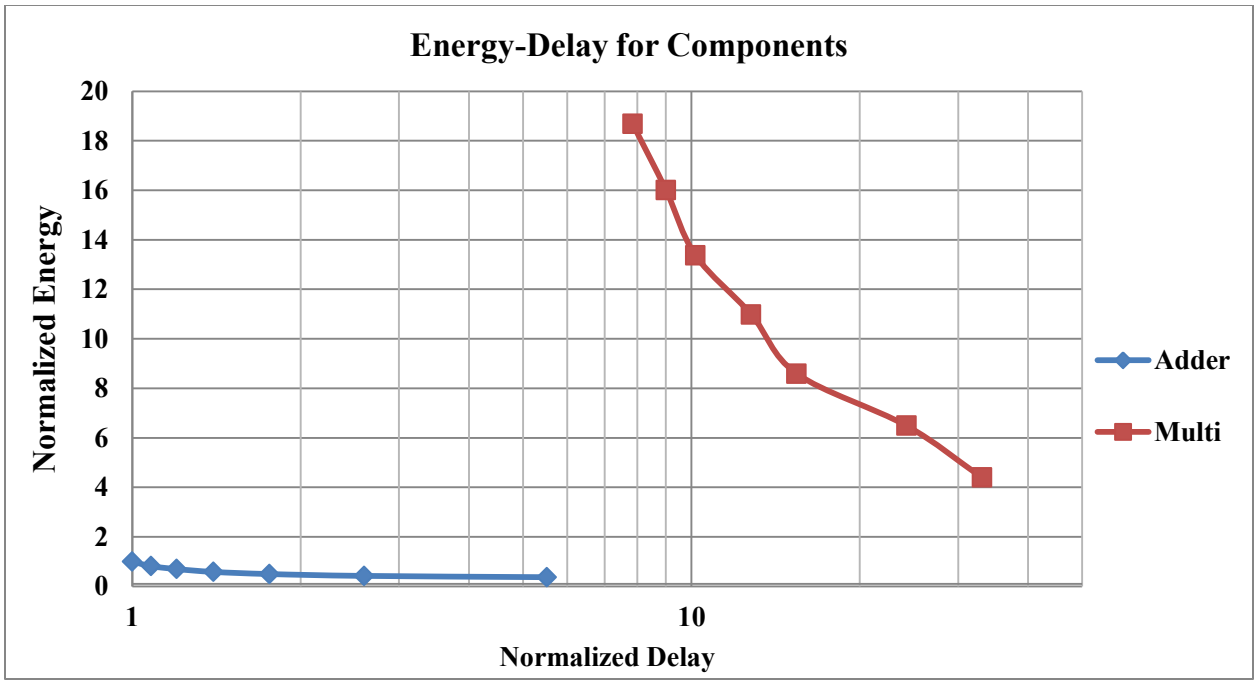


Figure 3-6: Energy-delay graph for arithmetic components used in PDVS normalized to adder running at 1.2 volts

3.3 Scheduling with Circuit-Level Information

Scheduling and binding is the process by which application operations are assigned to a clock cycle (C-step) and a specific resource (adder/multiplier) for execution. Typically, the inputs to this process, shown in Figure 3-7 (left), are the various operations (represented by nodes), the data dependencies between those operations (represented by vertices), the time constraint (typically given in C-steps), the available computational resources (adders, multipliers, registers, etc.), and, unique to multirail systems, the number and voltages of the rails. The output of the algorithm is the time-specific assignment of those operations to the specific resource. The schedulers in DVS systems that are capable of operation-level voltage selection have to determine, additionally, the voltage of each operation without violating the timing constraints [55].

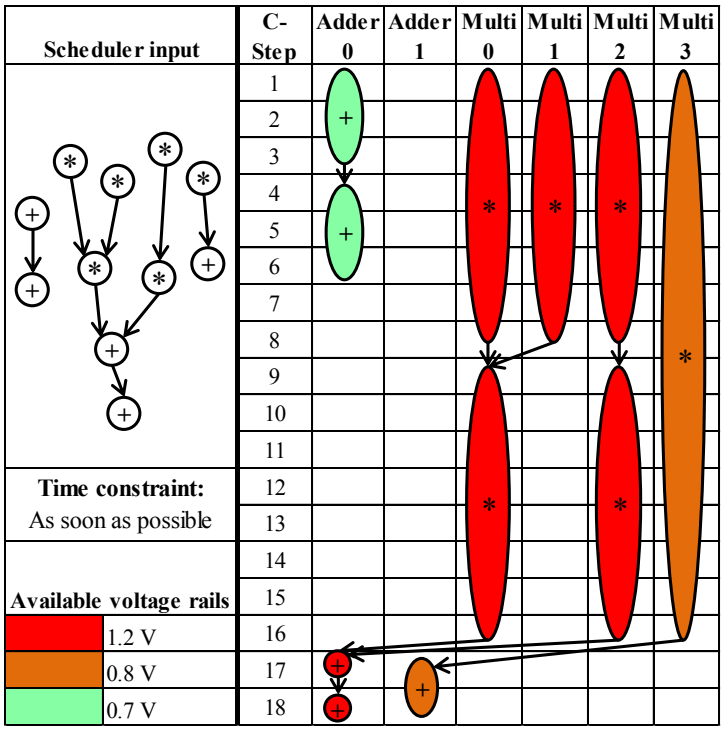


Figure 3-7: Scheduling DiffEq showing lower voltages assigned to operations with timing slack.

However, unlike non-header-based systems, components in header-based systems can switch voltage independently, thus requiring modifications to preexisting multirail scheduling algorithms. Figure 3-7 (right) shows an example of scheduling the DiffEq benchmark on a header-based multi–voltage rail system. The schedule is capable of reaching the same maximum performance as a non-DVS system while reducing system energy.

As an input for the scheduler, we used the simulated energy and delay information about the Kogge-Stone adder and the Baugh-Wooley multiplier shown in Table 3-1**.** Those numbers were used to schedule the benchmarks, trying to reduce their total energy as much as possible without violating resource or timing constraints set by the system.

Table 3-1: Simulated energy and delay of the components including level converters overhead. Results show the large energy savings and the extra delay at lower voltages

Voltage (V)	1.2	1.1	1.0	0.9	0.8	0.7	0.6
Adder Delay (C-steps)	1	1.08	1.2	1.4	1.76	2.6	5.51
Multiplier Delay (C-steps)	7.85	9.01	10.17	12.79	15.42	24.25	33.11
Adder Energy (pJ)	2.95	2.4	2.05	1.71	1.46	1.24	1.09
Multiplier Energy (pJ)	55.07	47.17	39.41	32.37	25.32	19.12	12.94

3.3.1 Scheduling Background

Obtaining the optimal schedule even in non-header-based systems is an NP-complete problem. Therefore, researchers and designers have opted to use heuristics algorithms instead [57][58]. One of the more successful and simple of these algorithms is the list-based algorithm [56]. We have created a list-based algorithm to accommodate our system's independent voltage-switching feature.

The scheduling algorithm we implemented is designed to minimize energy consumption while satisfying time and resource constraints. The scheduler minimizes energy through assigning some operations to lower voltages. The criterion to choose which operations to assign to a lower voltage determines the effectiveness of the scheduler. We studied five different selection criteria; these criteria differ in the priority given to assigning operations to lower voltages.

Since we use a list-based algorithm, the complexity of scheduling for this system is similar to a single voltage rail system. The scheduling occurs before the application execution, which is typical in most systems, and therefore does not incur additional delay overheads during the application.

3.3.2 The Algorithm

We describe in this section the algorithm we used to schedule and bind the components in our PDVS system. Below are the terms we used in the algorithm, and then we present the algorithm itself.

Algorithm Terms

- ***Nodes***: The operations in the application. In our applications, operations are addition/subtraction or multiplication. However, the algorithms presented can be applied to additional types of operations.
- ***C-step***: This represents the system's clock cycle. The timing constraint is supplied to the scheduler as a maximum number of C-steps to complete all the operations in the application.
- ***Relaxing a node***: This means decreasing the voltage of an operation (node) to decrease its energy consumption and increase its execution time.
- ***ASAP schedule***: "As soon as possible" is a schedule where all operations are assigned to the earliest possible start time.
- ***ALAP schedule***: "As late as possible" is a schedule where all operations are assigned to the latest possible start time.
- ***Slack***: The maximum number of C-steps a node's start time can be delayed without violating the timing constraint of the application.

The Scheduling Algorithm

We present the pseudo code of the scheduling program used to generate the subsequent data. The algorithm itself is not a significant contribution of this chapter, but it is presented for clarity.

1. Assign highest voltage to all components
2. Make an ASAP schedule that takes resource constraints into account, prioritizing nodes that reside on longer paths
3. Create an ALAP schedule, taking resource constraints into account
4. Assign the differences between the start times of the operations in the schedules from step 3 and step 2 as the slack of each node
5. While there is a node that can be relaxed:
    a. Lower the voltage of a node or a group of nodes (chosen by a specific selection criterion)

b. Adjust the schedule by moving the start times of the necessary nodes to maintain resource constraints and correct data dependencies

c. Recalculate the slack time for all nodes

We see in that algorithm that we reduce the voltages of some nodes. The choice of which node is relaxed first affects the efficiency of the algorithm. In the next section, we will discuss several options for selecting the nodes' order.

3.3.3 Node Relaxation Criteria

We present the various criteria that we use to determine which nodes to relax first (decrease their assigned voltage). After the relaxation of a node, the scheduler will go through an iterative function to move other nodes around to ensure data dependency and resource constraints are not violated due to relaxation. The selection criteria are as follows:

- ***Random***: Randomly choosing a node to relax
- ***Slack***: Prioritizing to relax first the node with the largest slack
- ***Inverse slack***: Prioritizing to relax first the node with the least slack
- ***Energy per slack***: Prioritizing to relax first the node that saves the most energy relative to the total slack lost by all other nodes
- ***Max energy step***: Prioritizing all the possible nodes in the C-step that can achieve the highest energy savings relative to other C-steps (typically means that most nodes in the selected C-step can be relaxed)

Table 3-2: Different node selection criteria impact on energy consumption across different rates. The complexity of all the criteria is the same (O(N^2*V)).

Benchmark	All Filter	AR Lattice	DiffEq	Ellip	FFT	FIR8	GCD	Kalman	Average
Selection Criteria	Rate range 1x to 2x								
MaxEnergyStep	0.688	0.696	0.662	0.663	0.634	0.711	0.747	0.665	0.683
EnergyPerSlack	0.699	0.755	0.667	0.665	0.622	0.771	0.747	0.680	0.701
InverseSlack	0.759	0.773	0.662	0.721	0.630	0.772	0.798	0.688	0.725
Random	0.845	0.839	0.785	0.842	0.717	0.841	0.899	0.767	0.817
Slack	0.702	0.779	0.693	0.690	0.701	0.804	0.798	0.711	0.735
	Rate range 1x to 3x								
MaxEnergyStep	0.588	0.592	0.573	0.575	0.558	0.601	0.635	0.575	0.587
EnergyPerSlack	0.594	0.627	0.576	0.576	0.552	0.641	0.635	0.584	0.598
InverseSlack	0.627	0.636	0.573	0.607	0.556	0.648	0.663	0.588	0.612
Random	0.721	0.732	0.673	0.717	0.643	0.741	0.818	0.663	0.713
Slack	0.596	0.640	0.590	0.590	0.598	0.656	0.663	0.601	0.617
	Rate range 1x to 4x								
MaxEnergyStep	0.565	0.568	0.552	0.555	0.541	0.576	0.610	0.555	0.565
EnergyPerSlack	0.570	0.597	0.555	0.556	0.536	0.608	0.610	0.562	0.574
InverseSlack	0.598	0.604	0.552	0.581	0.539	0.614	0.633	0.565	0.586
Random	0.674	0.683	0.634	0.671	0.610	0.696	0.759	0.626	0.669
Slack	0.572	0.607	0.567	0.567	0.573	0.621	0.633	0.576	0.589
	Rate range 1x to 5x								
MaxEnergyStep	0.529	0.530	0.520	0.523	0.513	0.536	0.569	0.522	0.530
EnergyPerSlack	0.532	0.549	0.522	0.524	0.510	0.557	0.569	0.527	0.536
InverseSlack	0.550	0.554	0.520	0.540	0.512	0.561	0.584	0.529	0.544
Random	0.599	0.605	0.573	0.599	0.558	0.617	0.666	0.569	0.598
Slack	0.533	0.556	0.530	0.531	0.535	0.565	0.584	0.536	0.546

Analyzing the selection criteria shows that all of them have the complexity of $O(N^2 * V)$, where N is the number of nodes in the schedule and V is the number of voltage rails. The algorithm loops N * V times to iterate through each node and each voltage level, and when extending any node, there is a chance that N-1 nodes will need to be moved around to accommodate constraints. The complexity is similar among the different criteria; though some of them are more complex, they relax more nodes in each loop, thus achieving the same overall complexity.

Table 3-2 shows the comparison between the different relaxation selection criteria for seven DSP benchmarks running on the header-based multi–voltage rail system, using 1.2V, 0.8V, and 0.7V for the three rails. The minimum values are highlighted. We see that all of the selection criteria provide significant energy savings (~40% over single–voltage rail systems) and the differences between their

effectiveness are limited. However, we see that the MaxEnergyStep criterion provided the best results almost in all benchmarks at all rates. This indicates that scheduling approaches that take multiple nodes into consideration is preferable, since they enhance results without incurring additional complexity.

3.3.4 Changing Operating Rate

Header-based multiple–voltage rail systems are designed for applications that require variable operating rates and transition frequently between those rates. To exploit extra slack in the time constraint, the DFG needs to be changed to represent a different operating rate. This enables the system to save extra energy. In this section, we will discuss three main methods to achieve that: simple extension, partial extension, and multiple DFGs.

Simple Extension with or without Dithering

The simplest scheme to extend a DFG is lowering the voltage of each component by one voltage rail. If an operation was scheduled to run at the highest voltage, simple extension will schedule the operation to run on the medium voltage. If it was assigned to run at the medium voltage, it will run at the lowest voltage. Operations that were already scheduled to run at the lowest voltage would simply remain at that voltage.

The simple extension scheme is easy to implement. We add extra status bits, indicating the operating rate. With the addition of simple combinational logic, we can sufficiently implement this scheme in most systems. Figure 3-8 shows the changes to the DFG after the simple extension compared to its original form, shown in Figure 3-8a. Note: this is analogous to DVS in non-header-based architectures, as all components connected to a particular rail have their voltages adjusted by the same amount.

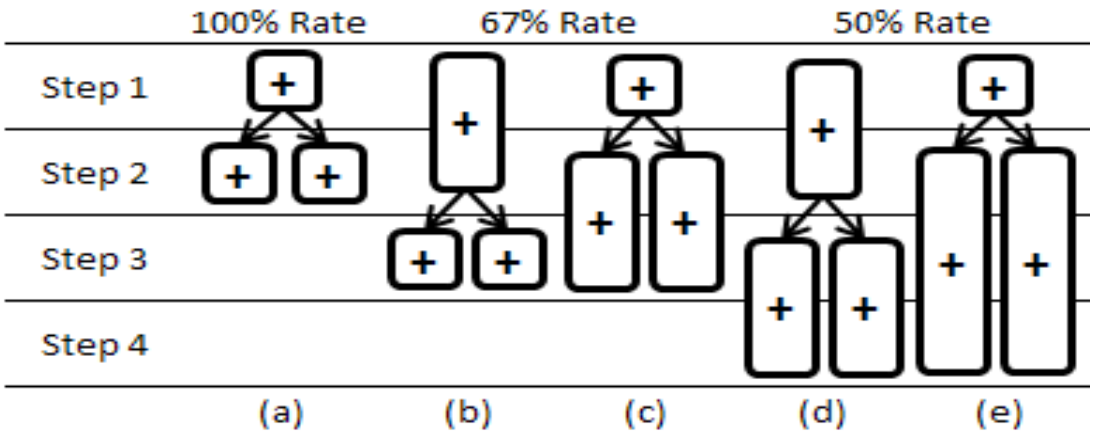


Figure 3-8: Example DFG and scheduling strategies to extend DFGs for different processing rates

Since this scheme can switch every operation or not switch at all, it requires extra slack by a factor equal to the highest component delay factor to ensure functionality. Heterogeneous components, such as multipliers and adders, do not have similar energy-delay trends. For example, Figure 3-9 shows the energy delay for a Kogge-Stone adder and a Baugh-Wooley multiplier, with clear differences in their energy-delay relationships. The energy-delay differences will reduce the efficiency of this scheme because we will not be able to switch the DFG to a lower rate until we have extra latency equal to the highest component delay factor. For example, if one component's delay factor is 2x and the other component is 2.5x, then we cannot extend the DFG unless we are operating, at least, at 2.5x the latency. Dithering can be used to save energy while running at intermediate rates by spending certain percentages of the execution at the available discrete rates. This dithering is made efficient by the header-based architecture and is less practical for traditional DVS systems.

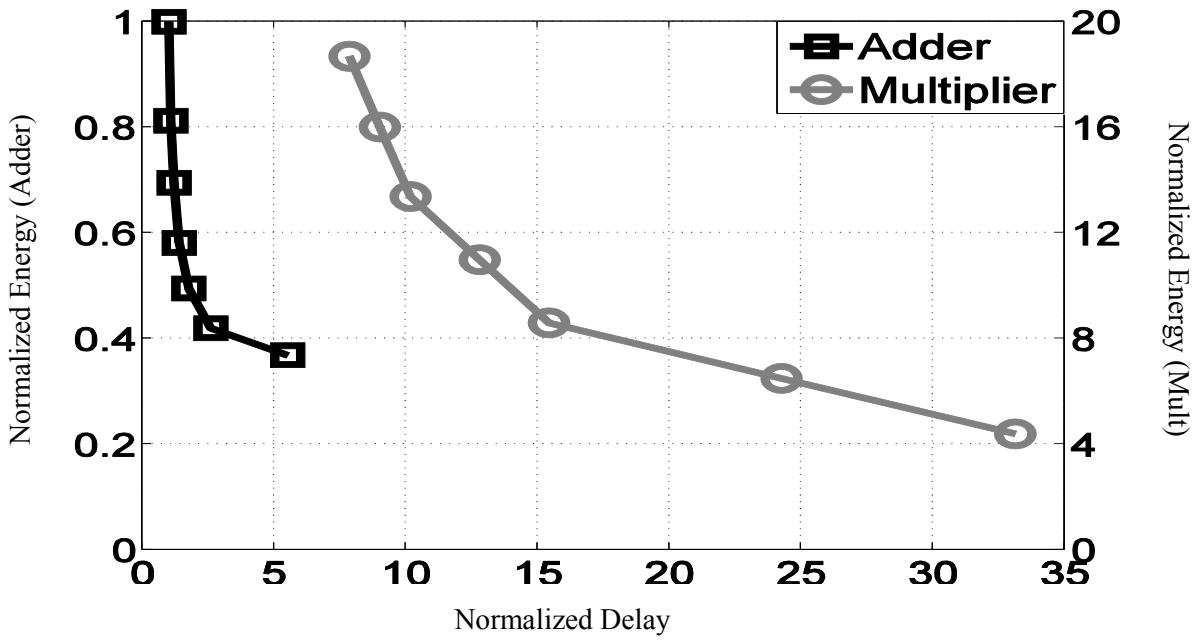


Figure 3-9: Normalized energy and delay of the arithmetic components showing the general trend when varying voltage

For our system, the biggest ratio of component delays is 2x and 3.125x. Therefore, the discrete latencies we can achieve are 1x, 2x, and 3.125x without dithering. **Error! Reference source not found.** shows the energy of applying the simple extension scheme with and without dithering. Other two schemes, Partial Extension and Multiple DFGs, are illustrated in the figure also. Those two schemes will be introduced below. Note that energy results at 1x, 2x, and 3.125x rates are the same with or without dithering, because dithering is not needed when operating at one of the available discrete rates (the rates are determined by the voltages of the rails). In other rates, the energy benefit of dithering between two rates is clear and significant.

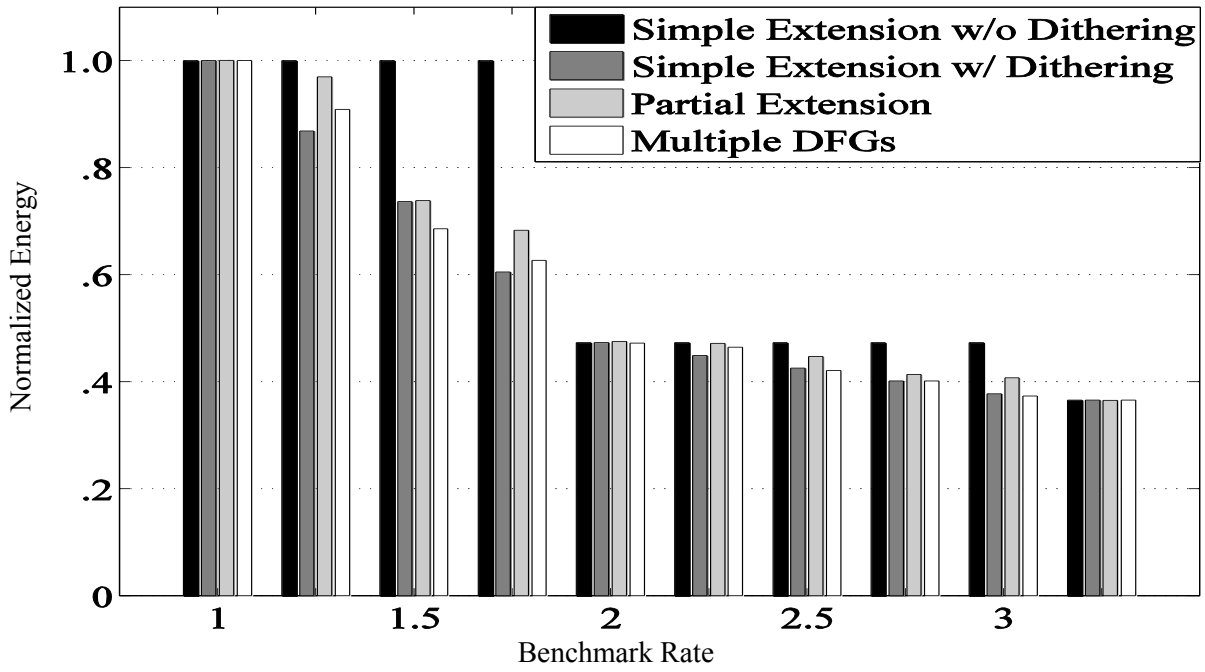


Figure 3-10: Average energy of benchmarks when using the different schemes for changing rates

Partial Extension

In the previous scheme, we discussed simple extension's limitation in achieving intermediate rates without dithering. We can introduce a modification on the simple extension scheme to enable it to achieve very fine granular rates. In partial extension, we extend only part of the DFG while running the rest of the operations at the original rate. For example, if the DFG contains 20 C-steps, then extending only the first 10 C-steps will produce a 1.5x delay DFG. By choosing the percentage of the DFG we are extending, we can have very fine granular operating rates.

This extra flexibility comes at the cost of efficiency since blind partial extension is not efficient in specific situations. Consider for example Figure 3-8b-c, if the algorithm blindly extends the first part of the DFG (Figure 3-8b), the solution will be suboptimal (Figure 3-8c), which is clearly non-optimal. Therefore the algorithm must select its extensions strategically.

Implementing this scheme is more complex than the previous one, the simple extension scheme, but still not difficult. The scheme is implemented by adding a simple tracker of the current C-step to the previous scheme. By tracking the C-step and depending on the desired rate, a flag indicating extension can be toggled.

Since this method extends the DFG conservatively to ensure functionality it cannot use dithering. Dithering can be used only if there is extra slack after finishing the whole DFG. In this scheme, unlike the previous one, we extend the DFG to consume as much as we can from the extra slack. We can see in **Error! Reference source not found.** that this technique is useful only if dithering is not available or too costly to implement.

Store Multiple DFGs in Cache

In the two previous schemes, the extension of the DFG is done without rate-specific optimization, thereby losing opportunities for further energy savings. Therefore, the third scheme stores multiple copies of the same DFG, optimized for different rates.

Additionally, the operating rates can be chosen independently from the latency factor of the components, unlike the first scheme. This can be particularly useful if the profile of operating rates is known *a priori*. Knowing the operating rates of the application enables the production of the corresponding rates, achieving maximum efficiency, especially when combined with voltage rail selection (see below).

The overhead of this scheme can vary depending on several aspects of the system, such as memory access time and energy and cache capacity. If the instruction cache is much larger than the DFG, then the overhead is pretty minimal since we can easily store multiple copies of the DFG inside the cache. If the instruction cache can fit only one copy of the DFG, the scheme will incur memory reading and writing overheads for switching rates. For example, looking at memory reading and writing overheads using CACTI, each “instruction word” read from a 256kb cache, which is reasonable for an L2 cache, needs 21pJ and writing each instruction word into a 64kb instruction cache needs 13pJ. These numbers assume

a 1 GHz clock speed. We use the DiffEq DFG to illustrate this example. The DFG, which contains around 16 instruction words, will need 32ns and 544pJ to switch to a different rate. The DiffEq DFG at 1x latency consumes 312pJ. By extending the latency to 1.5x, the DFG consumes 218pJ. This introduces the concept of a breakeven time to overcome the energy overhead of rewriting the cache. In this example, the DFG needs to run at least six consecutive times at the 1.5x latency to outweigh the energy overhead consumed by switching from and to the 1x latency. Running six consecutive times is very reasonable practical scenario since these DFGs are used for applications processing thousands of data, like sound and video applications.

The example assumed the worst-case scenario, where without the extra copies, the 1x rate DFG will fit inside the cache perfectly. Most DFGs (and the DFGs of their various rates) have to be stored in multiple cache levels due to their size. Therefore, the overhead of reading and writing from higher memory levels might make having longer DFGs not as energy efficient.

Another issue with this scheme is that although it achieves maximum efficiency, it will only provide operating rates corresponding to the stored DFGs. However, we can use dithering between few copies of the DFGs while operating at intermediate rates. This will provide better efficiency than simple extension even with the same number of points. This is because we can choose the rates of those DFGs unlike simple extension, where they are bound to the corresponding latency factor of the voltage rails. However, the overhead of switching DFGs must be considered when determining the dithering duty cycle.

Error! Reference source not found. shows the energy of several DFGs obtained using the mentioned schemes. Multiple DFGs scheme does not always provide benefits over simple extension with dithering. This is the case because if the DFG is small and the rate of the DFG is close to 1x, the available slack for extending within the DFG is small. This may make it impossible for the scheduler to exploit this small slack, especially if the delay factor between voltage rails is high. On the other hand, dithering, by nature, accumulates those small slacks to enable exploitation by operating several iterations at a lower rate. This

effect will be small if the DFG is large because the dithering savings is linear, while savings from within a DFG is quadratic. Therefore if the DFG is big enough, even small reductions in operating rates will produce enough slack to be exploited by the scheduler. There will be enough quadratic savings within the DFG, even if the slack was not exploited fully, it will triumph over the linear savings of dithering.

Each of the three mentioned schemes has its own benefits and drawbacks. Summary of those aspects are shown in Table 3-3. Characteristics of the system and the nature of the applications will determine which scheme is better.

Table 3-3: General comparison between the extenstion schemes, highlighting the main differences

	Fully Extended DFG	Partially Extended DFG	Multiple Copies of the DFG
Possible Rates	Limited to the number of rails	Very fine granularity	Limited to the number of stored DFGs
DFG's Efficiency	Within 5% of optimal using dithering	10%–20% of optimal	As optimal as the scheduler
Overhead	Very simple glue logic	Slightly complicated glue logic	Storing and retrieving multiple DFGs (system dependent)

3.4 Level of Reconfigurability

In this section, we will explore the trade-offs of adding more voltage rails as well as the potential selection methods for selecting the voltages of the rails.

3.4.1 Number of Rails

Through most of the literature and our results in the previous sections, three voltage rails are used [63][64][65][66]. In this section, we explore the trade-offs of the number of rails through qualitative analysis of the overheads and estimation of the energy savings provided by additional rails.

Less Metal per Rail Network

As technology scales, the same designated routing area for the power distribution network is required to support more components, causing power delivery and voltage drop problems [59][61]. Having more rails requires dividing this limited area into even smaller power networks, reducing the amount of metal in each rail, exacerbating these problems as follows.

In modern devices, a high surge in the current drawn from the power network is common when complex blocks power on. This in turn causes the voltage on the power network to drop momentarily. This drop can cause many faults and problems across the system, especially in near-threshold circuits. A reduction in the amount of metal per power network results in smaller rail capacitance and higher rail resistance, thus making the voltage drop even more severe. That is why commercial products try to limit the number of networks as much as possible.

DC-DC Voltage Converters

DC-DC converters are used to provide different voltages on chip. They are also used to change the voltage rail level dynamically in traditional DVS systems. In multi–voltage rail systems (header-based or non-header-based), each rail requires a separate DC-DC converter, while in traditional DVS systems, each voltage-independent section needs its own DC-DC converter. Typically there are more voltage-independent sections in DVS systems than number of rails in multi–voltage rail systems. This causes the complications of the DC-DC converters to be more severe in traditional DVS systems.

The main disadvantage to increasing the number of DC-DC converters is the increase in area and their energy inefficiency (e.g., peak efficiency of 77%) [62]. Furthermore, this peak energy efficiency is only achieved when the DC-DC converter is optimized for the given output load and voltage[60]. In header-based systems, the output voltage is constant but the output load varies, thus reducing the energy efficiency of the converters.

Increasing the number of voltage rails does not necessarily reduce the overall energy efficiency of the DC-DC converters. However, it lowers the average output load and changes the distribution of loads per rail. This significantly complicates the process of optimizing the DC-DC converters and renders achieving high energy efficiency very difficult. Traditional DVS systems suffer from this same problem because the output voltage varies.

Designing and optimizing the DC-DC converters for variable load and variable output voltage requires considering many other system variables that are outside of this chapter scope [62]. It is hard to predict whether adding voltage rails will decrease energy efficiency overall; however, it will definitely complicate the power system design and increase the area overhead.

Scheduling Results

The main benefit of increasing the number of voltage rails is the better utilization of timing slack available in the applications, thus allowing us to achieve a more optimal schedule. To investigate the potential energy savings from having more voltage rails, we used the "Best Step" selection criterion to schedule the benchmarks over various numbers of voltage rails. One voltage rail is kept at 1.2V (the maximum voltage) to ensure the ability to operate at the 1x rate.

To provide a fair comparison, we used an exhaustive search to choose the optimal voltages for each number of rails. The selection of optimal voltages was done over the average normalized energy of all rates of all the benchmarks. The energy overhead of adding voltage rails, previously discussed, is not included in the results because our system does not include the energy efficiency of the power system on chip.

Figure 3-11 presents the benchmarks' energy savings of increasing the number of voltage rails. These results reveal that there is little additional energy savings beyond two voltage rails (within 5%, 7%, and 11% for three, four, and five rails, respectively). Given all of the complexity and overhead of adding voltage rails to the system, two voltage rails seems the best option for the header-based system and benchmarks considered here.

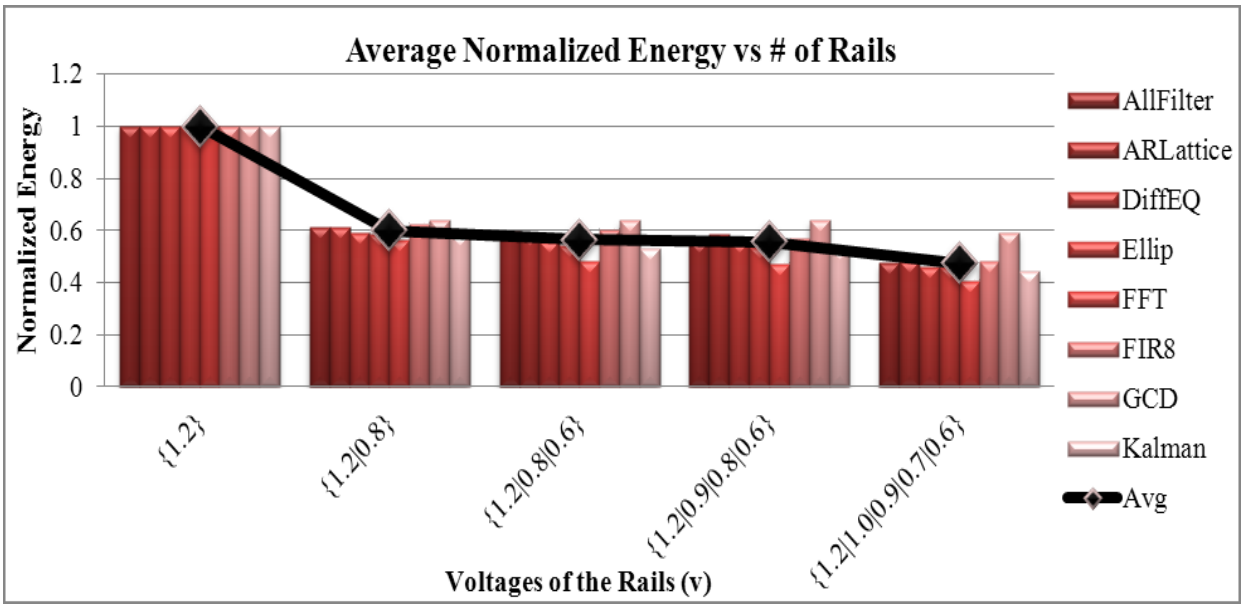


Figure 3-11: Energy savings provided by increasing the number of voltage rails. Adding more than two voltage rails have a greatly diminishing return (only 4% extra energy savings when adding the third voltage rail).

3.4.2 Voltage Selection

Voltage selection has a significant impact on the energy savings due to the limited number of voltage rails. Smaller voltage differences between the rails result in utilization of smaller timing slacks and more frequent switches. On the other hand, larger voltage differences can save more energy but only if the timing slack is large enough.

One of the simplest ways to choose the rail voltages is to select voltages that produce 1x, 2x, and 3x the delay of the components at the maximum voltage (in our case 1.2V). We set our C-step period typically to the delay of the fastest component at the highest voltage, including all the extra delays (e.g., level converters, control and data signals delay). Having an integer multiple of C-steps for component delay maximizes C-step utilization. However, achieving this integer multiple proves extremely difficult when dealing with multiple types of components. Since all component types use the same voltage rails, a voltage that results in one component type to run at an integer multiple of C-step will not necessarily result in an integer multiple C-step in a different component type as seen previously in Table 1.

Even if the delay trend is identical between different components, the absolute energy and delay can differ significantly; for example, in our system, a multiplication is around 8x of an addition's delay but 20x of its energy. Therefore, reducing the energy of a multiplication by 10% saves more energy than reducing the energy of an addition by 50%.

Additionally, having a reduced utilization of a one C-step (typically one CPU cycle) reduces the energy efficiency insignificantly in the big picture. Having the aforementioned complications and the insignificance of 100% C-step utilization made us conclude that aiming for the integer delay multiple when selecting voltages is not worth the effort. However, we are still motivated to analyze the impact of voltage selection criterion on the benchmark energy.

Table 4 shows the average energy of the benchmarks of every possible voltage combination normalized to single rail system, assuming three voltage rails and 100mV voltage granularity. The highlighted cells represent the minima for that range. We starred four noteworthy intuitive voltage selection criteria:

***Multiplication energy**: Choosing the voltages to achieve 3x, 2x, and 1x of the multiplication energy

****Addition energy**: Choosing the voltages to achieve 3x, 2x, and 1x of the addition energy

*****Equal differences**: Choosing equal voltage differences between the rails

******Component delay**: Choosing the voltages to achieve 1x, 2x, and 3x of addition/multiplication delay

The results show that intuitive voltage selection techniques, like according to component delay, produces near-optimal voltage selection results (within 0.03).

Table 3-4: Voltage selection impact on energy consumption across a range of rates. Most selection criteria provide significant improvements over single V_{DD}

Rail Voltages (V)	Range of Rates			
	1x–2x	1x–3x	1x–4x	1x–5x
1.2|1.1|1.0	0.803	0.757	0.742	0.735
1.2|1.1|0.9	0.750	0.668	0.641	0.627
1.2|1.1|0.8	0.720	0.593	0.551	0.530
1.2|1.1|0.7	0.794	0.643	0.551	0.504
1.2|1.1|0.6	0.832	0.769	0.611	0.532
1.2|1.0|0.9	0.743	0.665	0.639	0.626
1.2|1.0|0.8	0.714	0.591	0.549	0.529
1.2|1.0|0.7*	0.759	0.611	0.529	0.487
1.2|1.0|0.6**	0.769	0.641	0.558	0.491
1.2|0.9|0.8	0.715	0.591	0.549	0.529
1.2|0.9|0.7	0.736	0.588	0.514	0.476
1.2|0.9|0.6***	0.742	0.603	0.525	0.463
1.2|0.8|0.7****	0.725	0.566	0.499	0.464
1.2|0.8|0.6	0.722	0.565	0.487	0.433
1.2|0.7|0.6	0.834	0.662	0.549	0.479

3.5 Demonstration of PDVS

As a demonstration of the benefits of PDVS, we implemented a simple video processing application that brightens dimly lit areas of a frame. The workload of this application varies according to the number of dark pixels of each frame. The number of dark pixels that need to be brightened can easily be calculated from each image. With this information, we can compute the workload needed to achieve the required application rate, e.g., 24 frames per second (FPS) or 30 FPS. For the sake of simplicity, we created a program that brightens pixels by multiplying the pixels below a threshold value by a specific constant. The multiplication can be done either at V_{DDH} or V_{DDL} (i.e. fast or slow). With the knowledge of the total number of pixels that need brightening before processing the frame, we calculate the number of multiplications that will be scheduled at V_{DDH} and the multiplications scheduled at V_{DDL} to meet the exact deadline.

Since our test chip was a custom processor designed to demonstrate the benefits of PDVS, it does not contain any operating system that enables video data input, so we feed each frame as an image to the input data. We can see in Figure 3-12 the demo images before and after the processing that was executed on our test chip, which was the simple brightness application. We acknowledge that we actually distort the picture, but this application is for demonstration and it mimics, in behavior, practical image enhancing applications. If the processor knows the number of pixels that need processing beforehand, it can reduce its work rate to finish right on deadline thus saving energy when the workload is low. Figure 3-13 shows the significant (~60%) energy savings obtained from running the brightness application on a PDVS processor relative to single voltage rail system.

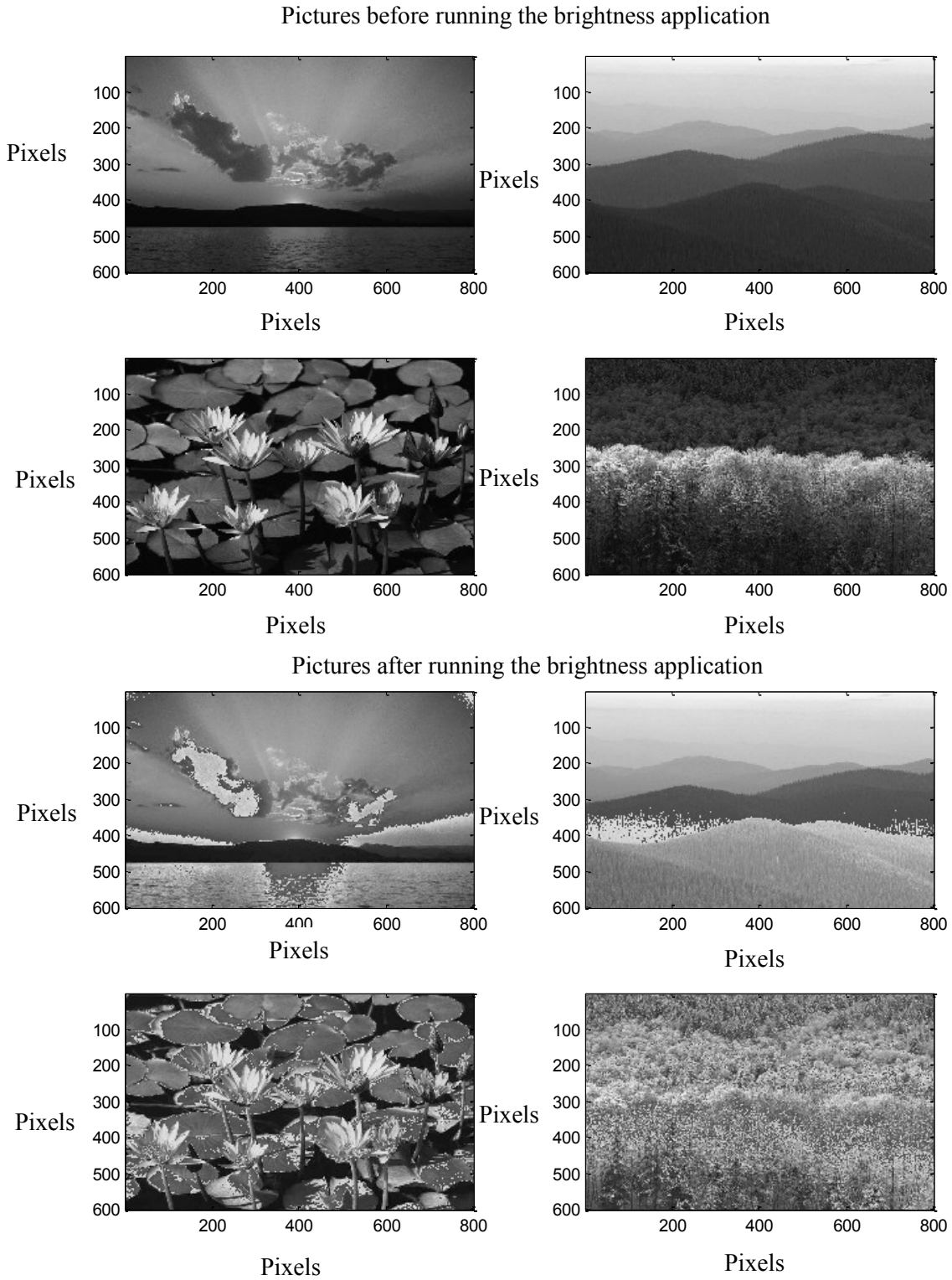


Figure 3-12: Pictures before and after running through the brightness application. Depending on the number of dark pixels, the workload changes thus allowing the processor to save energy

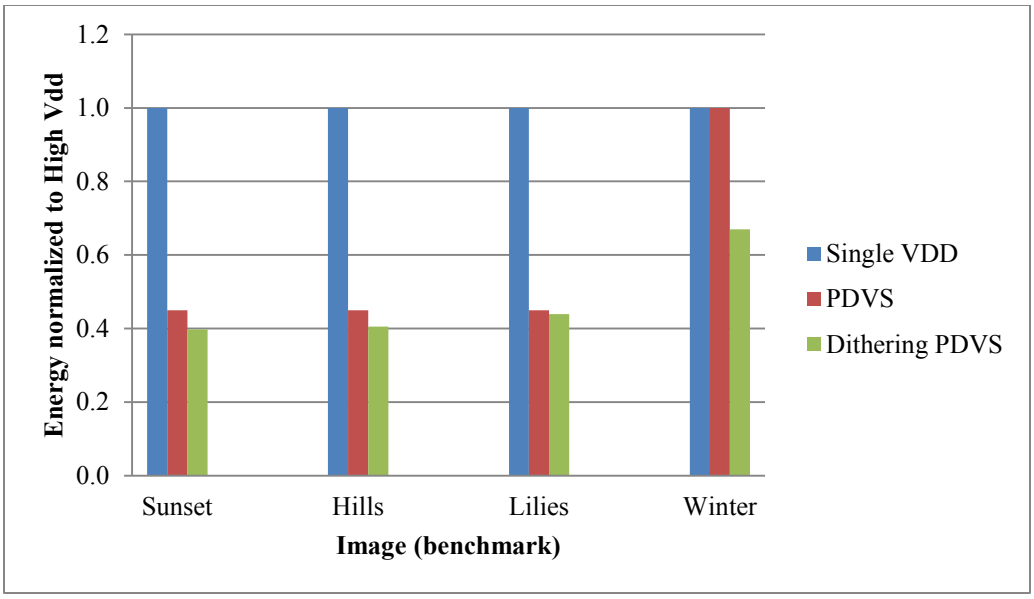


Figure 3-13: Brightening different pictures in PDVS can save different amount of energy depending on the workload

3.6 Conclusions about Voltage Reconfigurability

During this project, we aimed to study the potential benefits of adding configurability to voltage scaling on a finer scale than traditional DVFS. We have seen significant benefits without any big sacrifice of performance. The downside of adding the fine granularity falls mainly on the layout of the power delivery network, which gets pretty more complicated with every additional rail we add. The other components (i.e., the switching headers and the level converters) are small in area, and their energy overhead is relatively insignificant.

We studied as well the changes that had to be done to the scheduling system. The adjustment from other scheduling algorithms is pretty simple and reasonable. One important factor to keep in mind is the switching overhead of the header. With larger components, larger headers are required. This increase in header size increases the energy overhead of switching voltages, which can affect scheduling decisions. While in our case the switching overhead was less than the energy saved in the arithmetic components, we can easily see different systems where the switching overhead can overshadow the energy saved. It is vital to include switching overhead calculations in the scheduling algorithm.

After exploration of voltage selection methods, utilizing simple intuitive voltage-selection techniques seem to provide near-optimal results. However, the granularity of reconfigurability (the number of voltage rails) was surprisingly low. It seems since our system is running at low voltages, it does not need more than two voltage rails. While different systems might have different results, we anticipate that most systems that use relatively low voltages will have greatly diminishing returns after the addition of the second voltage rail similar to our system.

Chapter 4 Reconfigurability at Processor Architecture Level[4]

In this chapter, we will describe the research in designing configurability at the processor level. We aim to create a dynamically reconfigurable SIMD/MIMD system using simple in-order processor components. The reason for using preexisting processor parts as the main components for this system is that with more transistors residing on the dame die, the bottleneck is shifting toward power delivery and consumption, so using many simple cores is now more desirable than one complex, power-hungry core.

4.1 Concept and Background

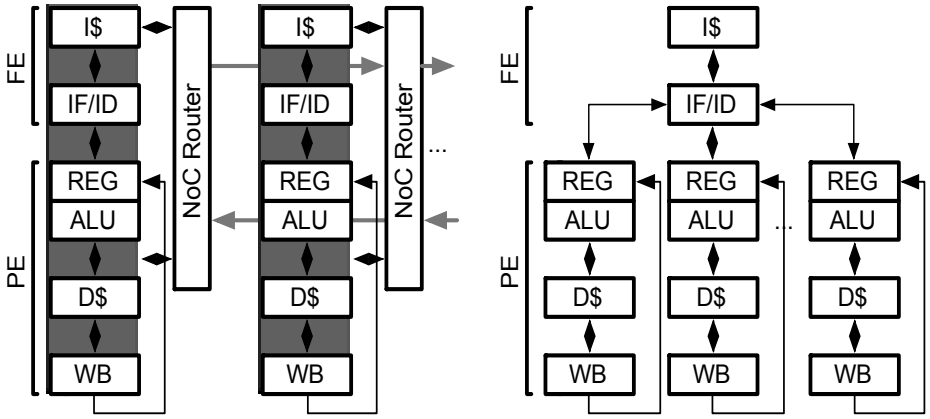


Figure 4-1: Typical MIMD (left) and SIMD (right) block diagrams

To improve performance at a lower system cost and ameliorate power and thermal issues, multicore architectures are increasingly customized to best exploit the available parallelism in a given application or set of applications. Such systems are often organized for data- or thread-level parallelism (DLP or TLP) or both.

DLP and TLP are different enough that, traditionally, distinct architectures have been employed to take advantage of each. Multiple instruction, multiple data (MIMD) architectures, illustrated in Figure 4-1 (left), are optimized for TLP. For the purposes of this discussion, the important feature of MIMD

[4] Most text and data in this chapter are currently under consideration for publication

architectures is that each core has all of the components necessary to fetch and execute instructions independently.

Single instruction, multiple data (SIMD) architectures, illustrated in Figure 4-1 (right), are a cost-saving refinement of MIMD architectures, specifically optimized for DLP. Data-parallel applications perform the same sequence of operations on a large number of data elements. In this case, the components that give MIMD systems the flexibility to execute arbitrary independent threads are unnecessary overhead; these applications can be executed using a single front end (FE) to fetch a single sequence of instructions and distribute it to processing elements (PEs) that operate on different data points. SIMD architectures, therefore, remove redundant FE components, reducing power (fewer instructions are fetched and decoded) at the expense of flexibility (only a single instruction stream is fetched and executed).

While MIMD and SIMD architectures are each able to execute certain classes of applications very efficiently, problems arise with inter- and intra-application parallelism diversity. In the literature, various classes of reconfigurable architectures have been proposed to address variations in available parallelism: Liquid SIMD [30] and Vapor SIMD [31] dynamically adjust SIMD width as DLP changes; other architectures employ heterogeneous cores [32][33][34]; other approaches combine SIMD and MIMD architectures into reconfigurable systems—the focus of this chapter. In Sankaralingam et. al. [25] and E. Mirsky [27], the authors create a reconfigurable SIMD/MIMD system by creating an array of components with a reconfigurable network. Other researchers focus on improving specific applications/algorithms, such as [26], [28], and [29], where a reconfigurable system is created to optimize a specific type of applications. Those researchers have shown the performance benefits for such reconfigurable SIMD/MIMD systems in the presence of parallelism diversity. However, the circuit-level overhead of introducing this reconfigurability has not been explored in depth. This chapter studies that overhead and how it impacts decisions on flexibility in reconfigurable SIMD/MIMD systems.

To investigate these design decisions and trade-offs, we designed a reconfigurable SIMD/MIMD system, by (1) partitioning an in-order core into its FE and PE, (2) interconnecting an array of FE-PE pairs, and (3) controlling which FEs drive which PEs to support various SIMD and MIMD configurations. To explore the design space, we performed circuit- and architecture-level simulations and found that partitioning results in high energy overheard and consequently leads to unexpected design choices. In particular, we found that the energy consumption of cache accesses and FE-PE interconnect dominate the system energy. Consequently, it is more energy efficient to *not* divide cores between pipeline stages like traditional SIMD architectures (e.g., Figure 4-1 [right]); instead, we partition at the interface between the instruction caches (I$) and the instruction fetch (IF) units.

We use our system to analyze the trade-offs between the flexibility of the system and the resulting energy overhead. We found, for this architecture, that due to the interconnect energy overhead, it is better to support a limited number of configurations even though supporting more configurations resulted in a greater number of energy-optimal benchmarks.

4.2 Vertical Integration in Design Decision

Vertical integration refers to any coordination or co-optimization between different design levels. During this work, we explored exposing low-level circuit information to the system-level tools to make better decisions. We also briefly explore the potential benefits and overhead of optimizing multiple components together.

4.2.1 System Design

In SIMD architectures, the main source of energy savings is having multiple PEs under the control of a single FE. Enabling a single FE to control multiple PEs requires that control logic drive long wires, consuming significant energy. This is especially critical as process technology advancements reduce transistor energy faster than the wire energy, making the energy consumed in wires significant [36]. By changing the partitioning point, a designer can change the number of control signals as well as the shared components and therefore the energy overhead/savings of SIMD operation.

We explored a variety of places to divide an OpenRISC [41] core into an FE and a PE: (a) before the instructing fetch (IF) stage (the FE is just the I$), (b) before the instruction decode (ID) stage (the FE is the I$ and IF stages), and (c) before the execution (EX) stage (the FE is the I$, IF, and ID stages). Because of the number of signals passing between the ID and EX stages, we also considered (d) replicating the ID stage in both the FE and the PE.

Using Cadence tools, we analyzed the energy consumption of various partitioning strategies, by determining (1) the per-cycle energy consumption of different processor components, (2) the energy consumption of wires of different lengths, and (3) the number of signals driven from FEs to PEs (and back again) under different partitioning approaches.

Components Energy

The first question we had to ask was how to partition the processor (OpenRISC) into two sections (FE and PE). Putting more components in the FE side will save energy and power during SIMD execution because in SIMD execution, one FE is controlling many PEs. While sharing more components is useful, it affects different parts, which we will mention in the next few sections.

To estimate the energy impact from sharing more or less components, we used circuit-level simulators (Cadence tools) to estimate the energy per cycle for the main components in the OpenRISC processor. The results of this are illustrated in Figure 4-2. We see that the majority of the energy is consumed in the registry file, the floating-point arithmetic units, then in the instruction fetch (IF) and the instruction decode (ID).

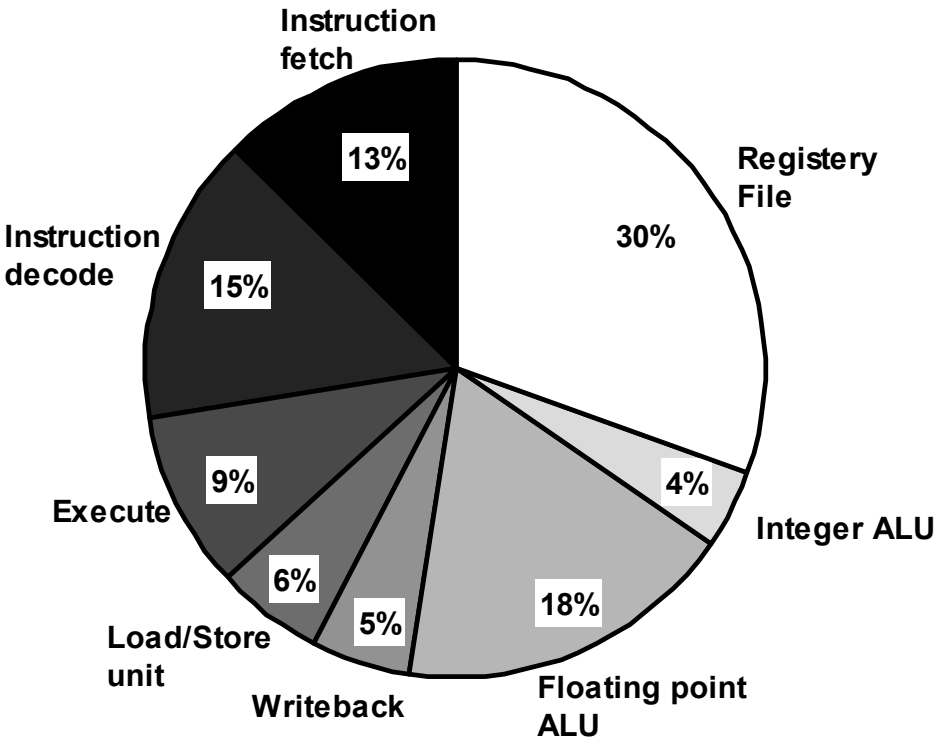


Figure 4-2: OpenRISC's components energy distribution per cycle assuming 100% utilization

Cache Energy

Another significant section of the processor is the cache access energy, which is triggered with every instruction fetch or data read or write. To estimate this energy, we used CACTI 6.5 [42], which is a tool to estimate cache parameters according to the configuration. While this is not a significant contribution of this work, we still had to sweep the variables of cache parameters to choose a reasonable set of parameters for our design. Illustrated in the figures below are the results of sweeping different variables related to caches on cache reading energy and area.

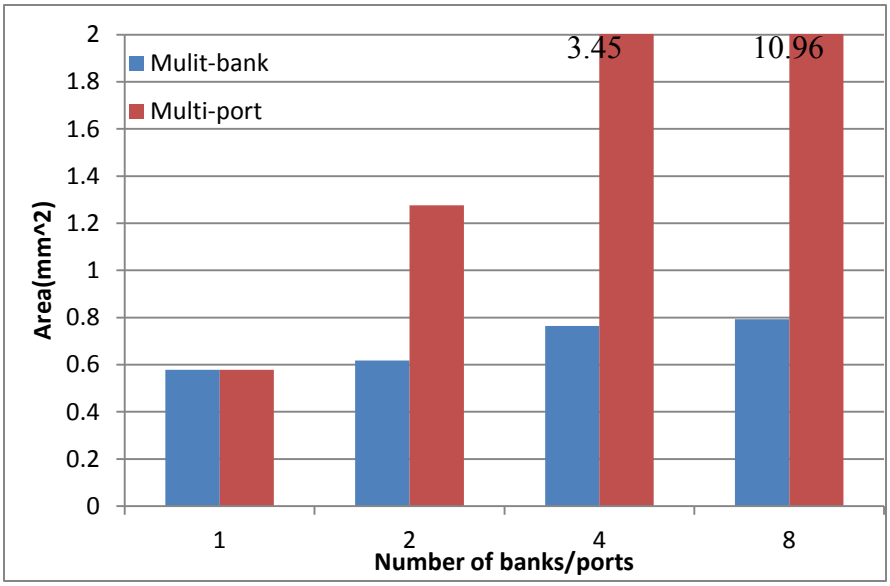


Figure 4-3: Area of cache when varying the number of banks and ports

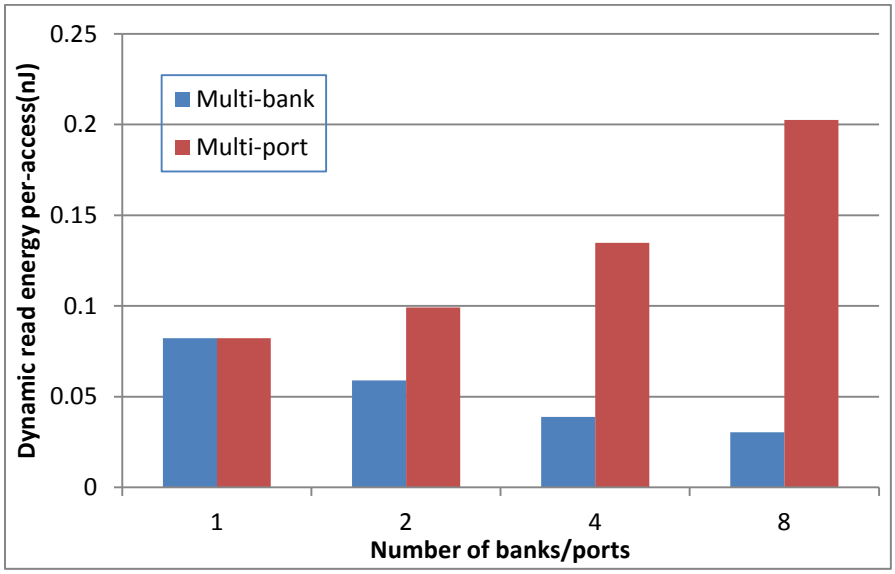


Figure 4-4: Energy of cache read when varying the number of banks and ports

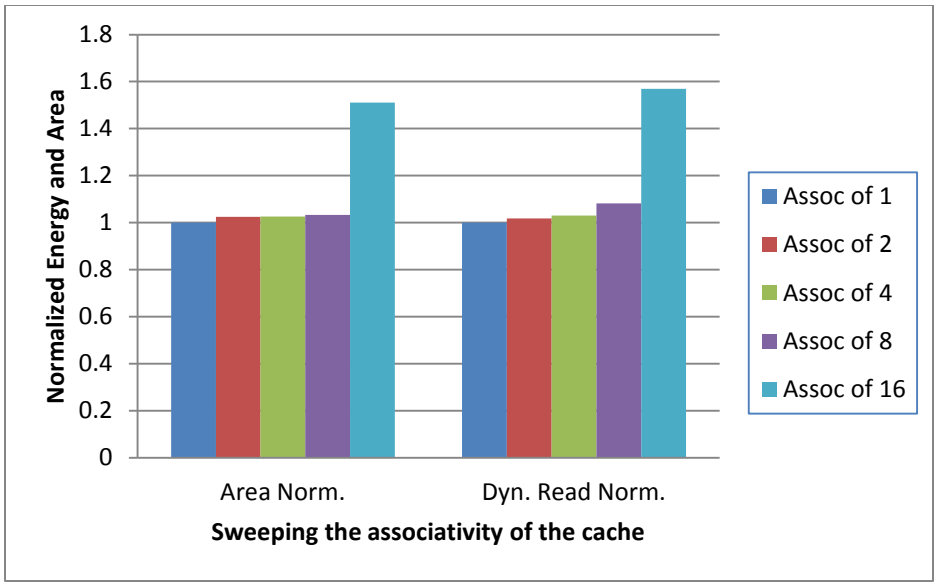


Figure 4-5: Normalized energy and area of cache when sweeping the associativity

From the figures, we see that as long as a reasonable number of ports and associativity is chosen, the cache configuration does not vary a lot. Since this work was concerned with adding configurability to the processor, we only needed a reasonable cache configuration that represents a typical processor.

For our analysis, we used the absolute energy value for reading and writing from such caches to represent the instruction cache (ICACHE), the data cache (DCACHE), and the register file (RF).

Wire Energy

With technology reducing the transistor energy very efficiently, the energy consumed in wires between those transistors became more significant to the system. In this section, we try to estimate how this low-level variable impacts the system architecture of the reconfigurable system. Using Cadence tools to estimate the energy transmitting one bit across different lengths of wires, we created Figure 4-6.

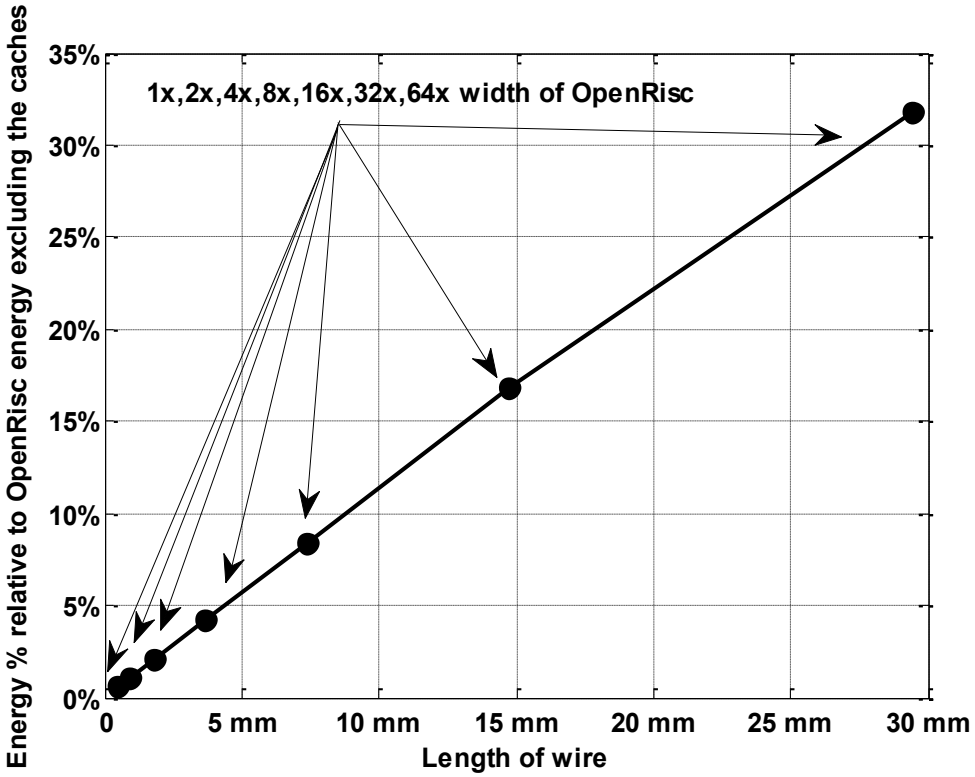


Figure 4-6: Energy of transmitting 1 bit across different lengths of wires

We see, as expected, the trend of energy consumption of transmitting a bit across different lengths of wires is linear. Interestingly, the energy consumed in this transmission is pretty high relative to the processor once the wire is long enough (transmitting a bit 8 OpenRISC cores away cost around 8% relative to the energy of 1 cycle of the component). This energy can grow very fast if we look at the number of bits transmitted between the components.

To better estimate the energy consumed in connections between the different stages of the processor, we counted the number of signals connecting each stage with its following stage. Figure 4-7 shows the number of connections between the FE and the PE if we varied the components distribution among the FE and the PE.

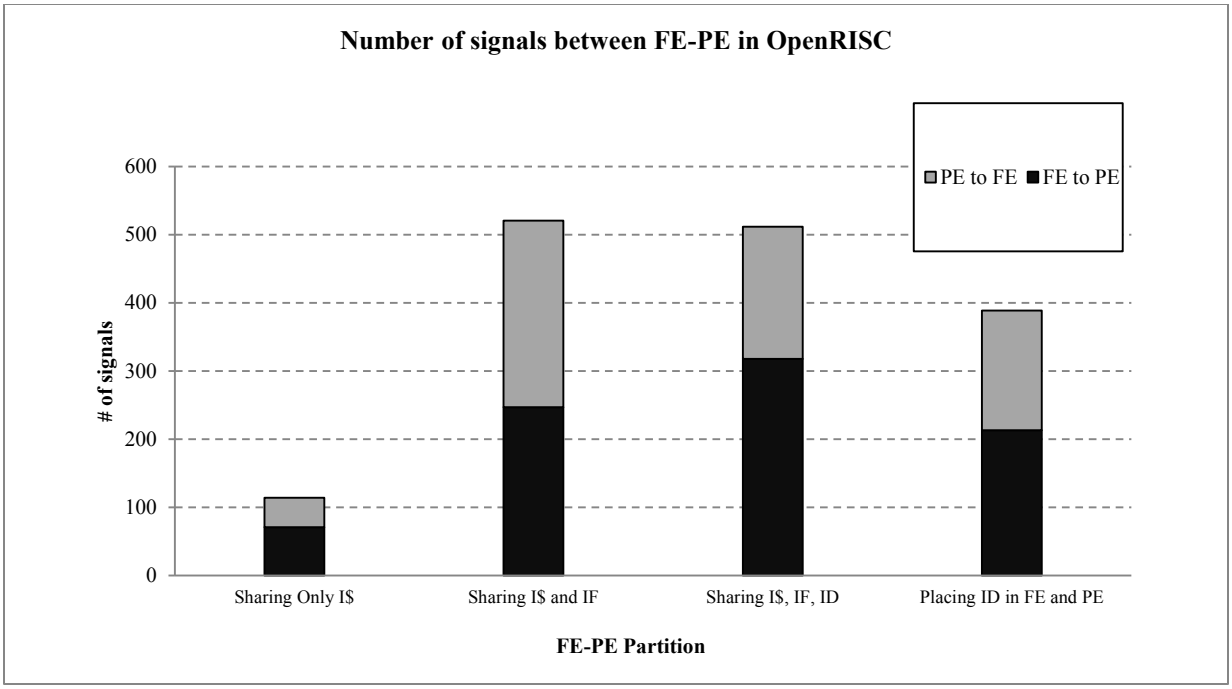


Figure 4-7: Number of signals connecting the FE and the PE under different configurations

We see the numbers of signals between the stages is much higher than the number of signals between the caches and the core. In the section below, we will present the final partitioning and configuration decision for our reconfigurable variable width SIMD/MIMD system.

Overall Energy

The data present before assumed 100% utilization. That means it measured the energy when each of those components is being exercised. However, practically speaking, only few of those components are executing code at a time; and depending on the benchmark, some components are exercised more than others.

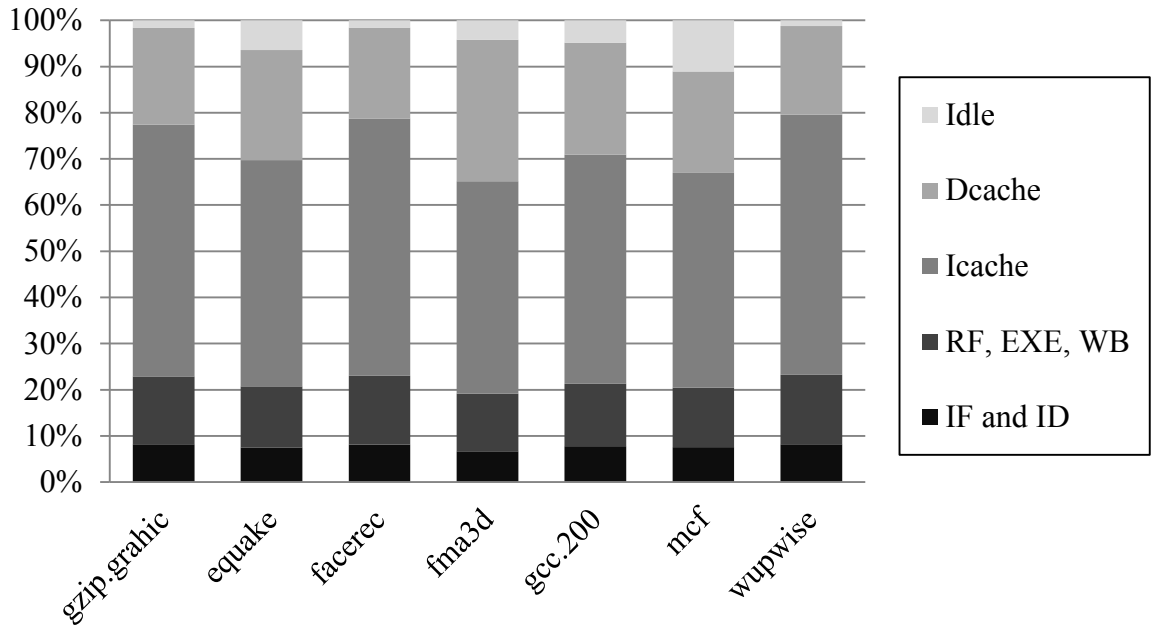


Figure 4-8: OpenRISC's components energy breakdown when running a variety of benchmarks

To better estimate the energy consumption breakdown in the processor during typical benchmarks, we used the architectural simulator gem5 [35] to count the number of instructions that triggers the different components during the execution of several benchmarks used before in [37]. Those benchmarks are extracted from MineBench [38], SPLASH-2[39], and Rodinia [40]. Every instruction will trigger the instruction fetch (IF) and the instruction decode (ID) components; however, only arithmetic operations will trigger the registry file and either trigger the integer ALU or the FPU. Each instruction will trigger different set of components. After taking this into account, we used the ratio of instructions that correspond to different components to have a better breakdown of components energy during benchmarks. Combining all the data extrapolated in the previous sections and using architectural simulators to estimate the utilization of each component according to the benchmarks, we came up with Figure 4-8.

In our system, we aimed to incorporate 16 cores to represent a sizable system that is capable of running highly parallel applications in a relatively efficient manner. Using this assumption and an activity factor

of 0.25 for the signals connecting the processor stages (this is a very standard activity factor value). Our calculations show that the energy of transmitting bits between the FE and the PE for the configurations we considered (sharing the I$ only, sharing the I$ and the IF, sharing the I$ and the IF and the ID, or sharing the I$ and IF and duplicating the ID) is 9.3%, 35%, 43.5%, 29.6%, respectively, relative to a 16-way MIMD configuration.

In conclusion, sharing just the I$ between the different cores results in the least energy overhead despite the fact that it requires redundant instruction fetching and decoding in each PE. Fetch and decode energy (~8%) is far less expensive than driving the extra signals (~25%), as shown previously in Figure 4-8.

The circuit-level energy overhead affected the best granularity of configurability as well. The description of that effect will be included in the following section, which focuses on figuring the best level of configurability such system should have

4.3 Granularity of Reconfigurability

In reconfigurable SIMD/MIMD systems, the number and nature of the supported configurations has significant implications for both system performance and energy overhead. In this section, we estimate the performance and energy benefits gained from supporting more configurations, as well as the reconfigurable interconnect energy overhead. The configurations vary from a 16-way SIMD (one FE controls all 16 PEs) to multiple narrower-width SIMD groups to a 16-way MIMD execution (every FE controls one PE).

4.3.1 Energy Results

Table 4-1: The number of long wires and multiplexers needed to support more configurations

Supported Configurations	16-way SIMD and 16-way MIMD	16-way, 2 8-way SIMD and 16-way MIMD	16-way, 2 8-way, 4 4-way SIMD and 16-way MIMD	16-way, 2 8-way, 4 4-way, 8 2-way SIMD and 16-way MIMD
Multiplexers	15 2to1	8 2to1, 7 3to1	4 2to1, 8 3to1, 3 4to1	4 2to1, 6 3to1, 4 4to1, 1 5to1
Extra Wires in the Interconnect	1 16x, 15 1x	1 16x, 1 8x, 14 1x	1 16x, 1 8x, 2 4x, 12 1x	1 16x, 18x, 2 4x, 4 2x, 8 1x

To estimate the energy overhead of flexibility, we measured the energy of the multiplexers and the long wires required to support different combinations of configurations. We then compare that overhead with the resulting benefit of reconfigurability with respect to improved application efficiency.

Table 4-1 shows the number of multiplexers and long wires needed to support a variety of different combinations of configurations. While other combinations might be better for any specific set of applications, it is very difficult to determine the best combination of configurations to support without prior knowledge of the applications running on the system. For our analysis, we chose what we considered to be reasonable combinations that cover the full range of flexibility possible using 16 cores. Based on Figure 4-7 and Table 4-1, the reconfigurable interconnect energy overhead for supporting two, three, four, or five configuration combinations listed in Table 4-1 is 11.5%, 13.3%, 19.4%, or 26.1%, respectively, relative to the average cycle energy of a static 16-way MIMD.

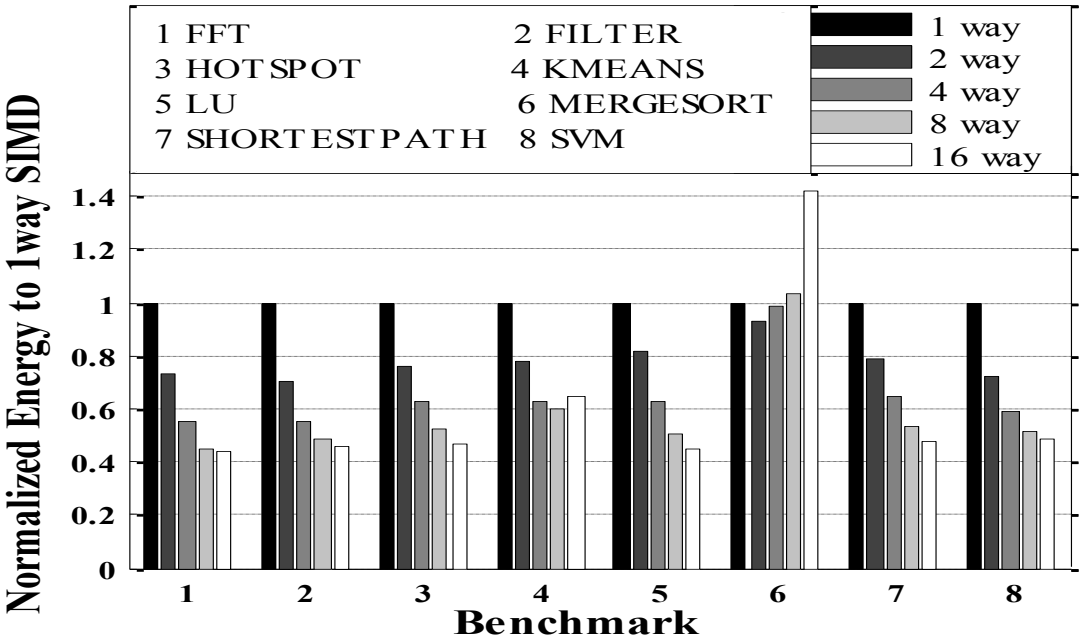


Figure 4-9: Normalized energy of different SIMD width execution of various benchmarks

While supporting more configurations increases the energy overhead of the interconnect, it reduces the energy consumption among various benchmarks since each benchmark has an optimal energy SIMD width. The more configuration we support, the closer to the optimal point we get. To estimate how the number of supported configurations affect the energy efficiecny, we ran the benchmarks on widths varying from 1 to 16. The results are shown in Figure 4-9. The figure shows how different benchmarks have different optimal SIMD widths.

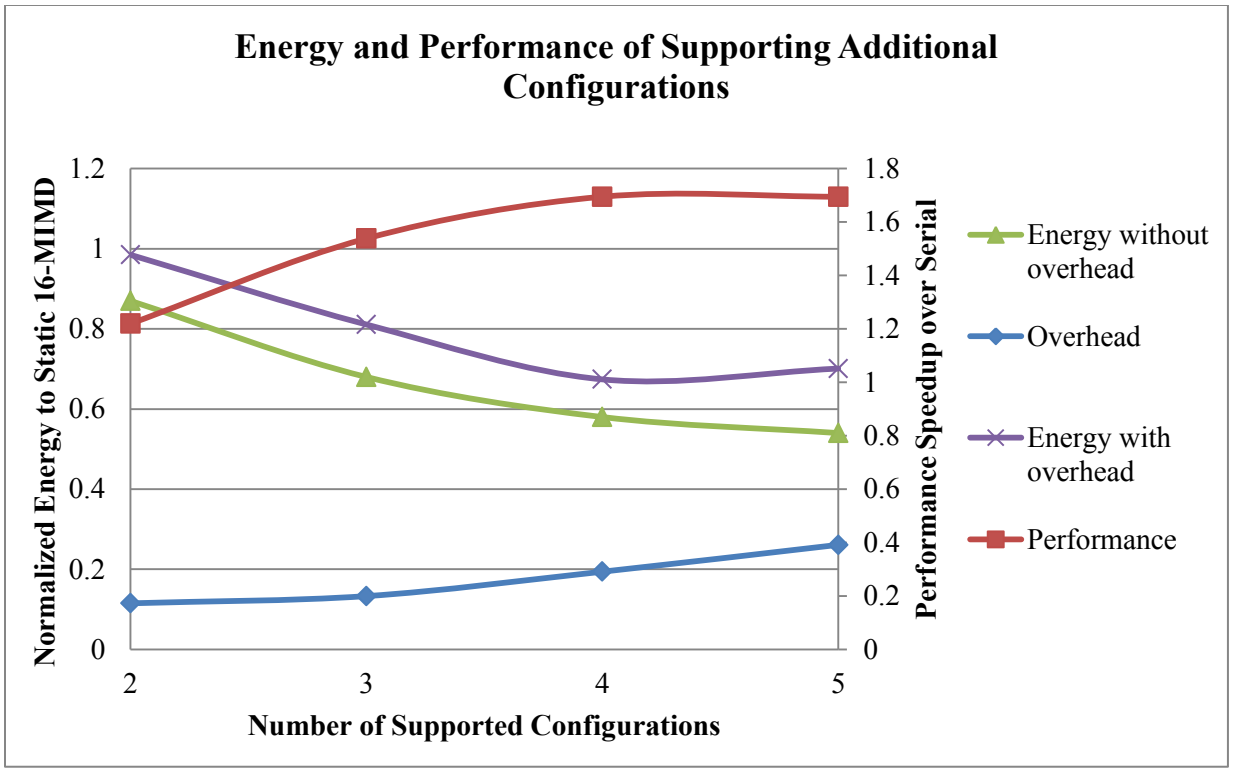


Figure 4-10: Average energy and performance of supporting more configurations (changing the level of configurability).

If we supported two, three, four, and five different configurations, which were listed in Table 4-1, then the average energy consumption of executing all the benchmarks is equal to 87%, 68%, 58%, and 54%, respectively, relative to a 16-way MIMD. While having more configurations reduces benchmark energy by better matching the parallelism of a specific application to the best configuration, each additional supported configuration adds energy overhead to the reconfigurable interconnect as seen previously. Adding the reconfigurable interconnect energy overhead of supporting two, three, four, and five configurations to the benchmark energy results shows that the overall system consumes on average 98.5%, 81.1%, 67.4%, 70.1%, respectively, relative to the energy of running the applications using a 16-way MIMD. We observe that due to the overhead of interconnect, supporting four configurations produces the optimal flexibility for our system rather than supporting more configurations. The results that directed us to this conclusion are shown in Figure 4-10.

4.3.2 Performance Results

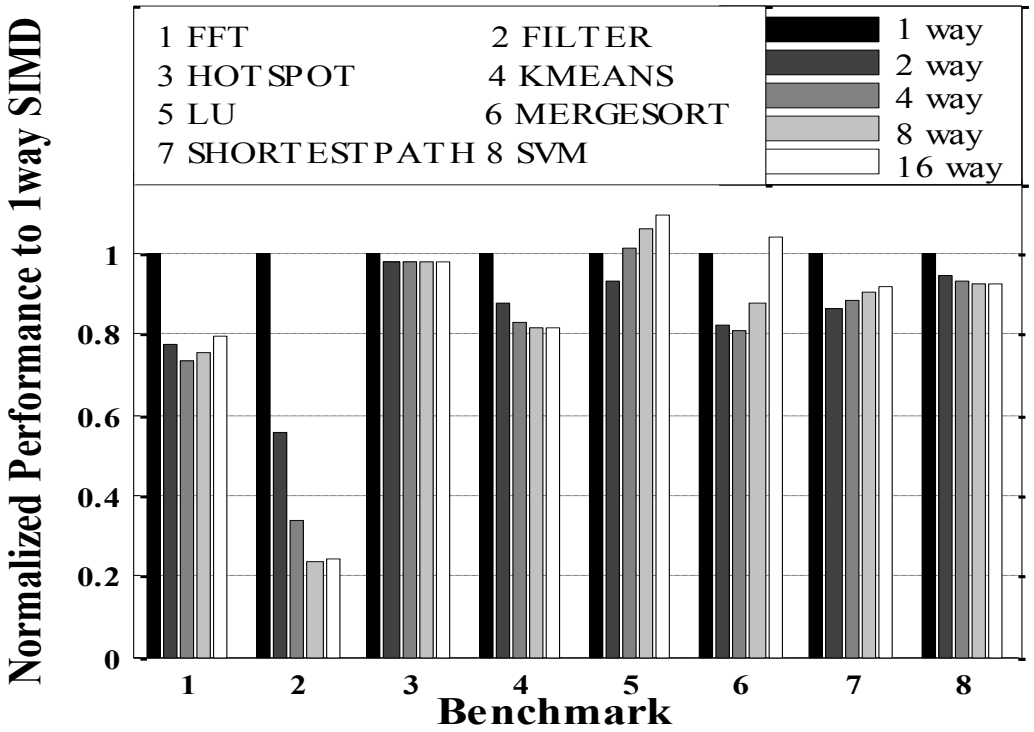


Figure 4-11: Normalized performance of various SIMD width systems

One of the main benefits of increasing the flexibility of reconfigurable SIMD/MIMD systems is to improve the performance consumption of applications by switching to the right configuration for the specific application. To determine the performance benefits each additional supported configuration option provides, we ran eight benchmarks on gem5 [35] with different SIMD widths; our results are illustrated in Figure 4-11. Assuming the system can switch perfectly to the optimal configuration for each benchmark, we found that the performance supporting two, three, four, and five configurations can achieve, on average of all the benchmarks, runtimes equal to 82%, 65%, 59%, and 59%, respectively, relative to serial execution.

As expected, the performance and energy benefits of supporting more configurations have a decaying nature. In our specific case, supporting four configurations achieved a similar performance as five

configurations. The fact that performance stopped improving after four configurations while the overhead of supporting more configurations is always increasing, it is clear that the optimal number of supported configurations for our system and for our suite of benchmarks is four configurations. While the specific number of supported configurations will vary from one system to another, it is interesting to point that the returns of configurability diminished very fast.

4.4 Sweeping Variables

In this section we briefly analyze the different variables in such a system that will affect the energy optimal partitioning configuration (i.e., deciding what components should be in the FE and the PE).

In the figure below we swept the energy of transmitting 1 bit in a wire across 16 cores to analyze at what point each partitioning configuration become the energy optimal one. We assumed a 0.2 activity factor for the signals going from FE to PE and 0.1 factor for the signals from PE to FE. The reason we assumed that different activity factors between FE to PE and PE to FE is because running in SIMD mode, one FE will be sending commands to all PEs for every instruction but not all instructions require commands back to the FE.

According to Figure 4-12, we see that different configurations are optimal at different transmission energies. If the system we used had a more energy consumed in the IF and ID components or if the wires used do not consume as much energy in transmitting signals, then the optimal configuration would have been affected significantly. However in our case, the wires were about 8% of the core's energy, so that is why in our system it was clearly better to only share the I$ among the cores rather than sharing more components.

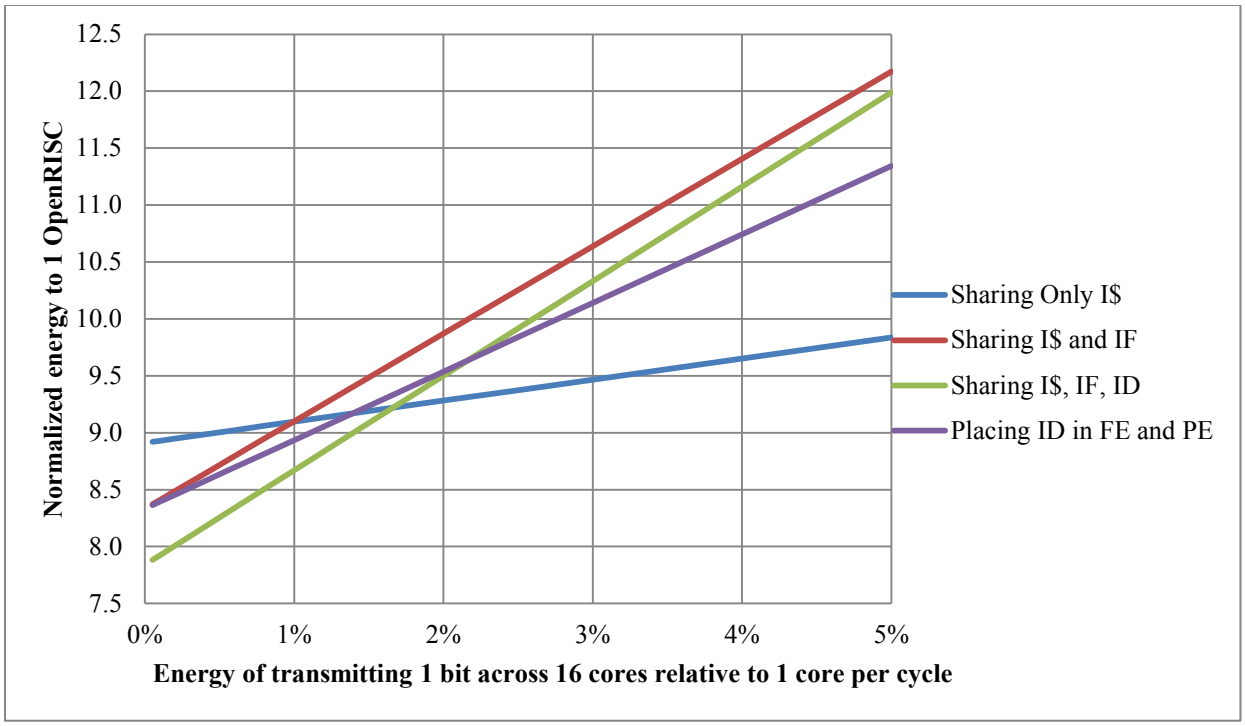


Figure 4-12: Energy of Paraflex according to various configurations if the wires energy was different. It is clear that the wire energy was a main factor of choosing a configuration

4.5 Conclusions about Adding Reconfigurability to Processors

Reconfigurable SIMD/MIMD systems are able to dynamically respond to changing application parallelism and therefore employ the most energy-efficient way to achieve optimal performance. We analyzed the circuit-level energy overhead of such systems by designing a simple reconfigurable SIMD/MIMD architecture, and we found out that due to the high energy overhead of control signals, it is better to minimize the communication between blocks by sharing only the instruction cache. Cache structures consume a significant amount of energy relative to simple in-order cores, and partitioning at any other point significantly increases the communication between SIMD FE and PE.

We also analyzed the benefit and energy overhead of supporting more SIMD/MIMD configurations in the system. We found that supporting four configurations seems to achieve the best energy savings overall without sacrificing any significant performance, consuming on average 70.1% relative to a 16-way MIMD.

We analyzed which components of the processor use the most energy, and we have created an appropriate reconfigurable system accordingly. We saw that cache configurations in general produce a similar amount of cache access energy, and this energy during typical benchmarks consumes a significant percentage of the total energy of the system. Reducing the energy of cache access for simple processors like OpenRISC seems the most effective way to increase the energy efficiency of the system.

Chapter 5 Summary of Work and Contributions

In this chapter, we go over a summary of the main tasks we have done and an overview of the main conclusions we had about vertically integrated dynamically reconfigurable systems.

5.1 Summary of Work

In this work, we aimed to increase energy efficiency of systems by adding vertically integrated dynamic configurability to them. When we added configurability to the compression algorithm, the results were much better without incurring much overhead. We saw that even small embedded systems can benefit of dynamic reconfigurability if done with other aspects of the design in mind. We showed that simple metrics typically used to compare algorithms cannot be used when dealing with special systems like low energy embedded body sensors and different set of metrics spanning to various design levels should be considered.

We showed how adding voltage reconfigurability at a finer granularity improves energy efficiency significantly. During the process of adding this configurability, we went over the main additional aspects this configurability adds, like how the scheduling system has to change to accommodate voltage selection of individual operations. To maximize the energy efficiency of the reconfigurable system, we had to incorporate vertical integration by looking at multiple design layers. We explained how the switching overhead due to the transistors in the headers affects the scheduler since it will prevent some operations of running at a lower voltage due to the switching overhead. We showed that including some of the low-level information into the higher-level design decisions increases energy efficiency without incurring unreasonable efforts.

We showed a comparison of the methods of selecting the voltages for multi–voltage rail systems. We showed how simple methods, which are used traditionally, are actually reasonable effective, like using the delay of components. We investigated the number of voltage rails a system should support to maximize benefits, and we saw very limited benefits of adding more than two rails. The benefit of adding more rails gets overshadowed very fast by the complexities it causes to the rest of the design, mainly in the power

delivery network. While the quantitative analysis of those effects is outside the scope of this work, we have discussed qualitatively the effect of adding more voltage rails to the rest of the system.

In the programmable core array project, we showed how reconfigurability at a relatively high design level (system architecture) is still very useful and can be done without recreating every component in the processor. We used existing components from the OpenRISC core to build a bigger and a reconfigurable system capable of switching between various width SIMD and MIMD configurations. During the design of this system, we investigated the effect of interconnect energy consumption on the overall system. We also showed how including this information in the architecture decision has significantly changed the design and without it the energy efficiency of the system would have suffered. We analyzed the energy overhead of adding the configurability, we studied the benefits of adding or decreasing the level of configurability, and we showed that we reach near-optimal results even with a limited level of configurability and how adding more configurability will just add more overhead than benefits.

In general, this work focused on designing systems from beginning to end using reconfigurability and low-level circuit information in system-level decisions to show the potential of dynamic configurability and vertical integrations. Throughout this work, we showed many times how configurability is currently underutilized and that it should be a more explored option in designing new systems. We showed how this configurability combined with vertical integration can save a significant amount of energy without significant negative effects on other aspects.

5.2 Individual Contributions

- Developed a simple adaptive compression algorithm suitable for BASN devices by adjusting preexisting algorithm ideas
- Analysis of the created compression technique against others from the aspect of BASN devices
- Equal partner in creating and designing the first processor that uses PDVS
- The main designer of the block-level design of the PDVS chip

- The designer of the microcode, the scheduling algorithm, and the series of programming tools for the PDVS chip
- Main partner for the measurement and testing/debugging of the PDVS chip
- Readjusting the RTL Verilog of OpenRISC to create a partition used for creating different SIMD and MIMD widths
- Synthesize, place, and route several FPCA configurations for power and area estimations
- Performed performance and energy analysis on the benchmark results (running the benchmarks was not my individual contribution)

5.3 General Guidelines and Recommendations

- Adaptive delta encoding compression provides a very high energy-saving potential, especially with its simplicity, which enables implementability on most embedded devices.
    - As see in the BASN project where simple algorithms can make big difference.
- Having small components switching voltages independently is energy efficient.
    - As seen in the PDVS project, increasing spatial and temporal voltage scaling granularity is beneficial.
- If the voltage reconfigurability is targeting small components, few voltage levels are sufficient to extract most of the potential energy savings.
    - As seen in PDVS, the number of rails determines the number of voltage options each component has. Working on such spatial granularity means a greatly decaying benefit of each additional option.
- If the level of configurability is relatively high, then having more options is desirable.
    - As seen in FPCA, since we are changing the width of SIMD and affecting the whole system (coarse granularity), supporting relatively high amount of configurations (widths of 1, 2, 4, and 16) was more beneficial.

- The overhead of the circuit levels should always be considered in system architecture and layout for it might impact things heavily.
    - As seen in the FPCA project, the transmission bits between different components dominated the energy consumption which affected the optimal energy configuration.
- Exposing circuit-level information to system-level tools is an efficient way to optimize energy.
    - As seen in the PDVS project, giving the scheduler the switching overhead information is sufficient to save energy without requiring complex co-optimization.

5.4 Open Questions and Future Work

Though this work have discussed many questions related to introducing configurability and vertical integration to a system, there are many more that have not been addressed. The configurability and vertical integration aspects are very generic and have almost unlimited supply of questions to be answered. Here we will present some of the questions and topics that we felt we would have tried to answer if more work is done on this topic.

This work was focused on energy efficiency more than anything; it would be interesting to see if power consumption or performance were the main aspects to optimize. Would the design decision differ, or would there be an optimal configuration that optimizes most of those aspects?

In each of the projects we worked on, we still have numerous variables we would have liked to explore and analyze—like the number of arithmetic components connected to one header or if other types of single components should have access to voltage scaling.

During our work with a reconfigurable processor, we wondered if the decisions we made would be significantly different if other processors were used as our base in-order processor. Would the energy in wires still be as significant as it was in OpenRISC? What about different configurations that we could have supported, like federation, where two in-order cores act as a single out-of-order processor.

In all the projects, we always wondered how the results would differ if a different set of benchmarks were used. This was particularly interesting because we were strongly limited in the benchmarks we could use either because they use many features our custom PDVS processor does not possess or because the benchmarks are not compatible with the architectural simulators we used, like in our reconfigurable processor project.

Another aspect of reconfigurability that we did not have time to explore is how and when to actually switch the configurations. We investigated the overhead of the switch, but we did not explore algorithms that can determine when the hardware needs to switch to a different configuration according to the running application. This aspect was too complicated and large to be addressed and could become a stand-alone project by itself.

While there are many more questions to ask, those are some of our main questions that we wish we had time to explore. We hope in our future work or with the support of other researchers, we would be able to answer some of these questions.

Appendix A: Publications and References

[1] S. Arrabi and J. Lach, "Adaptive Lossless Compression Techniques for Body Area Sensor Networks" (BodyNets, 2009).

[2] B. H. Calhoun, S. Arrabi, S. Khanna, Y. Shakhsheer, K. Craig, J. Ryan, and J. Lach, "REESES: Rapid Efficient Energy Scalable ElectronicS," *GOMACTech* (2010).

[3] S. Khanna, K. Craig, Y. Shakhsheer, S. Arrabi, J. Lach, and B. H. Calhoun, "Stepped Supply Voltage Switching for Energy Constrained Systems" (ISQED, 2011).

[4] Y. Shakhsheer, S. Khanna, K. Craig, S. Arrabi, J. Lach, and B. H. Calhoun, "A 90nm Data Flow Processor Demonstrating Fine Grained DVS for Energy Efficient Operation from 0.25V to 1.2V" (CICC, 2011).

[5] K. Craig, Y. Shakhsheer, S. Khanna, S. Arrabi, J. Lach, B. H. Calhoun, and S. Kosonocky, "A Programmable Resistive Power Grid for Post-Fabrication Flexibility and Energy Tradeoffs" (ISLPED, 2012).

[6] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and K, Rabaey, "Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System" (VLSI, 2001).

[7] S. Sen, V. Natarajan, R. Senguttuvan, and A. Chatterjee, "Pro-VIZOR: Process Tunable Virtually Zero Margin Low Power Adaptive RF for Wireless Systems" (DAC, 2008).

[8] N. Zea, J. Sartori, B. Ahrens, and R. Kumar, "Optimal Power/Performance Pipelining for Error Resilient Processors" (ICCD, 2010).

[9] R. Jain and P. Panda, "Memory Architecture Exploration for Power-Efficient 2D-Discrete Wavelet Transform" (VLSI, 2007).

[10] P. Stelling and V. Oklobdzija, "Implementing Multiply-Accumulate Operation in Multiplication Time" (Computer Arithmetic, 1997).

[11] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany, "The Imagine Stream Processor" (VLSI, 2002).

[12] K. Usami and M. Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design" (ISLPED, 1995).

[13] R. Puri, D. Kung, and L. Stok, "Minimizing Power with Flexible Voltage Islands" (ISCAS, May 2005).

[14] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S. Fekete, and J. Veen, "A Practical Approach for Circuit Routing on Dynamic Reconfigurable Devices" (RSP, 2005).

[15] C. Yang and A. Orailoglu, "Predictable Execution Adaptivity through Embedding Dynamic Reconfigurability into Static MPSoC Schedules" (CODES+ISSS, 2007).

[16] Y. Zhao, A. Erdogan, and T. Arslan, "A Low-Power and Domain-Specific Reconfigurable FFT Fabric for System-on-Chip Applications" (IPDPS, 2005).

[17] T. Jhaveri, V. Rovner, L. Liebmann, L. Pileggi, A. Strojwas, and J. Hibbeler, "Co-Optimizing of Circuits, Layout and Lithography for Predictive Technology Scaling Beyond Gratings" (TCAD, 2009).

[18] V. Iyengar, K. Chakrabarty, and E. Marinissen, "Efficient Wrapper/TAM Co-Optimization for Large SOCs" (DATE, 2002).

[19] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic Voltage Scaling on a Low-Power Microprocessor" (SIGMOBILE, 2001).

[20] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads" (ICCAD, 2002).

[21] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. Rabaey, "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications" (ISSCC, 2000).

[22] V. Rana, M. Santambrogio, and D. Sciuto, "Dynamic Reconfigurability in Embedded System Design" (ISCAS, 2007).

[23] J. Owens, M. Houston, D. Luebke, S. Green, J, Stone, and J. Phillips, "GPU Computing" (JPROC, 2008).

[24] H. C. Powell Jr., M. A. Hanson, and J. Lach, "A Wearable Inertial Sensing Technology for Clinical Assessment of Tremor" (IEEE Biomedical Circuits and Systems Conference, 2007), 9–12.

[25] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous Trips Architecture," *IEEE Micro* 23 (2003): 46, 51.

[26] C. Lyuh, J. Suk, I. Chun, and T. Roh, "A Novel Reconfigurable Processor Using Dynamically Partitioned SIMD for Multimedia Applications," *ETRI Journal* 31 (2009).

[27] E. Mirsky and A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources" (FCCM, 1996).

[28] N. Shohei, K. Shorin, and O. Shinichiro, "Performance Evaluation of a Dynamically Switchable SIMD/MIMD Processor by Using an Image Recognition Application" (IPSJ transactions on system LSI design methodology, 2010), 47–56.

[29] P. Sinha, A. Sinha, and D. Basu, "A Novel Architecture of a Re-configurable Parallel DSP Processor" (IEEE-NEWCAS Conference, the 3rd International, 2005), 71, 74.

[30] N. Clark, A. Hormati, S. Yehia, S. Mahlke, and K Flautner, "Liquid SIMD: Abstracting SIMD Hardware Using Lightweight Dynamic Mapping" (HPCA, 2007).

[31] D. Nuzman, S. Dyshel, E. Rohou, I. Rosen, K. Williams, D. Yuste, and A. Cohen, "Vapor SIMD: Auto-Vectorize Once, Run Everywhere" (CGO, 2011).

[32] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction" (MICRO, 2003).

[33] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multi-Threaded Workload Performance" (ISCA, 2004).

[34] H. H. Najaf-abadi, N. K. Choudhary, and E. Rotenberg, "Core-Selectability in Chip Multiprocessors" (PACT, 2009).

[35] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R., Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The GEM5 Simulator" (ACM COMP AR, 2011).

[36] G. Guindani, C. Reinbrecht, T. Raupp, N. Calazans, and F. G. Moraes, "NoC Power Estimation at the RTL Abstraction Level" (VLSI, 2008).

[37] J. Meng, J. W. Sheaffer, and K. Skadron, "Robust SIMD: Dynamically Adapted SIMD Width and Multi-Threading Depth" (IPDPS, 2012).

[38] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A Benchmark Suite for Data Mining Workloads" (IISWC, 2006).

[39] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations" (ISCA, 1995).

[40] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A Performance Study of General Purpose Applications on Graphic Processors Using CUDA" (JPDC, 2008).

[41] http://opencores.org/or1k/Main_Page

[42] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.5: A Tool to Model Large Caches."

[43] K. C. Barr and K. Asanović, "Energy-Aware Lossless Data Compression" (ACM Transactions on Computer Systems, 2006), vol. 24, no. 3, 250–291.

[44] Bluetooth RN-41 Class datasheet, http://www.rovingnetworks.com/documents/RN-41.pdf

[45] N. Kimura, and S. Latifi, "A Survey on Data Compression in Wireless Sensor Networks" (International Conference on Information Technology: Coding and Computing, 2005), vol. 2, 8–13.

[46] W. Lu, and M. Gough, "A Fast-adaptive Huffman Coding Algorithm" (IEEE Transactions on Communications, 1993), vol. 41, no. 4, 535–538.

[47] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-Low Power Data Storage for Sensor Networks" (ACM International Conference on Information Processing in Sensor Networks, 2006), 374–381.

[48] J.C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential Benefits of Delta Encoding and Data Compression for HTTP" (ACM SIGCOMM Computer Communication Review, 1997), vol. 27, no. 4, 181–194.

[49] MSP430F1611 datasheet, http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf

[50] P. R. Panda, N. D. Dutt, and A. Nicolau, "On-Chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based Systems" (ACM Transactions on Design Automation of Electronic Systems, 2000), vol. 5, no. 3, 682–704.

[51] D. Petrovic, R. Shah, K. Ramchandran, and J. Rabaey, "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks" (IEEE International Workshop on Sensor Network Protocols and Applications, 2003), 156–162.

[52] H. C. Powell Jr., M. A. Hanson, and J. Lach, "A Wearable Inertial Sensing Technology for Clinical Assessment of Tremor" (IEEE Biomedical Circuits and Systems Conference, 2007), 9–12.

[53] D. Puccinelli, and M. Haenggi, "Wireless Sensor Networks: Applications and Challenges of Ubiquitous Sensing," *IEEE Circuits and Systems Magazine* 5, no. 3 (2005): 19–31.

[54] J. S. Vitter, "Design and Analysis of Dynamic Huffman Coding" (Annual Symposium on Foundations of Computer Science, 1985), 293–302.

[55] K. Usamia and M. Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design" (International Symposium on Low Power Design [ISLPED], 1995), 3–8.

[56] D. Gajski, N. Dutt, A. We, and S. Lin, "High-Level Synthesis Introduction to Chip and System Design" (Kluwer Academic Publishers, the Netherlands, 1994).

[57] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs" (Computer-Aided Design [CAD], 1989), 661–679.

[58] R. Camposano, "Path-Based Scheduling for Synthesis" (Computer-Aided Design [CAD], 1991), 85–93.

[59] W. Zhang, L. Zhang, A. Shayan, W. Yu, X. Hu, Z. Zhu, E. Engin, and C. Cheng, "On-Chip Power Network Optimization with Decoupling Capacitors and Controlled-ESRs" (Asia and South Pacific Design Automation Conference [ASPDAC], 2010).

[60] A. Shrivastava and B. H. Calhoun, "Modeling DC-DC Converter Efficiency and Power Management in Ultra Low Power Systems" (IEEE Subthreshold Microelectronics Conference [SubVT], 2012), 1–3.

[61] K. Mazumdar and M. Stan, "Breaking the Power Delivery Wall Using Voltage Stacking" (Great Lakes Symposium on VLSI [GLSVLSI], 2012), 51–54.

[62] W. Kim, D. M. Brooks, and W. Gu-Yeon, “A Fully-Integrated 3-Level DC/DC Converter for Nanosecond-Scale DVS with Fast Shunt Regulation” (International Solid-State Circuits Conference [ISSCC], 2011), 268–270.

[63] J. Huang, J. Bian, Z. Llio, and Y. Wang, “A Fast Algorithm for Power Optimization Using Multiple Voltages in Data Path Synthesis” (International Conference on ASIC, 2005), 900–904.

[64] S. P. Mohanty and N. Ranganathan, “Energy-Efficient Datapath Scheduling Using Multiple Voltages and Dynamic Clocking” (ACM Transaction on Design Automation of Electronic Systems [TODAES], 2005), 330–353.

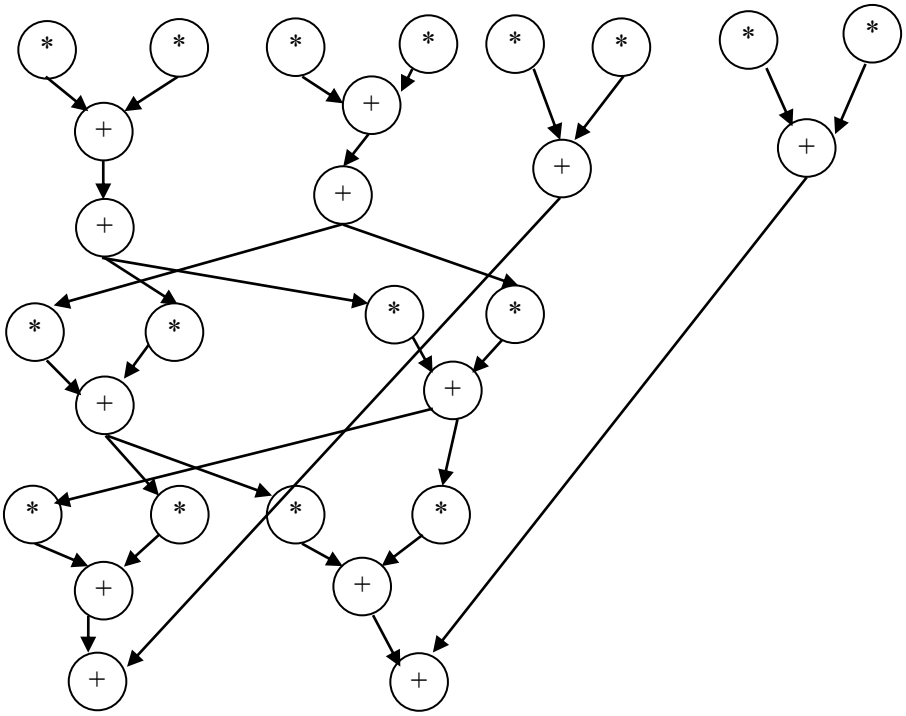
[65] D. Chen, J. Cong, and J. Xu, “Optimal Simultaneous Module and Multivoltage Assignment for Low Power” (ACM Tranaction on Design Automation of Electronic Systems [TODAES], 2006).

[66] H. Yang and L. Dung, “On Multiple-Voltage High-Level Synthesis Using Algorithmic Transformations” (Design Automation Conference [DAC], 2005), 872–876.

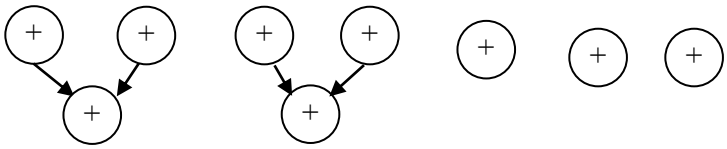
[67] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, “Analyzing and modeling encryption overhead for sensor network nodes” (ACM international conference on Wireless sensor networks and applications [WSNA], 2003). 151-159.

Appendix B: Data Flow Graphs of Benchmarks

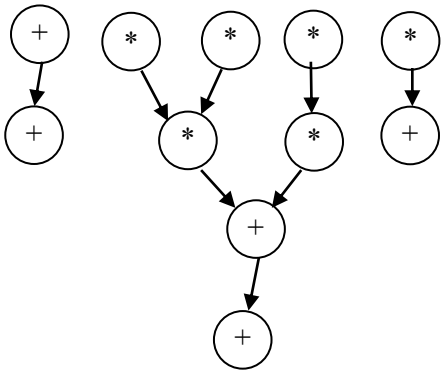
AR Lattice



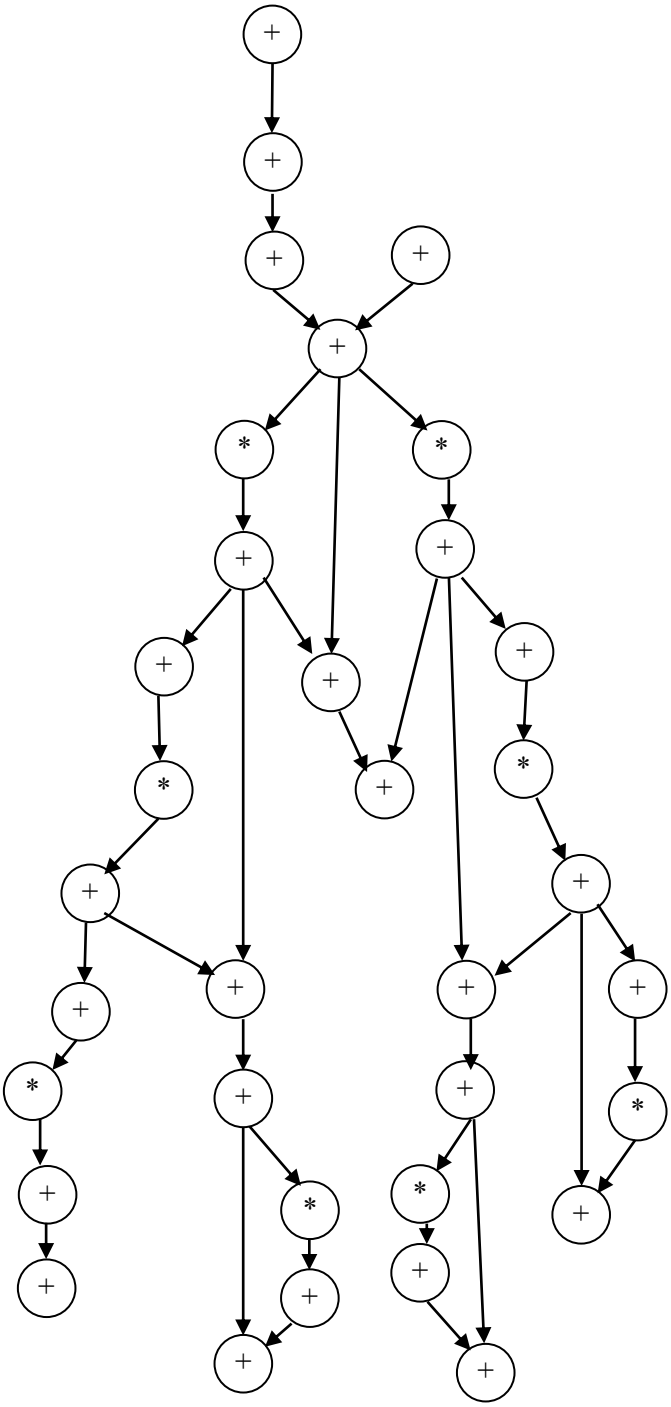
FFT



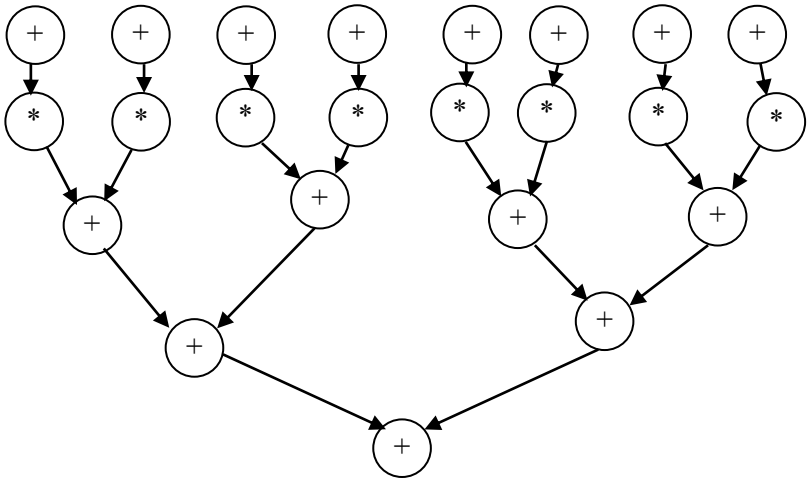
DiffEq



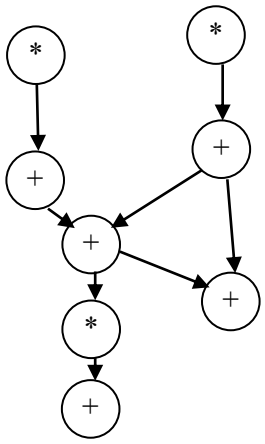
Ellip



FIR



All Filter



GCD