2015

ENERGY EFFICIENCY EXPLORATION OF COARSE-GRAIN RECONFIGURABLE ARCHITECTURE WITH EMERGING NONVOLATILE MEMORY

Xiaobin Liu
University of Massachusetts Amherst

ENERGY EFFICIENCY EXPLORATION OF COARSE-GRAIN RECONFIGURABLE ARCHITECTURE WITH EMERGING NONVOLATILE MEMORY

A Thesis Presented

by

XIAOBIN LIU

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2015

Electrical and Computer Engineering

ENERGY EFFICIENCY EXPLORATION OF COARSE-GRAIN RECONFIGURABLE ARCHITECTURE WITH EMERGING NONVOLATILE MEMORY

A Thesis Presented

by

XIAOBIN LIU

Approved as to style and content by:

Russell Tessier, Chair

Sandip Kundu, Member

Wayne Burleson, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

ACKNOWLEDGMENTS

I would like to thank Jesus Christ, my Lord and Savior, for leading me here and guiding me through all challenges.

I would like to express profound gratitude to my advisor in computer engineering department from University of Massachusetts, Amherst, Professor Russell Tessier. Thank you for your support for my master thesis research and helpful advices. Without his guide this master thesis will not be possible. I really appreciate that you let me know what I will be capable of and how to train myself as a serious computer engineer.

I would like to thank all colleagues in Reconfigurable Computing Group. Justin Lu provides so many insightful comments, his hard-working attitude and expertise in computer engineering envision me of what to do and how to do it for my research. I also want to thank Tedy Thomas, who worked with me on this thesis project and he contributed a lot in generating results in terms of STTRAM. He also provides insightful views of our MRAM implementation. KeKai Hu and Deepak gave me help when I got stuck in technical problems.

I want to thank my parents, they are always supportive and thank you for your unconditional love. Thank you for giving me this opportunity to get access to different cultures and technologies. Thank you for your understanding throughout my studies.

Last but not least, I want to give special thanks to my fiancee Jacqueline Chung. Thank you for being with me through my frustration, thank you for your comforting and encouraging. They did mean a lot to me. I would like to dedicate this thesis to my family.

ABSTRACT

ENERGY EFFICIENCY EXPLORATION OF COARSE-GRAIN RECONFIGURABLE ARCHITECTURE WITH EMERGING NONVOLATILE MEMORY

FEBRUARY 2015

XIAOBIN LIU

B.Sc., TONGJI UNIVERSITY

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Russell Tessier

With the rapid growth in consumer electronics, people expect thin, smart and powerful devices, e.g. Google Glass and other wearable devices. However, as portable electronic products become smaller, energy consumption becomes an issue that limits the development of portable systems due to battery lifetime. In general, simply reducing device size cannot fully address the energy issue.

To tackle this problem, we propose an on-chip interconnect infrastructure and program storage structure for a coarse-grained reconfigurable architecture (CGRA) with emerging non-volatile embedded memory (MRAM). The interconnect is composed of a matrix of time-multiplexed switchboxes which can be dynamically reconfigured with the goal of energy reduction. The number of processors performing computation can also be adapted. The use of MRAM provides access to high-density storage and lower memory energy consumption versus more standard SRAM technologies. The combination of CGRA, MRAM, and flexible on-chip interconnection is considered for

signal processing. This application domain is of interest based on its time-varying computing demands.

To evaluate CGRA architectural features, prototype architectures have been prototyped in a field-programmable gate array (FPGA). Measurements of energy, power, instruction count, and execution time performance are considered for a scalable number of processors. Applications such as adaptive Viterbi decoding and Reed Solomon coding are used for evaluation. To complete this thesis, a time-scheduled switchbox was integrated into our CGRA model. This model was prototyped on an FPGA. It is shown that energy consumption can be reduced by about 30% if dynamic design reconfiguration is performed.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Embedded Signal Processing Trends

As the use of mobile applications expands, data transfer between users and service providers is a bottleneck for "everything in your pocket". Concepts such as wearable devices and cloud computing continue to push the edge of wireless communication. Improved wireless communication would allow consumers to have richer user experiences, such as fast file transfers, multimedia streaming and online gaming. It has been projected that the daily average for received data per device will increase from the current 10MB value to over 280MB by 2020, possibly reaching 680MB by 2028 [27, 43]. Figure 1.1 shows that computing power would be quickly drained by drastic increases in requirements of multimedia applications. This motivation is now the single most important driver for semiconductor foundries to keep fighting deviations from Moore's Law. Even though new transistor technologies continue to be proposed and manufactured, e.g. multigate transistor, performance gains can no longer be assured simply by boosting clock frequency. New hardware architectures optimized for embedded signal processing must be proposed to take advantage of the additional transistors, achieving a goal of high data transfer rate while maintaining an energy budget.

Many low power techniques have been proposed at the circuit, architecture and device levels to address the energy efficiency problem over the past decade [17, 31, 36]. For digital signal processing algorithms, an application-specific integrated circuit (ASIC) would be the best solution for an application in terms of power and per-

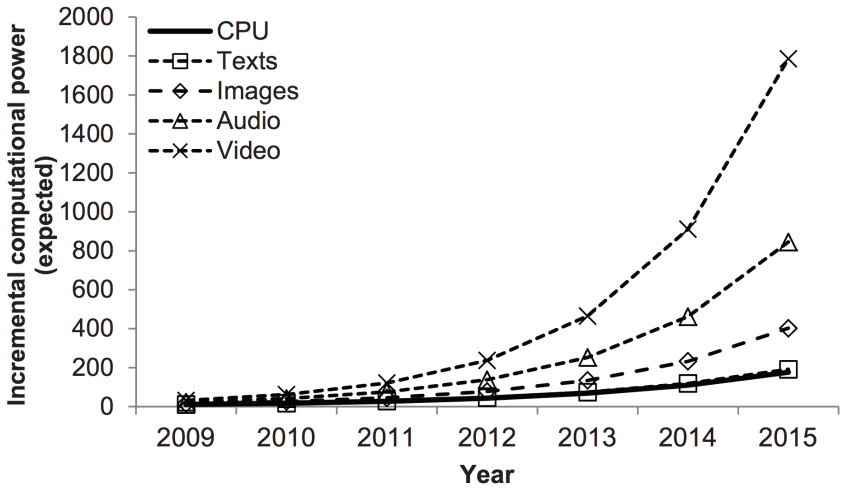


Figure 1.1: Computational power requirements for social sites in the form of text, image, video and audio along with available CPU computing power [42].

formance characteristics [8]. However, the high cost and limited flexibility of ASICs prohibit them from reaching market quickly. To obtain acceptable performance and platform flexibility, a digital signal processor (DSP) has been proposed and developed. This architecture is optimized for digital signal processing datapaths. In state-of-the-art DSPs, a general-purpose processor (e.g. an ARM processor) is coupled with other coprocessors (e.g. network and graphics processing units) as well as a DSP core [52]. A system-on-chip solution provides high programmability for a wide variety of applications, but power dissipation largely constrains its usage in mobile devices which are powered by batteries. The inherent sequential nature of DSPs also limits parallelism extraction which could support speedup. Another significant bottleneck is the latency associated with data transfer between off-chip memory and on-chip registers [14].

To balance hardware performance and software flexibility, reconfigurable computing has been proposed to fill the gap between DSPs and ASICs. Reconfigurable systems are classified into two categories based on the size of their elementary logic elements: fine-grained and coarse-grained. The former can be reconfigured at the bit level while the latter generally contains functional units at the word level. The Field

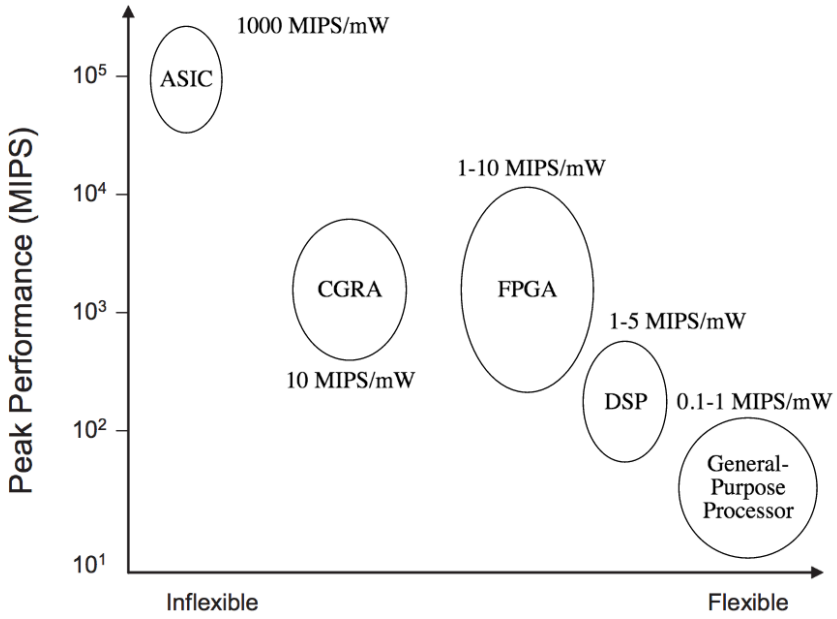


Figure 1.2: Comparison of peak performance, power efficiency, and programmability of different architecture design styles [22].

Programmable Gate Array (FPGA) is the most common fine-grained architecture [42]. FPGAs are capable of exploiting hardware benefits by parallelizing repetitive datapaths at the gate level. SRAM-based LUTs are fine-grained devices which can implement arbitrary logic functions, increasing ease of programming. However, most signal processing algorithms use many wide and complex arithmetic operations. This is not optimal for FPGAs since this requirement leads to excessive interconnect use [21]. Another issue is energy and area. Conventional FPGAs are composed of a sea of SRAM blocks. Though the 6-T architecture of an SRAM provides acceptable read and write speed, it sacrifices area and energy to achieve this goal.

To address the previously described problems of fine-grained reconfigurable architectures, two architectural solutions can be considered: coarse-grained reconfigurable architectures (CGRA) [7, 21] and emerging non-volatile memory (NVM) [32, 29, 26].

Coarse-grained reconfigurable architectures feature multiple simple processing elements (PE) which are composed of a complete ALU and peripheral circuitry (shown

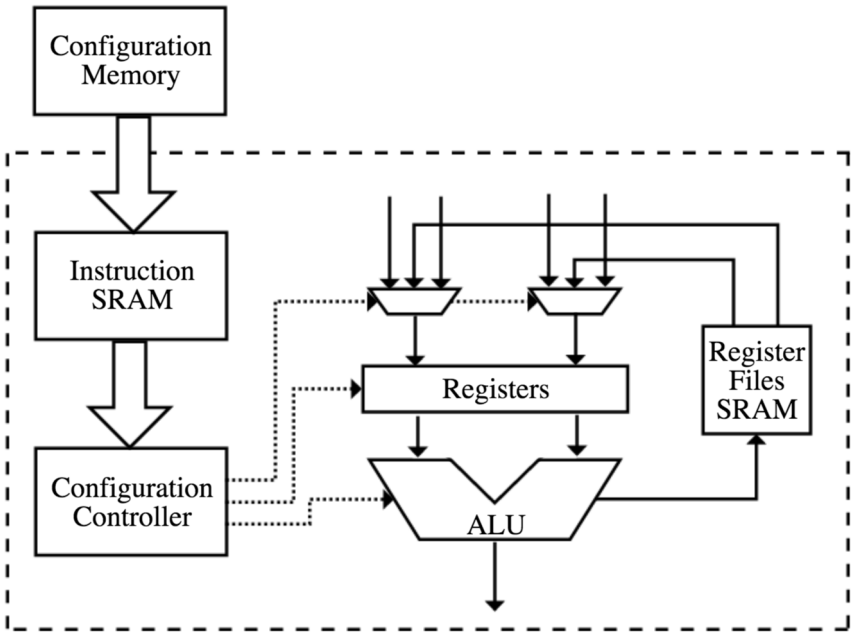


Figure 1.3: Details of a typical CGRA processing element (PE)

in Figure 1.3). The PE itself cannot be modified, although its interconnection is reconfigurable. A prime advantage of coarse-grained versus fine-grained reconfigurables is that coarse-grained architectures need less configuration data than FPGAs do, which results in faster reconfiguration speed (they can adapt to applications quicker). Coarse-grained architectures are suitable for computation-intensive workloads while nonvolatile memory offers resources for improvements on memory-intensive tasks. These two solutions form the cornerstone of our innovative architecture which targets energy efficiency while maintaining a pre-specified performance level. Figure 1.2 depicts pros and cons of different digital signal processing platforms in terms of performance, programmability and power.

Coarse-grained reconfigurable arrays (CGRAs) that include a grid of simple ALU-based processing elements and limited inter-element communication have been shown to be effective at implementing signal processing applications [21]. These devices are typically better suited for streaming applications that can be parallelized across abundant CGRA arithmetic units than their fine-grained field-programmable gate

array (FPGA) counterparts. Often, the amount of programming information needed to configure a CGRA can reach into the millions of bits [21] and the amount of time needed to change configurations can be an impediment to rapid application update. This issue can be addressed by caching multiple application configurations on chip in SRAM, preventing the need for time-consuming accesses to off-chip DRAM. However, for many embedded applications, the extra leakage energy dissipated by SRAM-based configuration memory is unacceptable.

In this thesis work, we address concerns about CGRA reconfiguration time and energy in response to changing environmental parameters for signal processing applications. Our main contribution is the storage of multiple CGRA configurations in *magnetic RAM (MRAM)* close to their target resources. This emerging memory architecture is well suited to bulk configuration storage, given its non-volatile and low leakage characteristics. The reduced size of MRAM in contrast with SRAM allows for increased configuration storage with the ability to support numerous applications, both active and inactive. A full CGRA architecture including compute blocks, interconnect, and configuration cache is introduced and validated. Our work includes a full evaluation of MRAM implementation tradeoffs and interfacing. We demonstrate that this memory design is rapidly responsive to changes in processing demands, providing *just enough* computation to the target application.

1.2 Thesis Overview

In this thesis we address some of the problems which have prevented CGRAs from being used as a more energy-efficient solution. We evaluate these advancements through the use of architectural techniques:

- In signal processing tasks, the computational workload is not always constant in terms of complexity. A good example is coding/decoding algorithms. For some Forward Correction Codes (FCC), the code rate is largely dependent on

the signal-to-noise ratio (SNR), which represents how much data in a wireless channel is likely to be corrupted. If the SNR is temporarily low, the data received could be unreliable, and more computing power and effort would be required to decode and correct possible errors in the received data stream. In such a scenario, more processing elements (PEs) of a CGRA would be preferred to maintain an acceptable throughput, although this action may lead to an increase in energy. This result is acceptable as long as long term energy consumption is reduced. Alternatively, it would be possible to power down some PEs if the signal quality is higher to reduce energy consumption.

- In each PE, an ALU needs access to configuration memory to determine functionality and interconnection and register files to perform computing. In a conventional CGRA, memory is built from 6-Transistor SRAM cells. Since SRAM is faster than other prevailing memory techniques (e.g DRAM and flash memory), SRAM is commonly the choice for on-chip memory solutions for high-performance computing. However, the 6 transistors in each SRAM cell lead to significant leakage, making SRAM power-hungry. Besides energy, an SRAM cell occupies 3 times the area of a DRAM cell. Portable and wearable devices and other embedded systems, could benefit from reduced energy consumption. A new memory technique is needed which can provide significant capacity while allowing for fast configuration access.

- Interconnect accounts for a large portion of area (80%-90% of total area) [53] and energy consumption (50%-60% of total energy consumption) [55] in spatial archiectures. To better use on-chip resources, time-scheduled interconnect was proposed. Some CGRA architectures including RaPiD [16] and Matrix [38] optimize the interconnect via a combination of scheduled and static switching. However, neither of these two projects studied an optimal mixture of these two

design styles. If all of the interconnect is statically configured, the area and delay will be improved but there is then no flexibility in signal routing. At the other extreme, if interconnect is fully scheduled, distributed configuration memory and associated control circuitry would result in a large area overhead. To better exploit the benefits of implementing scheduled/static combined interconnect in a CGRA, a tradeoff needs to be studied carefully to maintain reasonable area and desirable performance for digital signal processing algorithms.

The goal of this thesis is to investigate these issues for a CGRA which utilizes emerging embedded NVM. Our proposed CCGRA is capable of dynamically reconfiguring its topology and the functionality of each PE to adapt to time-varying computational complexity. A time-switched interconnect will be investigated to find the appropriate mixture of time-scheduled and static interconnect [55]. Specific research described in this document includes:

1. A CGRA prototyped in an FPGA with soft processors as the PEs. Three digital signal processing applications are implemented to test this prototyped CGRA to measure power, energy and throughput. Possible energy reductions due to reconfiguring the CGRA architecture are studied. The reconfiguration overhead of the CGRA approach is quantified.

2. A new time-multiplexed interconnect to allow CGRA processor cores to communicate in an energy-efficient fashion. Routing is implemented with multiplexed switchboxes and unidirectional wires. In this design, the topology of the interconnection could be reconfigured based on DSP algorithm parameters. Performance and area is studied to evaluate this interconnect. The energy benefits of the fast configuration of the interconnect is also investigated.

3. Implementation of MRAM in the proposed CGRA architecture is investigated to determine how it would interact with configuration memory and configura-

tion bits of the time-scheduled switchbox. Timing overhead is estimated with regards to loading instructions and configuration bits from MRAM to SRAM for different applications. Energy improvements based on this interface is factored into the final analysis of the overall system.

In this thesis, we target a CGRA which can be reconfigured rapidly to adapt to different applications or computation complexity changes within a single application. With the emerging NVM technology and time-scheduled interconnection, significant energy reduction could be gained without impacting performance.

1.3 Thesis Outline

In Background chapter, we review the development of digital signal processing platforms and the benefits from CGRAs. NVM and its interaction with CGRAs will be introduced. Three algorithms are studied for energy performance tradeoffs from system-level reconfiguration. Chapter 3 focuses on the detailed architecture of prototyped CGRAs. Its energy advantage over time-varying environment is shown through preliminary experimentation. Chapter 4 describes an on-chip time-scheduled interconnect that provides predictable energy savings. Chapter 5 describes a memory hierarchy and interface between emerging NVM and conventional on chip SRAM.

CHAPTER 2

BACKGROUND

In this chapter, background is provided about CGRA architectures, emerging non-volatile memory (NVM) and developing trends of digital signal processing. Applications, such as those used for wireless communication, can often be adapted over time to save energy. Previous work [48][4][51] has indicated the benefits of using reconfigurable devices for applications which can take advantage of dynamic changes in their operating environment, an issue we review in this chapter.

2.1 Coarse-Grained Reconfigurable Architectures

Our experimentation in this thesis is focused on CGRAs. In this section we provide an overview of typical CGRA architecture and usage.

2.1.1 Processing Element (PE) Architecture

CGRAs are composed of a matrix of PEs and interconnect. Each PE typically includes an ALU, a register file and associated controlling circuitry [5]. Interconnect in CGRAs is typically realized using switchboxes [55] or simple networks-on-chip (NoCs) [47] which allow for dynamic switching during application execution. An architectural overview is shown in Figure 2.1.

In each PE, the ALU performs arithmetic operations. It has interfaces to registers and data memory, where final and intermediate results are stored. There is a dedicated central controller that determines what data should be routed to the register file, in what mode the ALU is operating and access to the register file. The

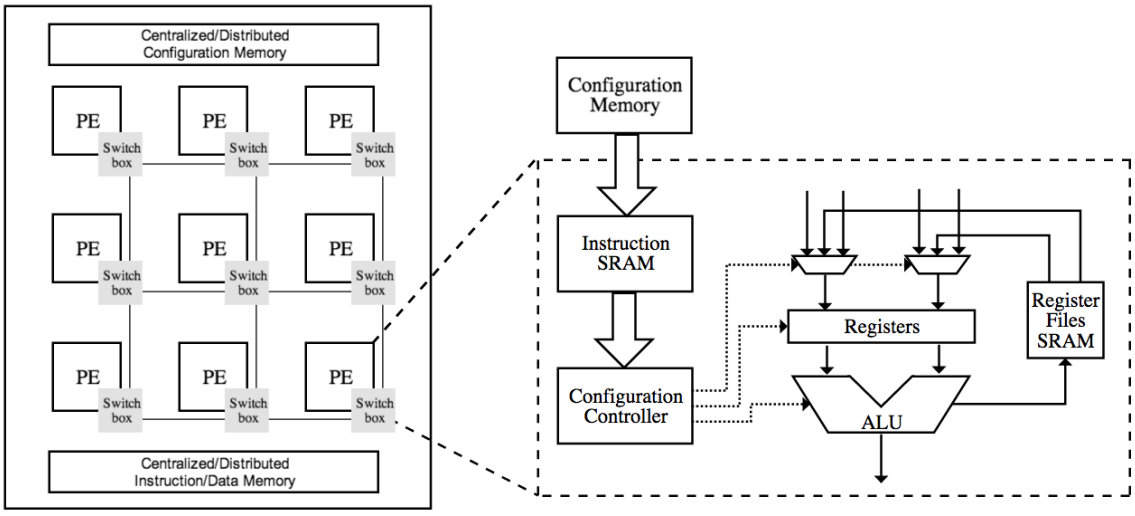


Figure 2.1: Overview of a Typical Coarse-Grained Reconfigurable Architecture

configuration memory can store a variety of different programs which can be quickly accessed as application needs change. Unlike general-purpose RISC cores, CGRA PEs are generally simple (e.g. no memory hierarchy or deep pipelining). Generally, all instructions for a task implemented by the core are located in the Instruction SRAM. Since the computing load for a single PE is fairly low, typically all needed instructions are loaded into the instruction memory before running the application [37]. One of the goals of our work is to implement Configuration Memory in MRAM so new instruction sets can be very quickly loaded into the Instruction Memory on demand.

2.1.2 Interconnect Infrastructure

Interconnect for our planned CGRA will be implemented by an array of switch-boxes inspired by [54]. On-chip switchbox hardware connects wires in adjacent channels together forming paths between processing elements. In general, switchboxes can use unidirectional or bidirectional wires [33]. For unidirectional wires, only one driver is present, while bidirectional wires can have multiple drivers. Although most CGRAs use bidirectional wiring, we will explore both types of interconnect in this thesis, with

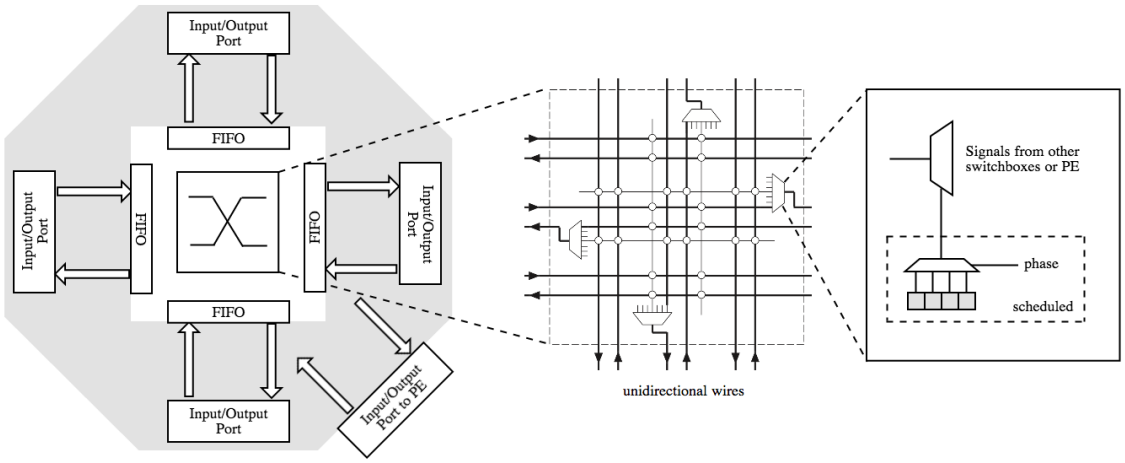


Figure 2.2: Architecture Overview of Time-Scheduled Switchbox

an emphasis on unidirectional wiring. For a track in a channel, there are two wires of opposite directions to realize bidirectional communication. An architectural overview is shown in Figure 2.2. Inside the switchbox, there are multiplexers enabling data flow to turn left, right, up, or down. The multiplexers are configured by SRAM bits.

To reduce energy consumption, our proposed interconnect infrastructure features time scheduling. Instead of statically mapping the data flow onto wires, we expect that a cycle-by-cycle assignment of data to wires could give a better area utilization of routing resources. The dynamic data mapping is realized by a *phase* signal, which configures a multiplexer by selecting a configuration bit every cycle. The pattern of *phase* signal over time could be reconfigured to adapt to different applications. Less area overhead could be obtained. A similar approach, outlined in [54], indicated that area-delay products could be reduced by two-thirds versus a static connection. Unlike the previous approach, we will also consider the impact of using MRAM to configure the switchboxes.

2.1.3 Previous CGRAs

CGRA research started with the development of the PADDI [10] architecture in the 1990s. During the subsequent period of time, multiple designs with different

Table 2.1: Comparison of different Coarse-Grained Reconfigurable Architecture[53]

Project	Granularity	Interconnect Fabric	Mapping Tool	Target App
RaPid	16 bit	segmented buses	channel routing	pipelining
PipeRench	128 bit	(sophisticated)	scheduling	pipelining
MorphoSys	16 bit	mixed	Manual	N/A
Mosaic	32 bit	time-scheduled switchbox	SPR Tool [23]	loop pipelining

granularity, fabric, mapping and target applications have been proposed, as discussed below.

The Reconfigurable Pipelined Datapath (RaPiD) [19] aims to accelerate highly repetitive and computationally intensive datapaths using a control flow which has low data dependency. The architecture takes advantage of deep pipelines to provide high throughput for streaming applications organized in a single dimension. Our two-dimensional approach focuses on a broader range of applications. The PipeRench architecture [24] is an accelerator that also primarily features the acceleration of linear applications. What differentiates it from RaPiD is its capability to reconfigure interconnect dynamically. If an application is too large to fit into available pipeline stage "strips", a controller will dynamically reconfigure available hardware to accommodate the task.

MorphoSys [46] is composed of a tiny on-chip RISC processor and a reconfigurable array (RA). MorphoSys features three layers of interconnect in the RA. Each layer provides connection to a nearest neighbor, inside a sub-quadrant and between quadrants. It aims to lower data transfer latency at the cost of interconnection complexity. In this thesis, we propose a time-scheduled interconnect infrastructure derived from a mesh architecture realized by multiplexed switchboxes instead.

Mosaic [39] is a CGRA which is optimized for reduced energy consumption. Supporting tools include a programming language, compiler, and mapping tool. Mosaic is shown to accelerate computing-intensive loops in general-purpose applications. It features a von Neumann sequential controller and a parallel compute engine, which is

a reconfigurable PE array. The CGRA part is composed of an array of clusters. Each cluster features four 32-bit processing elements, data memory and register files. All components in a cluster are connected by a crossbar. Inter-cluster communication is realized by an array of time-scheduled switchboxes connected by unidirectional wires. Our work extends this type of architecture to consider the impact of using MRAM to store large amounts of configuration information. By using MRAM instead of SRAM we hope to dramatically reduce energy consumption.

2.1.4 Architectural Advantage in Signal Processing

Digital signal processing algorithms feature two characteristics that benefit CGRAs [6]:

- ***Math-Centricity***: Digital signal processing algorithms generally require word-level mathematical operations (e.g. add, subtract, multiplication, division). In most cases, digital signal processing applications feature computations which can be parallelized, represented as a stream, or both. These computations can often be accelerated by a collection of parallel processing elements whose interconnection is configurable.

- ***Standards***: Digital signal processing commonly takes place after analog-to-digital conversion (ADC). Conventionally, ADCs generate data in multi-byte fashion. For example, for Reed-Solomon decoding, a common symbol width is 8 bits. CGRAs are generally built to operate on word or double-word data. Instead of operating on bit-level operands, the internal architecture of a CGRA features multi-bit channel widths in routing channels and processing elements. This parallelism minimizes interconnect configuration memory, hence saves energy. This fixed width comes at the cost of unused resources if the word width of the CGRA does not match the width of the required computation.

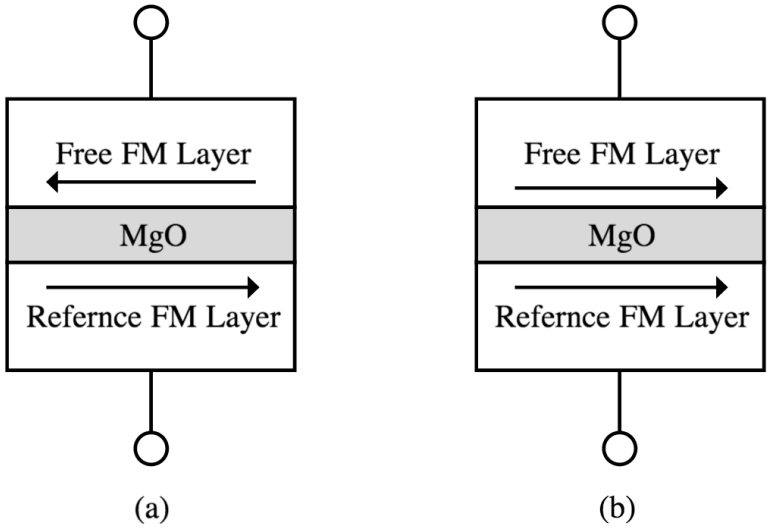


Figure 2.3: Architecture Overview of Magnetic Tunneling Junction [12].
(a) refers high resistance, (b) refers low resistance

2.2 Emerging Non-Volatile Memory (NVM)

Non-volatile memory refers to memory that holds data after a supply voltage is removed. Examples of NVM include read-only memory (ROM), flash memory and magnetic storage device like hard disk. The speed of NVM technologies can sometimes be low. In this section we review new emerging NVMs which have high capacity and relatively low access times.

2.2.1 MRAM Overview

Recently, two kinds of emerging NVM technology, phase-change RAM (PCRAM) and spin-transfer torque magnetoresistive RAM (STT-MRAM) [25, 28, 50], have received interest. STT-MRAM is generally considered the more promising candidate and it is the NVM of choice for this work. Unlike SRAM or DRAM that measures the voltage of a memory cell to read stored data, STT-MRAM uses resistance. A key component in STT-MRAM is the magnetic tunneling junction (MTJ) [34], shown in Figure 2.3. This junction is composed of two ferromagnetic (FM) layers separated by an oxide barrier, e.g. MgO [34]. One of the layers, the reference layer, has fixed mag-

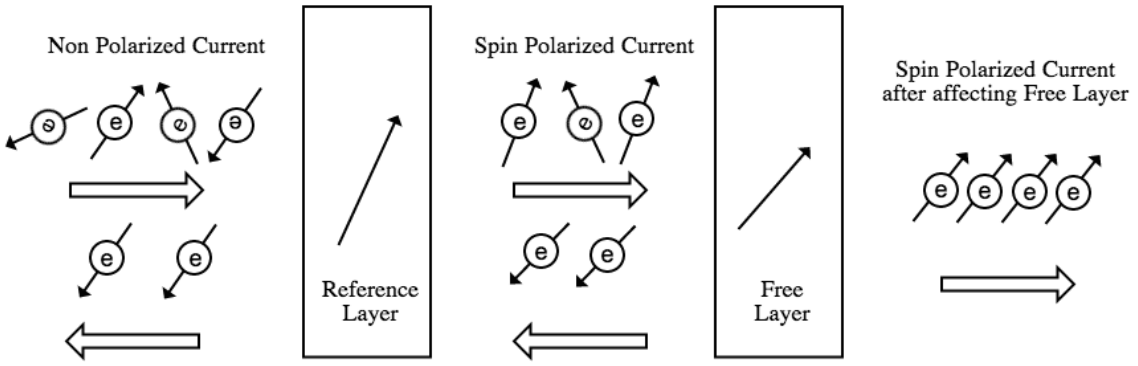


Figure 2.4: Writing a 0 into STT-MRAM cell (parallel, low resistance)

netization orientation. The other layer can be configured in a controlled manner. In conventional MRAM, writing process is performed through Field Induced Magnetic Switching (FIMS), i.e. the magnetization direction is switched by a current induced magnetic field. Due to the tunneling magnetoresistive effect, the resistance of an MTJ can be determined by the relative magnetization orientations of two different FM layers. If the direction of the free FM layer is the same as that of the reference layer, which means that they are parallel, the resistance is low and the data stored in the MTJ is *0*. If not, the storage value is opposite.

The storage technique for STT-MRAM is slightly different from a conventional MRAM. Instead of configuring the direction of a current, there is a wire which directly passes through the MTJ. The free layer can be manipulated by controlling the direction of the current flowing on that wire.

The operating theory of STT-MRAM is depicted as shown in Fig. 2.4 and Fig. 2.5. While writing a '0' into the cell, non-polarized current passes through the reference layer. The magnetic field in the reference layer spin-polarize the electrons so that only part of them with a specific direction could go through and in the free layer, they determine the magnetic field. Similarly, while writing an '1' into the STT-MRAM cell, the unpolarized current needs to be injected from the free layer side. Although at first the magnetic characteristic of the free layer is unpredictable (the dotted arrow

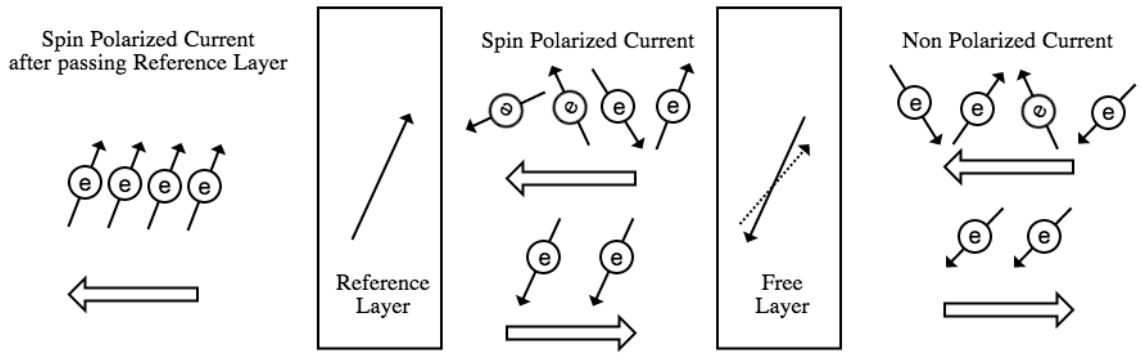


Figure 2.5: Writing an 1 into STT-MRAM cell (anti-parallel, high resistance), dotted line in Free Layer presents the original direction in that layer

assumes one of the possibilities), when the current hits the reference layer, a polarized current will be bounced back, as shown in Fig. 2.5, so that the magnetic direction in the free layer could be determined.

Table 2.2: Comparison of Different RAM Techniques [57]

Memory Tech	MRAM	SRAM	DRAM	Flash
Read Speed	Fast	Fastest	Medium	Fast
Write Speed	Fast	Fastest	Medium	Low
Area Efficiency	Med/High	Med	High	Med/Low
Scalability	Good	Good	Limited	Limited
Cell Density	Med/High	Low	High	Medium
Cell Leakage	Low	Medium	High	Low

MRAM can be fabricated using existing metalization techniques. MRAM needs 3 additional layers than CMOS logic [58] and 3-4 fewer than embedded eFLASH memory [40]. A comparison among MRAM, DRAM, SRAM and Flash is shown in Table 2.2.

2.2.2 MRAM Interaction with CGRA

MRAM memory system could either be centralized or distributed in CGRAs across an array of PEs. Since MRAM stores data in the resistance domain, extra circuitry must be implemented to convert data from resistance to a voltage level. Previous

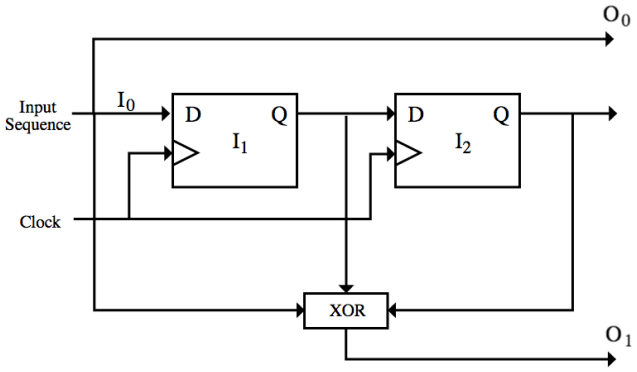
research [41] has indicated that MRAM and associated circuitry will benefit from a centralized design style for FPGAs since fault tolerance techniques can be readily employed. However, such a limitation is not applicable to CGRAs. A configuration memory in a CGRA needs to store multiple sets of configuration bits (different sets of instructions), whose size ranges from 10 kB to 100 kB. Part of configuration memory may also be used for general data storage. Therefore, configuration memory for each PE could be fairly large. To guarantee system reliability, a partially-centralized design style will been proposed. The reconfigurable computation fabric will be partitioned into different regions. Each region shares a block of MRAM. In this thesis work, we simplify the architecture and assume that each PE has a private block of MRAM. The block stores configuration contexts and signal processing data.

In each PE, as shown in Figure 2.1, both a large MRAM (Configuration Memory) and smaller SRAM (Instruction Memory) are needed to guarantee performance. The SRAM is directly connected to the ALU and other configurable points. The SRAM can be quickly configured from the MRAM during configuration changes.

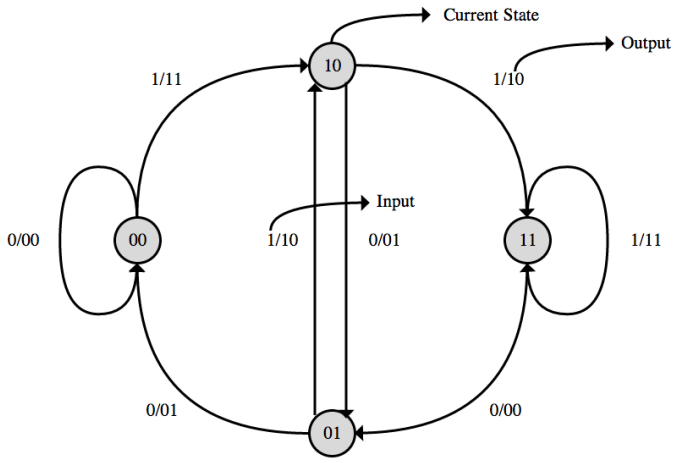
2.3 Case Study of Energy Performance Tradeoffs from System-Level Reconfiguration

To better understand the energy performance tradeoffs of the dynamic reconfiguration of a CGRA, three codec algorithms will be mapped onto our prototype architecture (Viterbi, AVA and RS algorithms). Although they use very different coding theories at the algorithm level (Viterbi and AVA are convolutional codes while RS is a block code), their computational complexities can benefit from reconfiguration. In this section, the benefits of adaptive software and hardware implementations for all three benchmarks are explored.

To further understand the benefits of reconfiguring with STT-MRAM, we increase the testbench with a motion estimation algorithm and an FIR filter. The former



(a) Circuit Model



(b) State Diagram for the convolutional encoder in Figure 2.6a

Figure 2.6: A (2, 1, 3) convolutional encoder [48]

requires high-throughput data processing and huge amount of data flow. The latter, on the other hand, is computationally intensive which could be potentially accelerated by a parallel array.

2.3.1 Viterbi Algorithm

The Viterbi algorithm has been proposed as a decoding algorithm for convolutional codes over noisy wireless communication channels. It has found applications in digital cellular, satellite, and deep-space communication and 802.11 wireless LANs. In general, a Viterbi code is an error correction code which is capable of detecting and correcting data transmission errors in wireless channels [56]. To support error tolerance, multiple redundant bits must be added to the original message to provide extra information for the decoder to handle the possible corruption of message bits.

A Viterbi encoder, which is identical to conventional convolutional encoder, generates output not only depending on the data bit received in a time point, but also on data received within a previous span of *K-1* time points, where K is a *constraint*

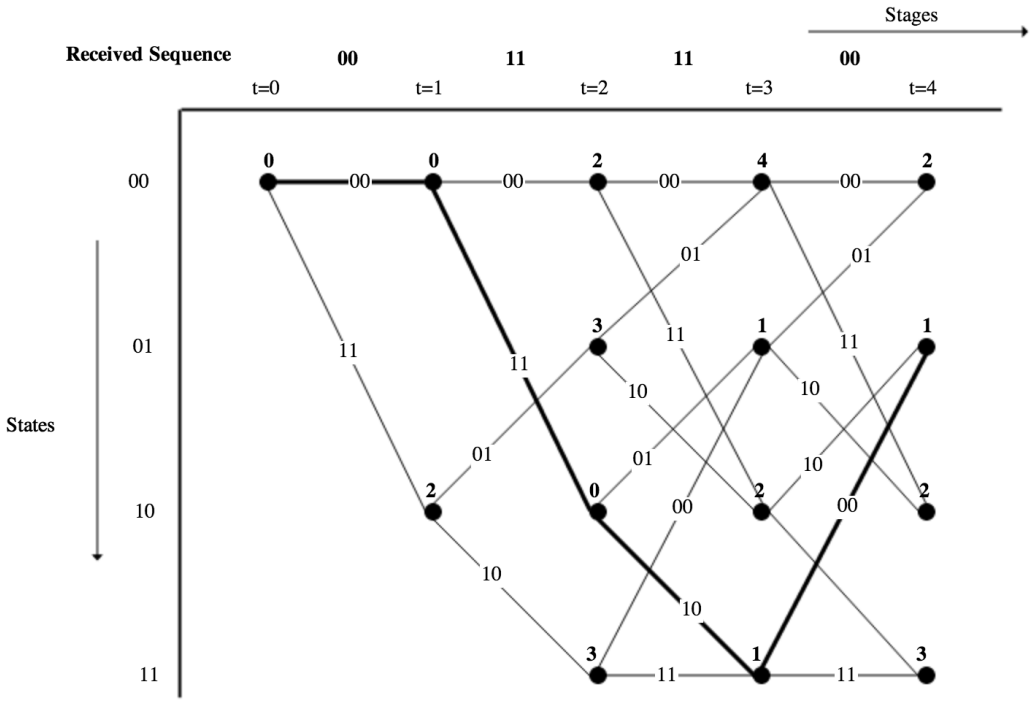


Figure 2.7: *Trellis Diagram* for the Convolutional Encoder in Figure 2.6 [48]

length which must be larger than 1. Encoding is accomplished by multiple shift registers and XOR function units. Characteristics of a convolutional encoder are determined by three factors: the number of output bits generated when receives an input symbol (v) is received, the number of input bits at a time (b) and the constraint length (K). Together, the information is represented as (v, b, K). Figure 2.6a [48] shows an example of a (2, 1, 3) conventional encoder. The encoding process can be represented by a state diagram, as shown in Figure 2.6b [48]. Each node represents the present state and edges are attached based on the input bit and the associated output sequence. For example, when the current state is a 00, if the next incoming bit is a 0, the state machine will output 00 and the current state will be updated to 00. If the input bit is a 1, the state machine will output a 11 and the current state will be updated to 10.

A decoder tries to reconstruct the original input sequence from the channel output, which might not be the same as the input sequence due to channel noise. A *trellis*

diagram (shown in Figure 2.7 [48]) is used to find the most likely transmitted data sequence. Its horizontal axis represents received v-bit symbols from the channel output in time order. The vertical axis represents present states corresponding to the state diagram. Each node in the diagram represents a probable current state based on the input symbol. Edges represent state changes after receiving a symbol. In the Viterbi algorithm, each *trellis* node produces two output edges because the incoming bits could be either 0 or 1. In Figure 2.7, the digit associated with an edge is the output of encoder, which could be found in the state diagram in Figure 2.6b. At each node, a *Hamming Distance* between the received symbol and an *expected* symbol is computed (not shown in the figure). An *expected* symbol is an output of the encoder. For example, if current state is 00, next state could either be 00 or 10, which is dependent on the next input bit to the encoder. Associated encoder output is 00 and 11. In Figure 2.7, the first incoming symbol is 00. For state 00, Hamming distance (simplified as *cost*) is 0, for 10, it is 2. The cost of each node accumulates after transiting from one state to the other (shown as the number next to each node). A large cost indicates a low probability that the *trellis* node would be part of the optimal path with lowest global cost. When two edges converge, the lower accumulated cost will be assigned to the converging point. An optimal path is found by connecting nodes with lowest costs along existing edges (The bold path in Figure 2.7). The following tersm allow for a more precise definition:

- *Path Metric* - The accumulative cost at each node
- *Branch Metric* - Error metric associated with a *trellis* branch
- *Truncation Length* - A span of time steps in which the lowest cost path is found

Generally, a larger constraint length K generates more states in a state diagram, hence a more complicated *trellis diagram*. [20] indicates that larger K provides greater

resilience to bit errors but at the cost of computing effort. By adapting K to channel quality, an improved energy consumption could be achieved.

2.3.2 Adaptive Viterbi Algorithm

The Adaptive Viterbi algorithm [9] has been proposed to reduce the average computation and *trellis* stage storage required by the Viterbi algorithm. Instead of retaining all 2^{K-1} stages for each received channel output symbol, only those that satisfy some cost conditions could be stored in a *survivor memory*. Intuitively, we could discard trellis nodes with high path metric before storing them. To quantify such intuition, previous research [9, 45] proposed the following criteria:

1. For each node, there is a threshold T that determines whether it could be stored in the *survivor memory*. If path metric is larger than $d_m + T$, it will be eliminated due to high cost, where d_m represents the minimum cost among all survived paths in the previous *trellis stage*.

2. The total number of surviving paths in a *trellis stage* is pre-fixed to be N_{max}.

While decoding, the first criterion filters out paths having a high distance from expected sequence. Since the remaining paths may have similar cost, the second criterion is used to control the total number of survivor paths. Figure 2.8 [48] shows the decoding process of the adaptive Viterbi algorithm, which has the same input symbols as shown in Figure 2.7. The bold path indicates the output stream. Values of T and N_{max} needs to be carefully determined. If T is too small, few paths will be stored in *Survivor Memory*, which could result in an increase in bit error ratio (BER). However, less memory and computing effort is required. N_{max} has a similar effect on BER as T. In general, the adaptive Viterbi algorithm can adapt itself to different channel quality by varying T and N_{max}. If SNR becomes higher, we could select a combination of smaller T and N_{max} and still obtain a target BER. Otherwise, more memory bits and computing effort are required.

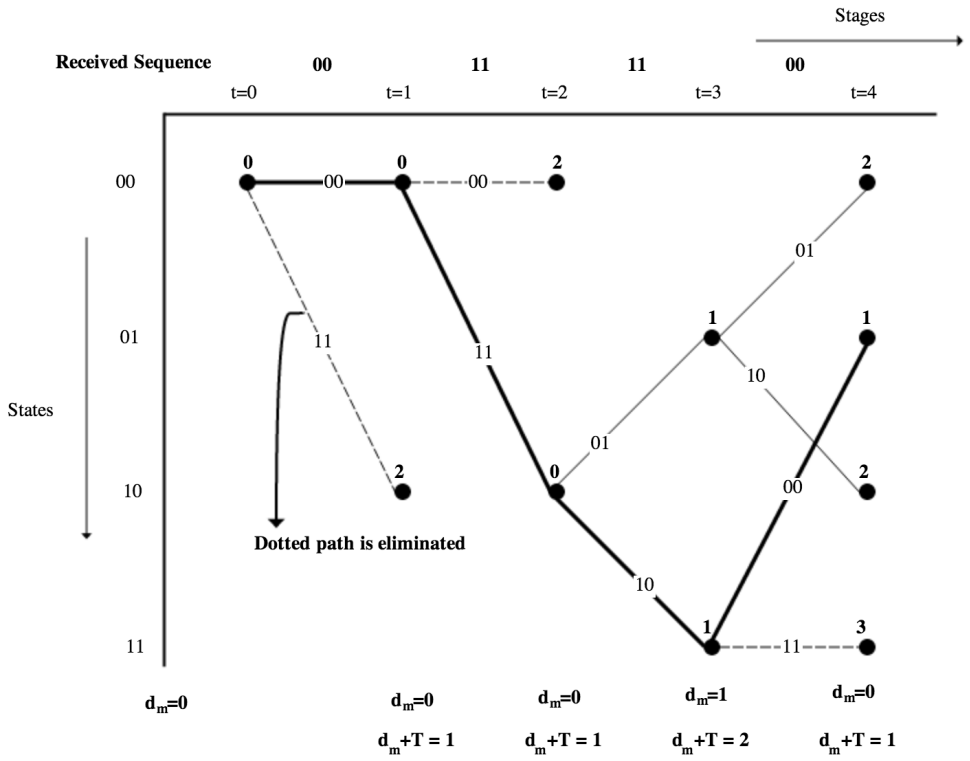


Figure 2.8: *Trellis Diagram* for AVA decoding with $T = 1$ and $N_{max} = 3$ [48]

In the algorithm implementation, reducing the number of survived paths to N_{max} is not as trivial as pruning paths based on path metric. Intuitively, the best way to remove paths is to sort all survived paths based on path metrics and select the first N_{max} paths. However, the sorting algorithm is at least $O(n \log n)$, which is too computationally intensive for embedded signal processing systems. To address this problem, reference [48] has proposed an approach of adapting threshold T to the number of survived paths. If the total number is less than N_{max}, no update is required. If the number of survived paths exceeds N_{max} in a *trellis* stage, the threshold is decreased by 1, which is adjustable. With a decreased T, the number of survived paths will decrease until the survivor count is equal to or less than N_{max}. Reference [48] indicates that if we select the original T carefully, only a couple of T adjustment loops are needed.

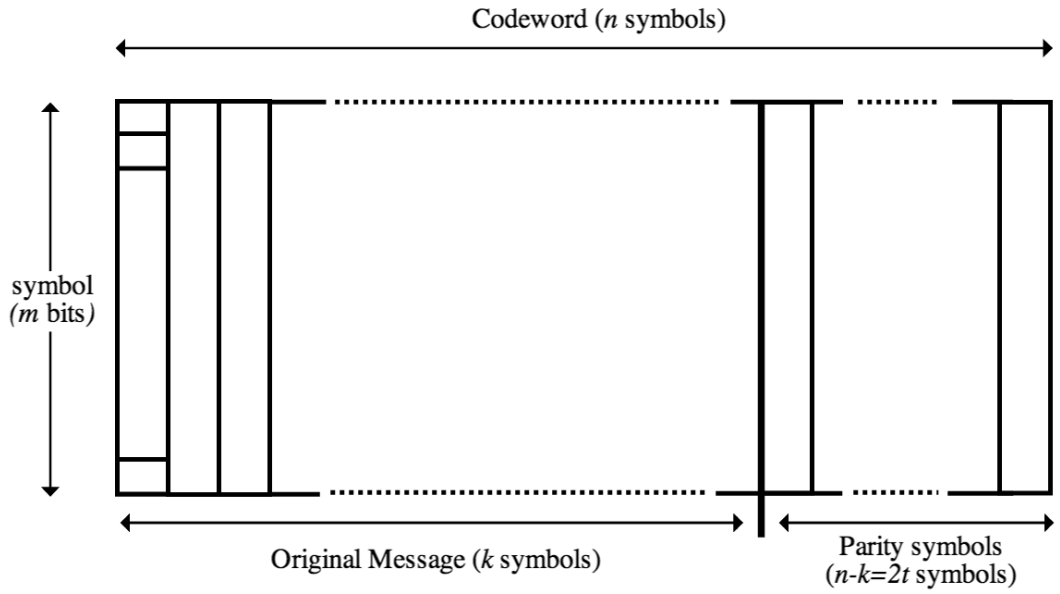


Figure 2.9: Reed Solomon Code definitions [13]

2.3.3 Reed-Solomon Algorithm

Reed-Solomon (RS) codes are non-binary linear block codes [15], which can detect and correct a limited number of errors that occur anywhere in an erroneous message with the help of redundant symbols in a *codeword*. Figure 2.9 shows the definition of a Reed-Solomon code [13]. It could be described as a $(n,\ k)$ code, where n represents block length in symbols, k is the length of original message in symbol, m is the number of bits in a single symbol, and t is the number of symbol errors being able to be corrected by a receiver in a *codeword*. The relationship between n and m is

$$n \leq 2^m - 1.$$

The number of correctable errors is

$$t = \frac{n-k}{2}, \text{for } n-k \text{ even}$$

or

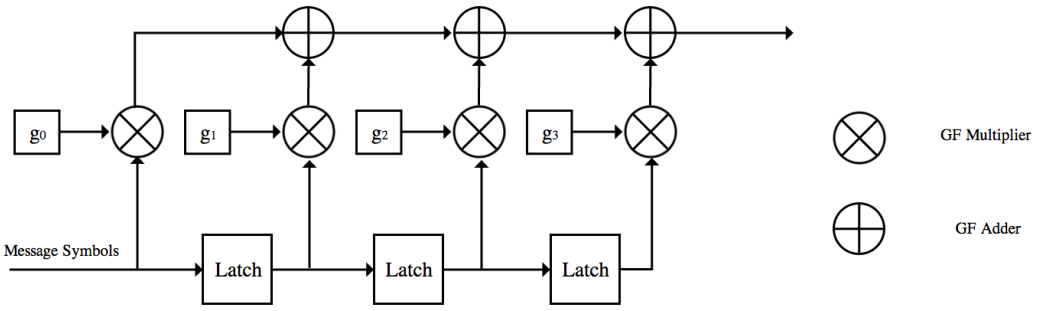


Figure 2.10: A Brief Overview of Reed-Solomon Encoder [49]

$$t = \frac{n-k-1}{2}, \text{for } n-k \text{ odd.}$$

RS codes are designed to handle both random and burst bit errors. Since a consecutive bit errors can affect at most $\lceil \frac{a}{m} \rceil$ symbols, if t is large enough, it can handle an error burst. For the random-error case, b errors will affect at most b symbols. Therefore, an appropriate t could handle both burst and random error scenarios.

The RS algorithm works in *Galois Fields* (GF) [15] in which there are 2^m elements. The RS algorithm adds *n-k* parity symbols to a codeword to supplement the original message. This information provides redundancy for the encoded message symbols. All parity symbols are computed through multiple pipelines of GF addition and multiplication. Figure 2.10 shows the encoding data flow [49]. These operations are performed using coefficients (g_0, g_1, g_2, g_3 in the figure) of a *GF polynomial*, defined as generator polynomials, which is predefined based on the number of parity symbols. The complexity of encoding (and decoding) is dependent on *code rate* (the non-redundant portion of the codeword). A lower value of k requires more computational effort to process parity symbols. A detailed encoding example is provided in [1].

The RS decoding algorithm represents the received word as coefficients of a polynomial $r(x)$. A code generator polynomial can be represented as $g(x)$. The original message polynomial is defined as $p(x) = \frac{r(x)}{g(x)}$. Considering that the received word

could be corrupted by channel noise, an error polynomial $e(x)$ needs to be added. The relationship among polynomials is

$$r(x) = p(x) \cdot g(x) + e(x).$$

Polynomial division is performed on every received symbol. If the remainder $e(x)$ is not zero, an error has occurred during data transfer. While locating errors, the error vector $e(x)$ could be determined from the received codeword polynomial. Hence, by adding the error vector back to the product of message and generator polynomial, a corrected received word can be obtained.

Since the code rate of the RS algorithm can be adjusted to adapt to physical environments, it is applicable to a runtime reconfigurable architecture. Our experiments have shown that high-code-rate configurations consume less energy but at the cost of decoding accuracy. When the original message size k is large, the number of parity symbols t will be small. By dynamically tuning k, we can achieve reduced computational complexity while maintaining an acceptable codeword error rate (CER).

2.3.4 Three-Step-Search Motion Estimation

Three-Step-Search is a motion estimation algorithm. It is simple, robust and provides near optimal performance. The algorithm first searches in a coarse pattern and converges into a fine one. It works as follows:

1. An initial step size is picked. Eight blocks at a distance of the step size from the center point are selected for comparison.

2. The step size is cut by half and the center point moves to one of the eight blocks with the least distortion.

3. Keep doing steps 1 and 2 until the step size reaches one.

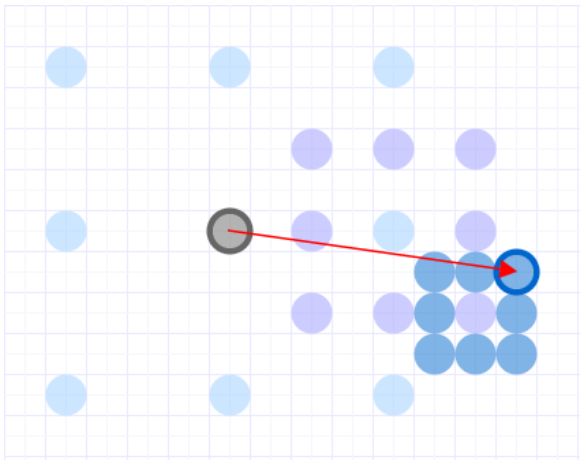


Figure 2.11: Convergence of Three-Step-Search

A brief example of convergence of the algorithm is shown in Fig. 2.11. For this application, a series of 1024×1024 pixel greyscale images are distributed to a CGRA array for testing. Each of these images is part of a sequence in which motion can be detected. Subsequent images in the sequence are split into windows and each window is tested. In our experiment, the motion estimation algorithm will receive two images simultaneously (one old image and one new image), each image is segmented into hundreds of windows.

In this implementation, the accuracy of the motion estimation algorithm is dependent on the window size. For higher accuracy, a smaller window size is used, leading to more windows and increased computation. For reduced accuracy, the opposite effect is observed. In our experimentation, we consider square window sizes of 14, 16, 18, 20, 24 and 26 pixels on a side.

2.3.5 Finite Impulse Response (FIR) Filter

An FIR filter is a common multiply-accumulation based DSP algorithm. The accuracy and quality of the filtered output is based on the number of coefficients the algorithm has. In the thesis, the FIR filter is configured as a band-pass filter. The frequency of the band ranges from 0.35 to 0.65. Taps (coefficients) are generated from

MATLAB with the `fir1` command. This application involves the implementation of a filter with tap counts ranging from 120 to 1920 taps.

2.4 System Adaption to a Changing Environment

Changes in signal processing algorithms based on noise and other environmental factors need to be carefully evaluated to allow for associated energy-saving dynamic hardware changes. For example, in future wireless communication devices, three industrial trends motivate the development of a reconfigurable platform:

1. Modulation and codec algorithms often adapt to the quality of radio channels to maximize throughput. For modulation, a system can automatically select the simplest technique to save energy as long as it provides an acceptable bit error rate (BER). Codec algorithms, such as adaptive Viterbi algorithm (AVA) [9], can adapt their computational complexity to a varying signal-to-noise ratio (SNR). For example, two parameters of AVA dynamically determine the computing workload: threshold T and number of survivors N_{max}. A larger T retains more trellis nodes, hence accuracy is achieved at the cost of computing effort. Similar to T, a large N_{max} gives better BER but sacrifices energy. By dynamically changing both parameters, we can adapt the system to different channel qualities. One of our goals is to rapidly configure T and N_{max} to achieve reduced energy consumption at an acceptable decoding performance. Other applications for audio and video experience similar characteristics.

2. We expect that a signal processing chip is robust enough to make the above tradeoffs. Signal processing chipsets must support a wide range of applications and deliver tradeoffs for performance and energy. For example, if a back-end system requires a byte stream instead of a bit stream, the adaptive system should be able to switch from AVA (outputs bit stream) to a Reed-Solomon (RS)

algorithm, which can naturally support byte-level codec and error correction. Additionally, if a mobile device enters an environment subject to burst errors (i.e. multiple bit errors happen in sequence), we expect the system could switch from a convolutional code (e.g. AVA) to a block code (RS), since the latter algorithm could handle such a scenario [4]. With high-density MRAM, more application configurations can be located close to a PE array, allowing for fast reconfiguration.

3. Hardware resources in signal processing devices are not always fully utilized due to time-varying computational complexity of the adaptive digital signal processing algorithms. Typically, hardware is designed to achieve acceptable performance in a worst-case physical environment. If channel quality gimproves, the computing power offered by a platform will be wasted because workload is no longer intensive. If the platform is able to adapt itself to the operating environment, unnecessary hardware resources can be turned off. Our experimentation discussed in the next chapter shows that there is a 31.3% energy difference between architectural adaption and non-adaption to channel quality (SNR increases from 2.45 dB to 2.85 dB, throughput is constant). Therefore, we expect that a CGRA-based processing platform could be dynamically reconfigured to adapt hardware infrastructure to the target computing task.

In the next chapter we examine these tradeoffs for a multicore system prototyped in an FPGA.

CHAPTER 3

PRELIMINARY EXPERIMENTATION

In this chapter, we discuss energy tradeoffs achieved by varying algorithm implementations in CGRAs based on environmental factors such as communication channel noise. To generate preliminary data, we have prototyped a mockup of a CGRA in an FPGA to explore energy consumption and performance tradeoffs. Three codec algorithms (Adaptive Viterbi, Viterbi, and Reed-Solomon) have been manually mapped to our prototype CGRA. By studying results from preliminary experiments, we conclude that adapting an algorithm implementation with changing environment parameters in a CGRA can yield energy savings. For AVA, a 0.4 dB improvement in channel quality results in 31.3% energy savings. For Viterbi, a 0.1 dB improvement gives 85.7%. For RS, the number is 20.3% with 1 dB increase in SNR.

3.1 Energy Benefits Exploration of a Dynamic CGRA

In a typical CGRA, each PE includes an ALU, peripheral control circuitry and memory blocks. The PE program and the contents of memory blocks can be reconfigured to adapt to different computing tasks either across applications or within a time-varying application. In order to examine the benefits of reconfiguration in terms of energy and performance, we implemented several CGRAs using pre-synthesized soft processors. An Altera NIOS II soft RISC processor is used to implement a PE. The system is prototyped using a Stratix IV FPGA. Our initial experiments do not dynamically reconfigure processor functionality at runtime, although experiments with several different parameter sets help determine the likely benefit of dynamic recon-

figuration. As part of the continuing thesis work, dynamic reconfiguration based on channel noise will be considered.

Case studies in the previous chapter indicate the possible benefits of adapting communication decoder parameters during runtime to different SNRs (e.g. T and N_{max} of AVA). To quantify such energy savings, we compile multiple versions of decoder applications with different parameter sets and evaluate them for performance and energy. Each parameter set targets a certain channel noise scenario: low, medium or high SNR. A low SNR is associated with a parameter set requiring a high computational complexity for the decode algorithm, a high SNR presents the opposite case. In our experimentation, all decoders provide the same Bit Error Rate (BER) in simulation and after mapping to our FPGA-based CGRA prototype. In our experimentation we show that when SNR changes from low to high (channel quality gets better), energy savings can be obtained by switching from an executable whose parameter set yields high computational complexity to a lower one. In this way, we explore the energy benefits from algorithm level adaptation. For a set of decoders, although BER is constant, a parameter set with high computational complexity and the same set of compute resources can lead to reduced throughput.

To maintain a constant throughput for an application with a fixed BER, hardware resources (PEs) must be added to the computing system. To accurately determine the number of required PEs, simulations can be performed. The experimentation in this chapter explores appropriate hardware configurations of PEs to maintain a pre-specified BER and decode throughput. The energy consumption of these configurations is then explored. Ultimately, these configurations could be swapped in real time to reduce energy costs as channel noise changes. This goal will be explored in later chapters.

3.2 Experimental Approach

To determine the energy efficiency of a dynamically reconfigurable CGRA, energy, power and execution time of different configurations need to be measured. Also, code size, instruction count and performance (*throughput* for codec applications) are quantified. They are computed using the following methodologies:

- *Power:* Since our prototype is implemented in a Stratix IV FPGA, a software tool which uses simulation information, Altera PowerPlay [11], is used for power estimation. Generally, PowerPlay estimates power consumption using signal toggle rates and the static probability of I/O signals. Frequent toggles lead to increased dynamic power consumption. Our prototypes are designed using Qsys, a block-based design tool from Altera. All components, including NIOS processors and memory blocks, are built in Qsys and only CLK and RESET are used as external I/O pins. Since RESET is tied to ground, CLK is the only dynamic I/O in our design. Our DE4 board also provides on-chip current and voltage drop sensors connected to FPGA via SPI bus. Run-time power numbers could be obtained by reading those ADC's using NIOS processor.

- *Execution Time:* The Altera IP core library includes a hardware counter which can be used in circuit to measure the execution time of a circuit component in clock cycles [3]. We included it in our prototype to measure the time span to decode a given volume of input bits.

- *Energy:* Energy is determined as the product of time and power, as determined above.

- *Decoder Throughput:* Throughput is calculated by dividing the number of output bits by execution time.

Table 3.1: Parameter sets that determine computational complexity

BER=10^{-4}	Viterbi		Adaptive Viterbi		Reed-Solomon	
Channel Quality	SNR (dB)		SNR (dB)		SNR(dB)	
Low	2.70	$K = 6$	2.85	$K = 11$ $N_{max} = 25$ $T = 23$	17.0	$t = 12$ $n = 255$ $k = 231$
Medium	2.65	$K = 7$	2.65	$K = 13$ $N_{max} = 26$ $T = 24$	16.5	$t = 14$ $n = 255$ $k = 227$
High	2.60	$K = 8$	2.45	$K = 14$ $N_{max} = 41$ $T = 24$	16.0	$t = 16$ $n = 255$ $k = 223$

- *Code Size and Instruction Count:* Instruction count is determine after program compilation with a MIPS GCC compiler. Code size in bytes is four times the instruction count since each instruction requires four bytes.

To estimate the behavior of a dynamically reconfigurable system, multiple static hardware configurations are pre-synthesized to our multiprocessor system. For these experiments, we pre-synthesized six multiprocessor configurations, which include processor counts of between 1 and 6. The details of the hardware-level configuration are provided in the next section. Our energy saving approach considers that some processors can be shut off depending on channel noise conditions for a fixed bit error rate (BER).

To determine a corresponding parameter set for each SNR, we simulated algorithm behavior on a workstation. Previous work [48] and [4] have researched the impact of algorithm parameter sets on AVA and RS performance. Based on this work, we generated parameter sets for our experimentation, which are organized in Table 3.1. A 10^{-4} BER for VA and AVA and a 10^{-4} codeword error rate (CER) for RS are targeted (based on simulation of one million input symbols). SNR selections for low, medium and high channel quality are based on available data from previous research. In [48], SNR values from 2.5 dB to 6.5 dB are evaluated for AVA. Since AVA is a modified version of the Viterbi algorithm, we select SNR numbers from the same

Table 3.2: Hardware configurations which result in the same throughput

	Low SNR	Medium SNR	High SNR	Throughput
PE Count (VA)	6	5	3	≈ 1.65 $Kbps$
PE Count (AVA)	6	2	1	≈ 0.17 $Kbps$
PE Count (RS)	6	5	4	≈ 26.91 $Kbps$

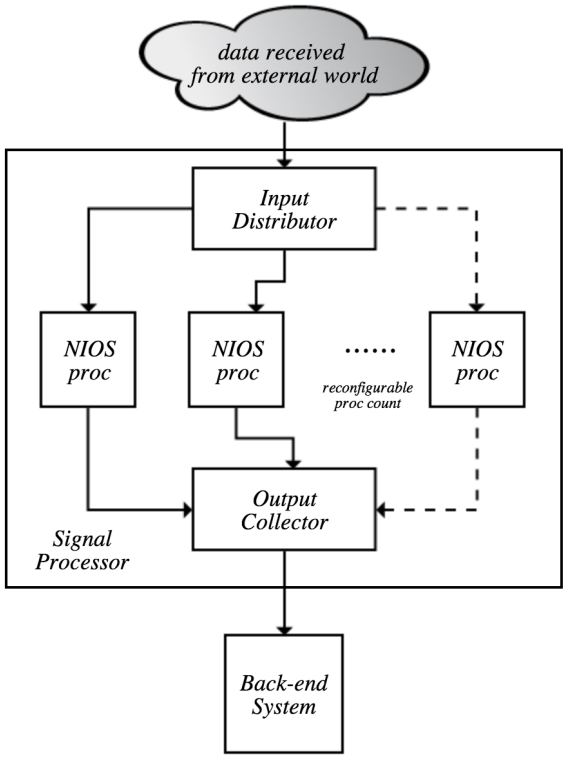


Figure 3.1: Architectural Overview of a Prototyped CGRA in a Stratix IV FPGA Using NIOS II Soft Processors

range for VA. For previous RS experimentation [4], SNR ranged from 13.6 dB to 20.0 dB. We define our configuration similarly.

Table 3.1 indicates parameter values for a fixed BER. The number of PEs needed to achieve a given throughput are indicated in the table.

3.3 Experimental Platform

Our prototype is implemented as shown in Figure 3.1. The prototyped system receives a continuous data stream, parallelizes the decoding workload, executes de-

Figure 3.2: Inter-NIOS Communication via Shared Memory

coding algorithms, and then collects results. The data distributor, collector and all parallel decoders are implemented using NIOS-based soft processors. The distributor packetize streaming data into packets before sending them to available parallel NIOS II decoders. When computation is finished, the collector retrieves the results in the correct order and forwards them to a potential back-end system. All NIOS II decoders simultaneously execute the same source code but on different portions of incoming data, leading to parallelism.

Data distribution in Figure 3.1 is implemented based on codec type. For convolutional codes, such as Viterbi and AVA, input bits arrive in a continuous fashion. In our experimentation, each data packet contains 500 streaming bits. Each convolutional decoder will be assigned a data packet. For block codes, such as the Reed-Solomon code, data is transmitted in blocks and there is no data dependencies among blocks. Therefore, block codes naturally support decoding parallelization. In our experimentation, each RS decoder is assigned codewords. The length (in symbols, each symbol contains 8 bits) of the codeword is 255. The function of data collection is to concate-

nate outputs from different decoders. The order of collection must be the same as the order of distribution.

All blocks in Figure 3.1 are connected by Avalon Memory Mapped (AMM) buses, a hardware IP designed by Altera. This interface provides read and write capabilities for master and slave components in a memory-mapped implementation. Communication between blocks is realized by blocks of shared memories (SHM), whose architecture is shown in Figure 3.2. Through software-level flow control technique, one NIOS II PE can transfer data to another PE by writing to/reading from it. In Figure 3.1, all connections between the distributor and PEs and between PEs and the collector are made using these shared memory interfaces.

In our prototype, each PE is unable to directly communicate with all other PEs. For the complete thesis work, one of our goals is to replace the trivial on-chip interconnection with a time-scheduled NoC so that both streaming and parallel datapaths can be better implemented.

Figure 3.3 shows a complete implementation of our experimental platform in an Altera FPGA. To feed our prototyped CGRA decoding system with input data, an on-chip input generator is implemented. It uses the fastest NIOS II soft processor whose clock frequency is set to be 200 MHz, which is four times higher than the other NIOS PEs in our system. Note that this is the highest clock frequency that guarantees reliable behavior of NIOS II soft processors [2]. Since the available clock from the FPGA board is only 50 MHz, we used a phase locked loop (PLL) to generate the 200 MHz clock. The worst-case throughput of the input generator is 3.2 Kbps for convolutional codes and 83.3 Kbps for block codes (Worst-case refers to encoding with the lowest code rate). During experimentation, decoding is limited to this rate.

The input generator is shown in the grey block in the figure. It generates simulated channel outputs. Random numbers are generated to represent the original data message. The numbers are then encoded based on the target application. We did not

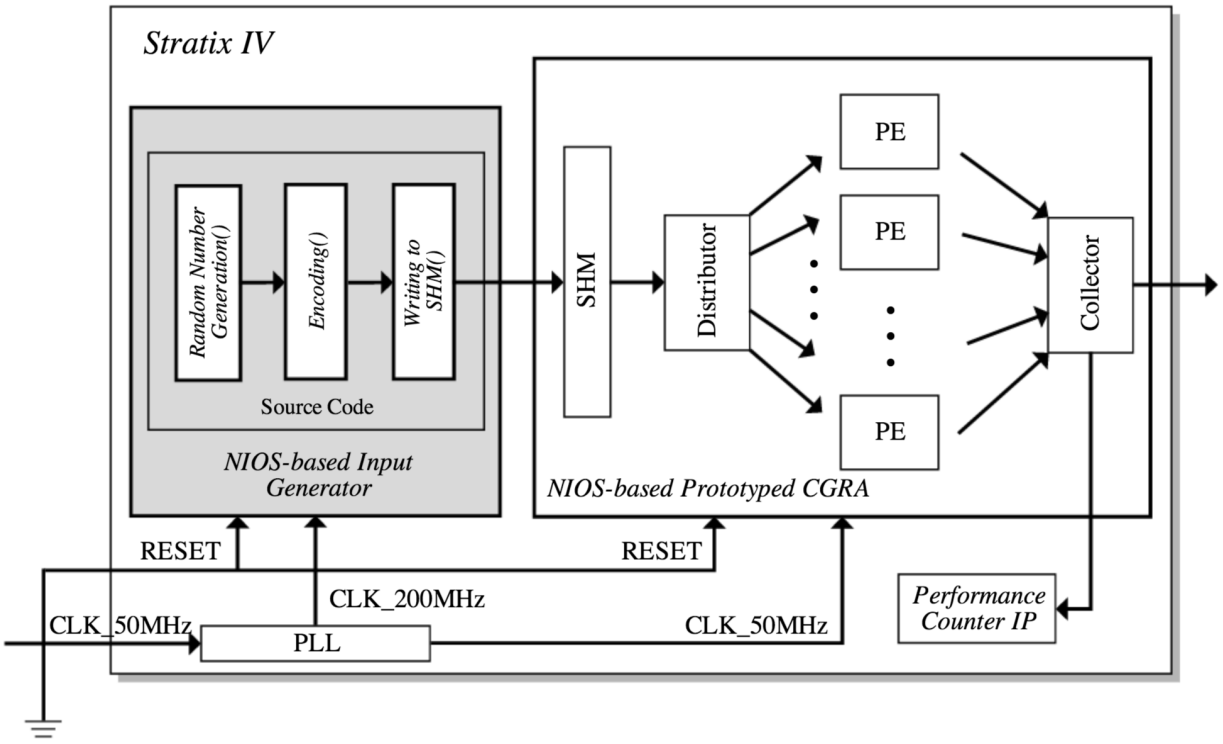


Figure 3.3: FPGA implementation of the complete experimental platform

include modulation and noise-based data corruption in the data generator because these parameters can be pre-set in building the IP core. A performance counter is implemented to measure execution time for throughput calculation.

3.4 Results and Analysis

Energy and throughput results are depicted in Figures 3.4 through 3.9. Figure 3.4, 3.6 and 3.8 show that throughput increases linearly with PE count, which is to be expected due to a lack of communication congestion. The PE counts in the graphs do not include the *Distributor* and *Collector* shown in Figure 3.1). Since each PE executes on different portions of input data stream, there is no data dependancies among PEs and all decoders operate in parallel. The data transfer overhead reduces overall throughput a bit, however, it is negligible compared to the decoding time in our experiments of tens to hundreds of seconds.

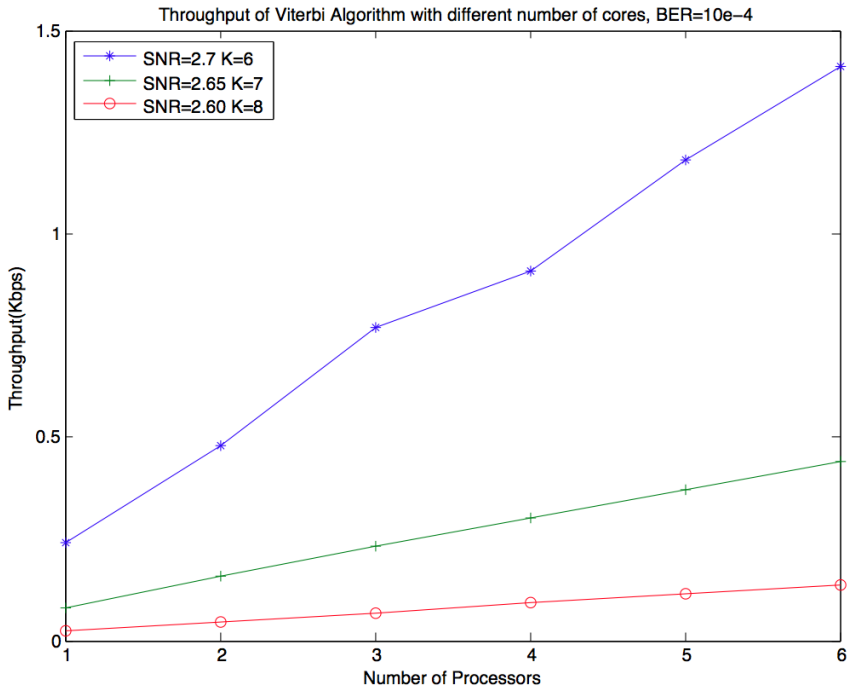


Figure 3.4: Throughput of Viterbi algorithm with varying PE count

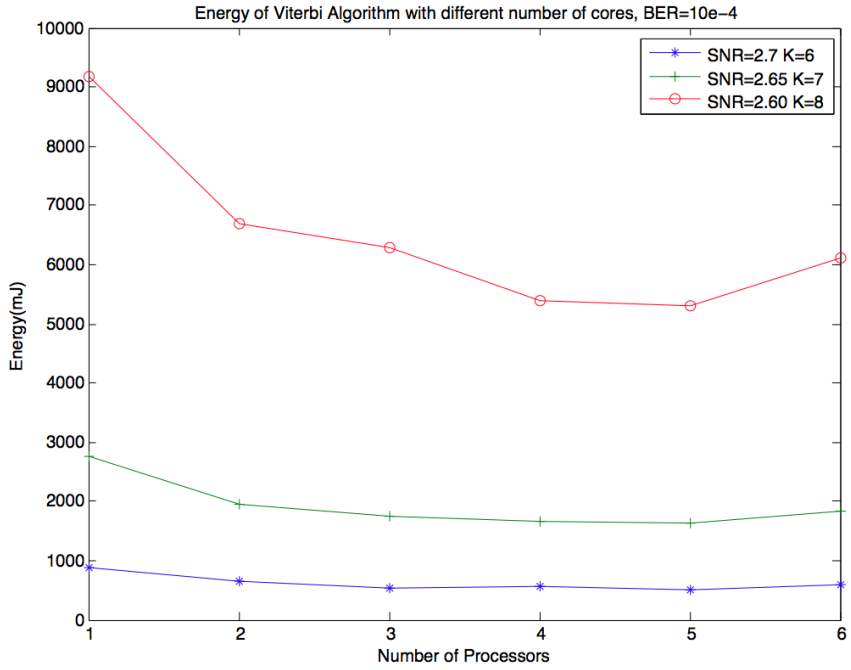


Figure 3.5: Energy of Viterbi algorithm with varying PE count

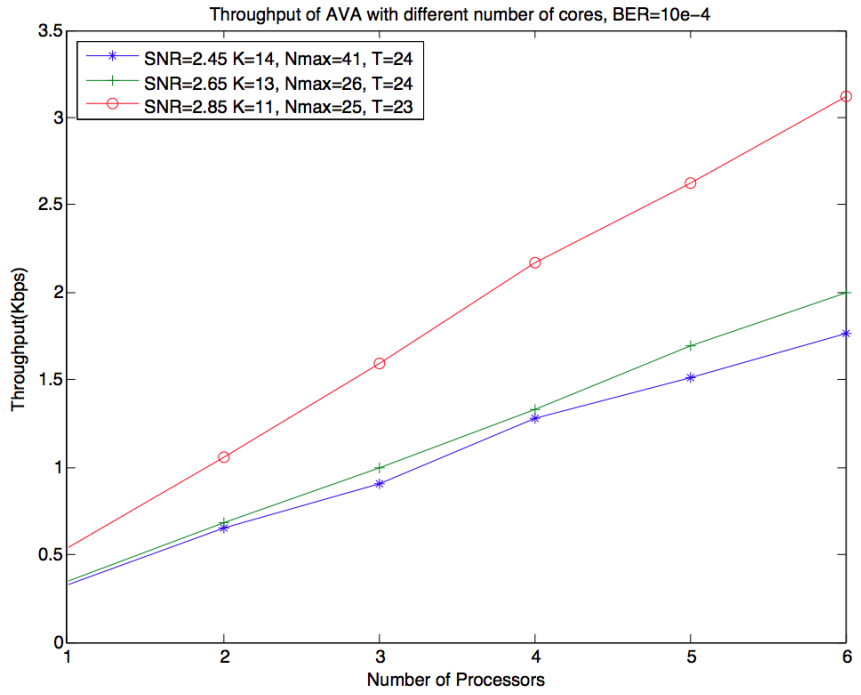


Figure 3.6: Throughput of adaptive Viterbi algorithm with varying PE count

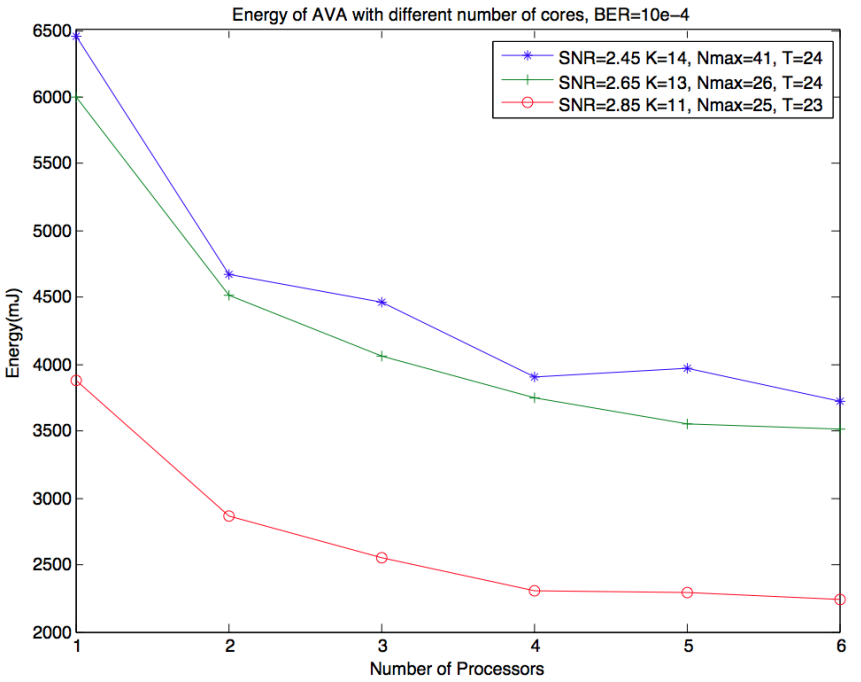


Figure 3.7: Energy of adaptive Viterbi algorithm with varying PE count

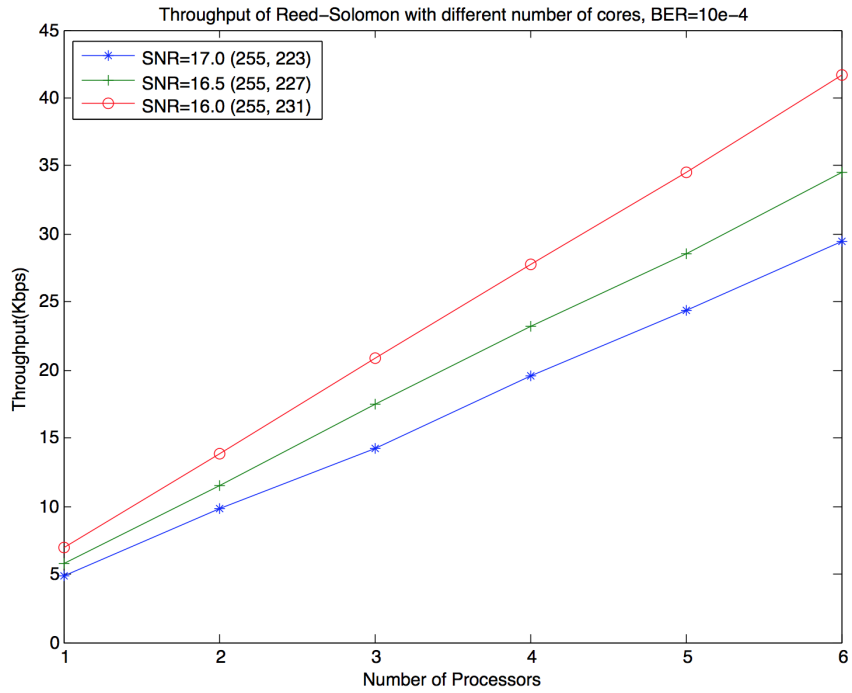


Figure 3.8: Throughput of Reed-Solomon algorithm with varying PE count

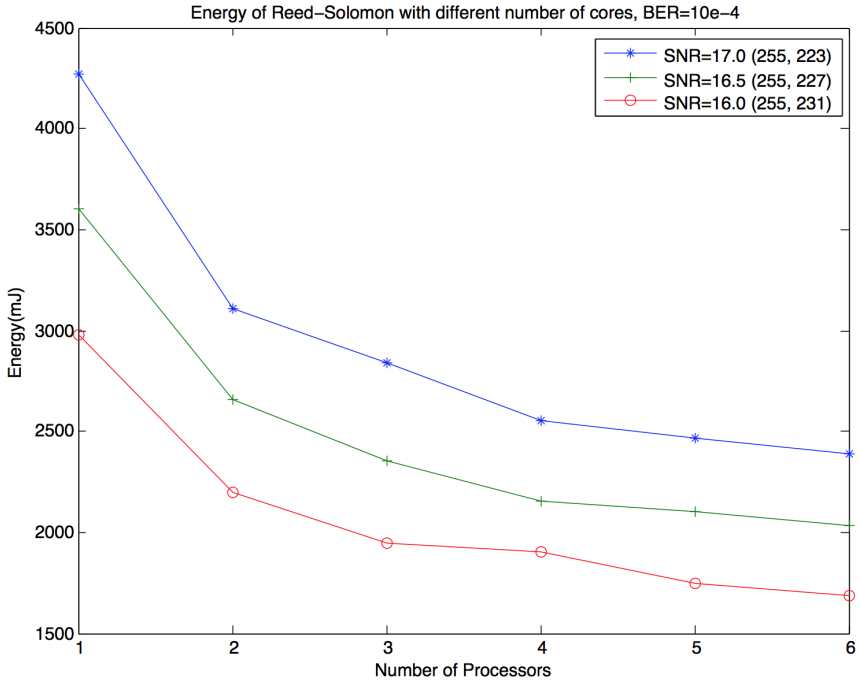


Figure 3.9: Energy of Reed-Solomon algorithm with varying PE count

Table 3.3: Energy Benefits of Adaptive Algorithm Implementation For Constant Throughput and Variable SNR

BER=10^{-4}	Viterbi		Adaptive Viterbi		Reed-Solomon	
Channel Quality	energy (mJ)	savings	energy (mJ)	savings	energy (mJ)	savings
Low	6108.3	0%	3721.1	0%	2387.1	0%
Medium	1952.0	68.0%	3550.6	4.6%	2106.3	11.8%
High	875.3	85.7%	2557.8	31.3%	1901.5	20.3%

Figure 3.5, 3.7 and 3.9 show the relationship between energy and PE count. In many cases, the best energy point corresponds to the maximum number of PEs. However, for reduced SNR, fewer PEs can be used to achieve the same or an improved energy level. For example, energy curves of AVA in Figure 3.7 indicate that when SNR=2.45, 3750 mJ energy is consumed to decode 10^5 bits over 6 AVA decoding PEs. In Figure 3.6, it is shown that 5 PEs with SNR=2.65 and 3 PEs with SNR=2.85 can give similar throughput. The corresponding energy is 3550.6 mJ and 2557.8 mJ. Energy savings are 4.6% and 31.3% versus the worst case with the lowest SNR. It can be concluded that substantial energy benefits can be obtained by dynamically reconfiguring algorithm implementation in response to changes in wireless channel quality.

The energy numbers and corresponding reduction for constant throughput at a fixed error rate are shown in Table 3.3. It is apparent that there would be an energy reduction by adapting a CGRA to a variable SNR with BER and throughput fixed. Note that the SNR range in our experiment is limited (from 2.45 dB to 2.85 dB for VA and AVA and 16.0 dB to 17.0 dB for RS). If a CGRA could adapt to a wider range of SNRs, it is possible that the energy savings could be even more substantial.

To complete the experiment, we also measured the instruction count and code size for executables assigned to the PEs (Table 3.4). In future experiments we can use this information to estimate the energy cost of reconfiguration. The Distributor and Collector will have variable code sizes which require configuration. Computational

Table 3.4: Code Size and Instruction Count

		PE Count					
		1	2	3	4	5	6
Distributor	Instruction Count	1546	1556	1564	1577	1588	1596
	Code Size($KBytes$)	7.07	7.11	7.14	7.19	7.24	7.27
Collector	Instruction Count	3358	3367	3374	3386	3396	3403
	Code Size($KBytes$)	14.68	14.72	14.75	14.80	14.84	14.86
Viterbi and AVA code size: 64.34KB, Instruction Count: 3287							
Reed-Solomon code size: 15.76KB, Instruction Count: 3036							

complexity is updated in PEs by changing parameters, therefore the code size and instruction count of the NIOS II decoder is not be affected in this experiment. The Viterbi Algorithm and AVA implementations have the same instruction count and code size because there computations are similar, although parameters determine how many operations are performed.

CHAPTER 4

TIME-SCHEDULED DYNAMIC INTERCONNECT

Interconnect is an important part of an overall energy-efficient system. In this chapter we describe an on-chip interconnect for a CGRA which can be quickly reconfigured. This interconnect uses time-multiplexing based on a schedule to transfer data in a mesh topology. As part of this thesis we will explore CGRA interconnect architecture and its performance for the three communication coding algorithms mentions in earlier chapters: Reed Solomon, Motion Estimation and FIR. To explore runtime reconfigurability, a detailed switchbox and associated configuration memory will be created. The throughput, energy and power of the system under reconfiguration will be evaluated.

4.1 Time-Scheduled on-chip Interconnect

A switchbox-based on-chip interconnect of a CGRA can be configured to support communication for many different applications. The use of time-multiplexing during execution allows the same interconnect wiring to be used many times, potentially saving energy and area. Generally, in a CGRA, not all channels need to be used on every cycle. As a result, wires can be time-multiplexed to eliminate the area overhead found in static signal assignment [35]. Figure 4.1 depicts a multiplexer found in a phase-cycle-based time-scheduled switchbox compared with another MUX in a statically-configured switchbox [55]. The former contains multiple configuration bits stored in SRAM, which can be selected on a per-cycle basis by a *phase* signal. The

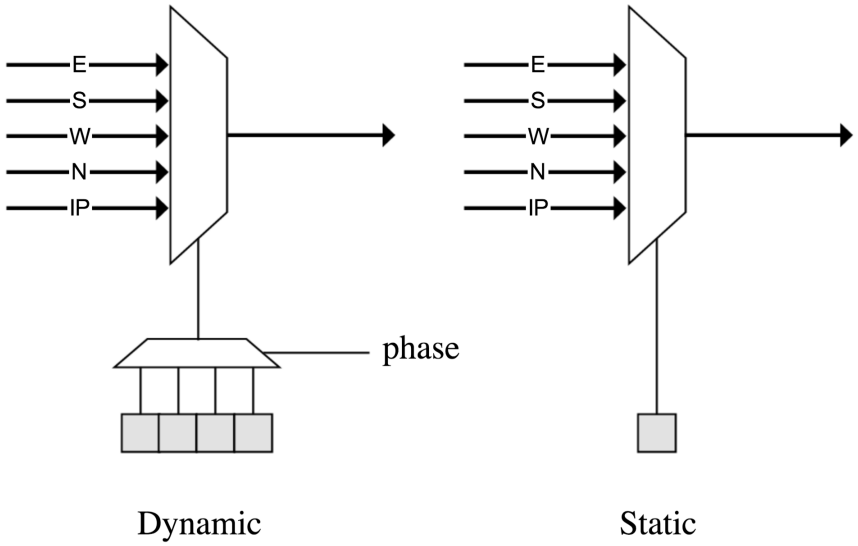


Figure 4.1: Dynamic vs. Static switchbox [55]

latter contains only one configuration SRAM bit. It can be concluded that routing flexibility and hardware multiplexing are obtained at the cost of control complexity.

Another challenge for time-scheduled interconnect is control strategy. Figure 4.2 shows how a phase signal could be generated and how it controls the internal architecture of a switchbox. The phase signal points into a block of memory which contains configuration information for a switchbox. It iteratively selects configurations in a round robin pattern. Each entry in the *SRAM interconnect memory* contains a sequence of bits that control all multiplexers in a switchbox. The control bits of one interconnect instruction is partitioned into different regions, as shown in Fig. 4.3. The first part determines which input buffer the switchbox will be reading, which will be elaborated in the following switchbox interface section. The following five segmentations determine the routing direction of all five input signals (from East, South, West, North and PE). Each segmentation is consisted of 3 bits. Four out of eight total combinations of the 3 bits are used to indicate where the input signal goes.

In Figure 4.2, configurations from 1 to N are pre-determined and stored in *Interconnect Memory*. All configurations are loaded into the switchbox in the order

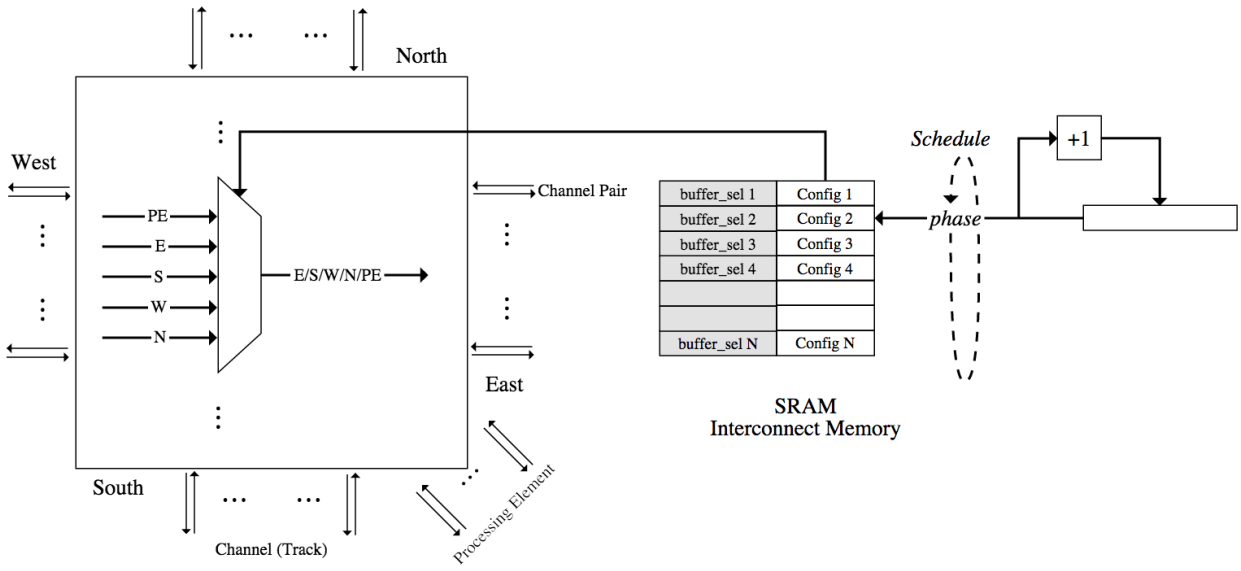


Figure 4.2: Implementation of Time-Scheduled Reconfigurable Switchbox

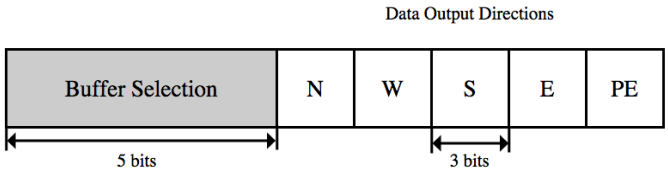


Figure 4.3: Breakdown of an interconnect instruction

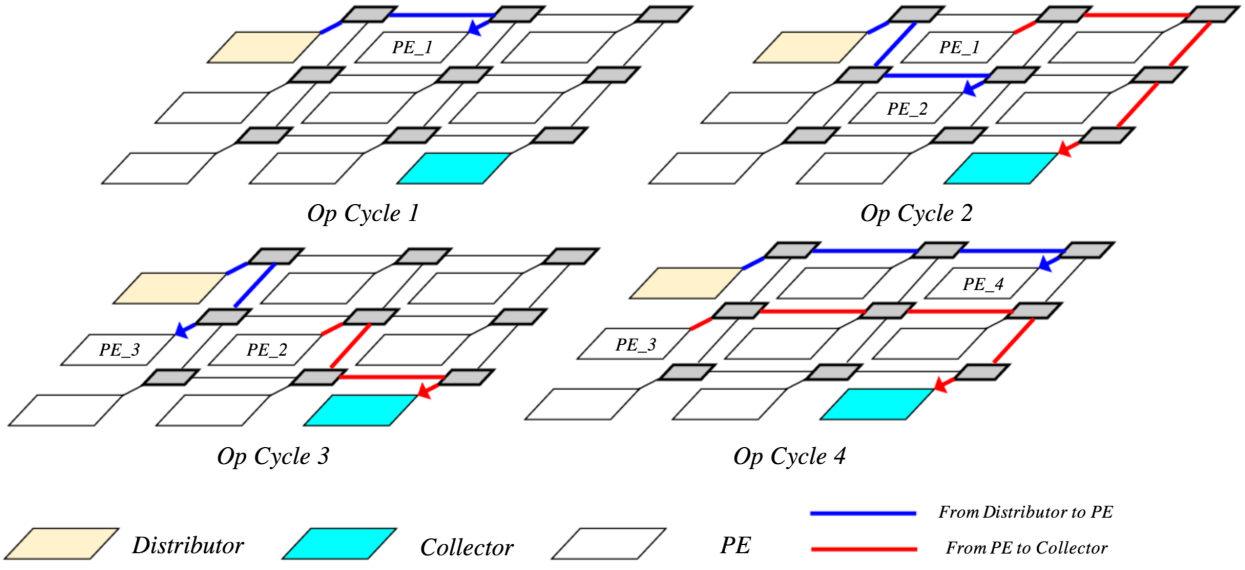


Figure 4.4: An Example of Time-Scheduled on-chip Interconnect

as number-labeled. The system has the flexibility to determine when to load a configuration (effectively perform an interconnect reconfiguration). The contents of the switchbox configurations are determined based on the repetitive datapath behavior of an application (Figure 4.4). The figure depicts 4 operational cycles of an application that requires a data Distributor as well as a Collector, as used in the previous chapter for communication coding applications. Solid blue lines indicate data output from the Distributor to PEs while red lines indicate data transmitted from PEs to the Collector. In operation cycle 1, a connection is built between the Distributor and PE_1 so that data is transferred from the former to the latter. When transmission is finished, the next phase is selected for operation cycle 2. This action creates a channel between the Distributor and PE_2 and another channel between PE_1 and the Collector are established. When data collection is completed, operation cycle 3 is triggered in which the Distributor sends data to PE_3 and the collector gathers data from PE_2. The multi-phase cycle then repeats. In this example, all configurations are organized in a time order, which is defined as a *schedule*. Reconfiguration is triggered by an external event.

In our work, STT-MRAM will be interfaced to the SRAM Interconnect Memory to allow for a quick reconfiguration path. Reconfigurations will be triggered by changes in a specific communication coding algorithm or a change in algorithm. Program memory in the CGRA processing elements will also be interfaced to STT-MRAM to allow for fast program download. The proximity of the STT-MRAM will allow for fast download bandwidth.

4.2 SwitchBox Interface

Previous section gives a big picture of how the time-scheduled switchboxes is built and controlled to route signals. In this section, the interfaces between switchbox and switchbox as well as switchbox and PE will be discussed. There are occasions

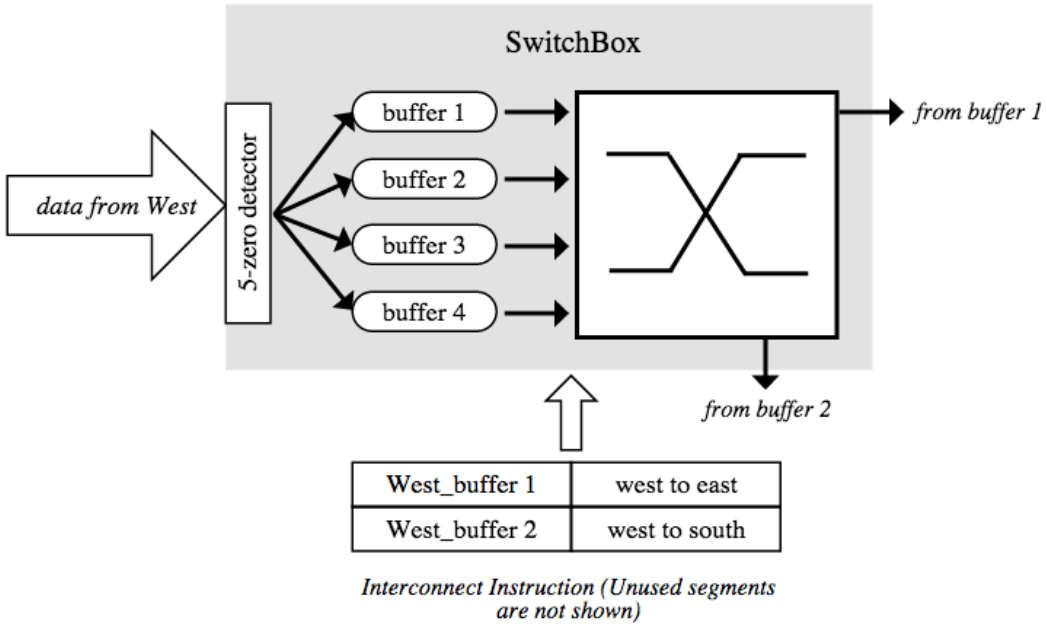


Figure 4.5: Interface of Switchbox to Other Peers and PE

that data from a single source will be routed to multiple sinks and from multiple sources to a single sink. This will be a problem for a time-scheduled interconnection because every clock cycle all multiplexers will be configured to adopt a different data path. Therefore maximum 4 input buffers are implemented on each input side to differentiate different data sources. Keep sending five zero's will switch from one input buffer to the next available one. By selecting different buffers, data from the same input port could be routed to different places. The number of buffers on one input side could be modified.

An example is shown in Fig 4.5. Data is sent through buffer 1 on the west side of the first switchbox. After a sequence of five zero's is sent, buffer 2 on the west side is picked to take the input data. Of each interconnect instruction, the first 5 bits are utilized to specify which buffer it is reading. There are four buffers on each input port, five ports yields 20 buffers. Each one is uniquely addressed with that five bits. With buffer 1 and buffer 2 on the same side, we could differentiate data streams going to different sinks. In this example, buffer 1 on west side goes east and buffer 2

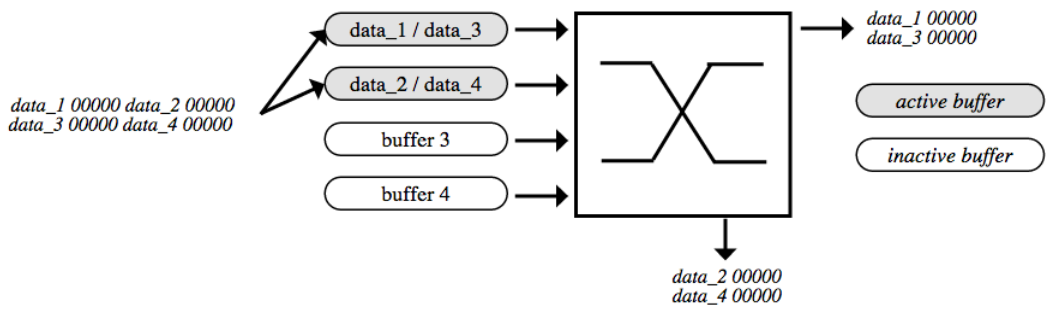


Figure 4.6: Buffer Switching of a SwitchBox Interface

on west side goes south. The five zero's are not consumed in the switchbox, instead, they ripple through all the switchboxes along the datapath so that buffers along the complete datapath could be switched to accommodate a new datapath. Considering that there might be continuous zero's in data streams flowing in the network, we attach a small piece of function before injecting data to the network which shifts the data left by one bit and modify the right most bit as 1. For the receiver end, it shift the data to the right by one. Before shifting and actually reading the data, the receiver will also need to filter out zero's. We believe it will not bring computational burden onto the system because hardware shifting and bitwise operation are very fast.

To better adapt the switchbox interface to different applications, the number of buffers on each input side is open for modification. This provides the ease for application mapping. The reason is that most DSP algorithms are streaming, it is very likely the same operation will be applied onto different data. Therefore, The buffers on one side will be iteratively chosen for multiple times. If the number of buffers is more than needed, an incoming five zero's will direct the input data to a redundant buffer which is not used by any instructions in the interconnect memory. This will halt the system. Therefore, for different applications, the number of buffer activated should be exactly the number of buffer needed. Notice that we set the

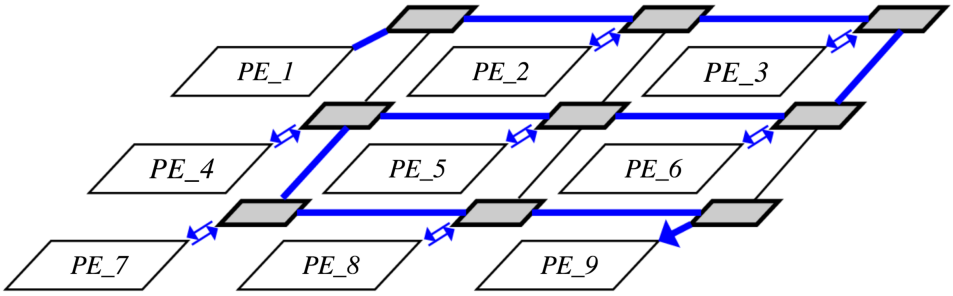


Figure 4.7: Mapping of Streaming Application

maximum to be 4. It is based on the assumption that each input port will at most take 4 different streams. This number could be expanded as long as the number of bits that address the buffers will be expanded accordingly. The process is depicted in Fig 4.6.

4.3 Application Mapping

For this thesis, apart from preliminary experimentation, we will map multiple applications (Reed Solomon, Motion Estimation and FIR Filter) onto our prototype CGRA architecture using the time-scheduled on-chip interconnect. Applications can be categorized into two classes: parallel and streaming. For parallel applications, the workload is evenly distributed onto all available PEs, similar to our preliminary experimentation. All PEs execute in parallel and they simultaneously fetch data from the Distributor and send results to the Collector. Streaming applications will also be mapped as shown in Figure 4.7. The Distributor and Collector are implemented by NIOS-based architecture. All PEs are connected in a nearest-neighbor fashion. Our switchbox is capable of handling both types of applications because for parallel algorithm, multiple input buffers could be activated so that data stream could be handled concurrently (Strictly only one stream is taken care of every clock cycle. However since we assume that there are at most 20 data streams going through one switchbox, which means the maximum between service interval for one data stream

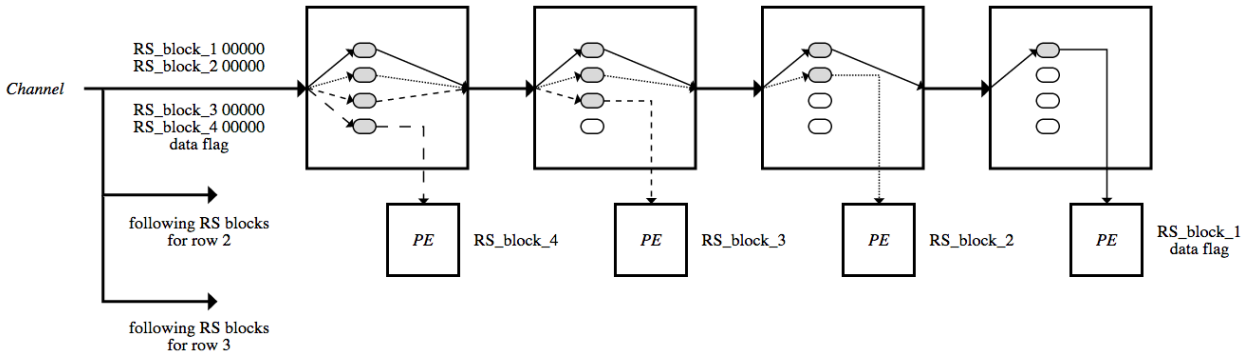


Figure 4.8: Data Distribution of Reed Solomon Application Mapped in Parallel

is 20 clock cycles, it could be assumed that data streams are handled in parallel). For streaming applications, the buffer number of each input side could be set as one. The interconnect memory would contain only one configuration which makes the switchbox static so that streaming datapath can be implemented.

4.3.1 Reed Solomon

In Chapter 2, Reed Solomon algorithm is introduced as a block codec application. The algorithm is mapped as a parallel application with each code block going to available decoding PE. All decoding PEs share the same source code so that no matter where the code block goes, it will be decoded in exactly the same way and sent back to the Collector. The decoding system is a 3x4 array. Four code blocks are sent every time to the decoding array from the communication channel. Each code block goes to one of the four PEs in a row of a decoding array. Five zero's are sent between every two code blocks. After sending four code blocks, a data flag is sent to indicate the termination of sending. Since each symbol in RS code is 8-bit, therefore the data flag needs to be larger than 2^8 so that it will not overlap the data from communication channel. Notice that the five zero's and data flag are generated not from the communication channel but from the interface between the decoding array and channel. A visual demonstration of the above operation is shown in Fig 4.8.

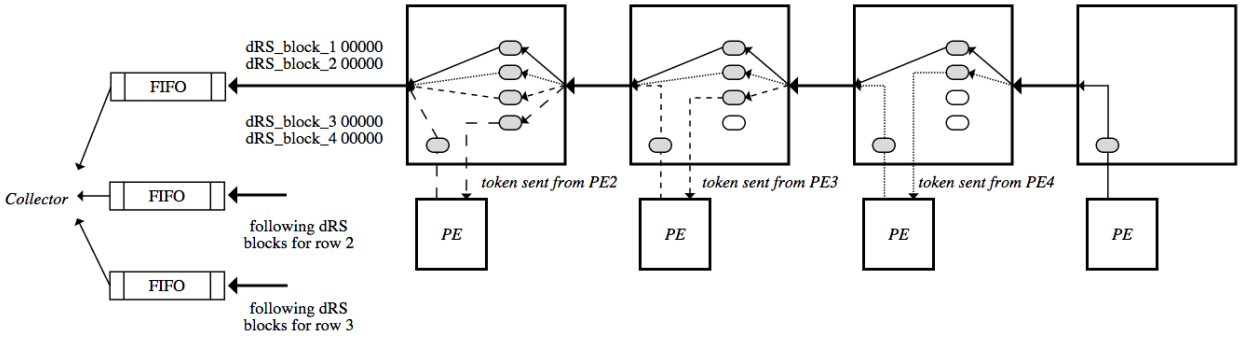


Figure 4.9: Data Collection of Reed Solomon Application Mapped in Parallel

In our emulation platform, we assume that all decoded data is sent back to a Collector. Therefore, the decoded data from four PEs should be routed to a single sink, the Collector. Since the Collector cannot take all four streams at the same time, therefore a communication scheme needs to be established so that only one PE could talk to the Collector at a given time. To enable this, another data flag needs to be created to play the role as a token. It will be passed from the first PE to the last one and go back to the first again. The PE with the token will be able to send the decoded data. This step is shown in Fig 4.9.

After distributing code blocks to the complete array (12 code blocks are distributed), the decoding array will halt the reading of the communication channel until decoding is finished(The decoding array cannot reading data and decoding or decoding and sending data at the same time). Each row of the decoding array has an output FIFO which also interfaces to the Collector. The Collector monitors the status of the FIFO and iteratively reads all the FIFO if they are not empty. There is a counter associated with reading of the last row that indicates if the decoding of one round, which is decoding 12 code blocks, has finished or not. The sequencing nature of the Collector determines that the three FIFOs cannot be read concurrently.

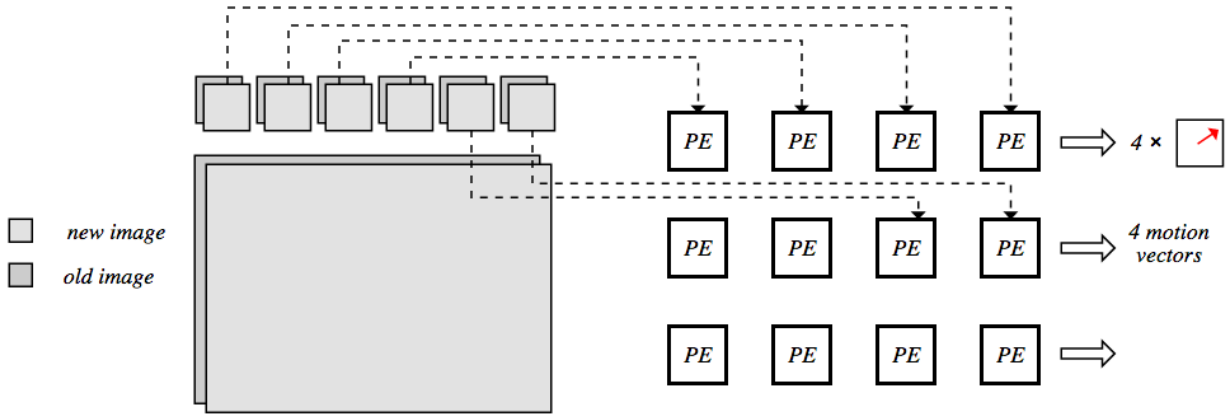


Figure 4.10: Mapping of Motion Estimation Algorithm

4.3.2 Motion Estimation

The Motion Estimation algorithm requires two pictures from different timing points at the same time. The older one, in terms of time, provides a center point from which a motion vector starts. The newer one, where the algorithm is applied, provides a pixel position where the center point in the older will most likely be, which is the ending point of the motion vector. Since each picture will be segmented into hundreds of windows, then two windows from both of the images that has the same location on the picture will be sent to one PE for comparison. A vector will be returned to indicate the estimated motion in that window. Since pictures are broken down into windows and data is distributed in the unit of window, therefore distributing and collecting models are the same as that in RS. Fig 4.10 shows how the pictures are broken down and sent. Similar to RS, all PEs share the same source code so that each motion vector is generated in the same way.

4.3.3 FIR Filter

As described in Chapter 2, the FIR filter we implemented is a band pass filter with ω sitting in the range [0.35, 0.65]. There are typically two ways to map FIR to our architecture, streaming and parallel. In this thesis we choose parallel because

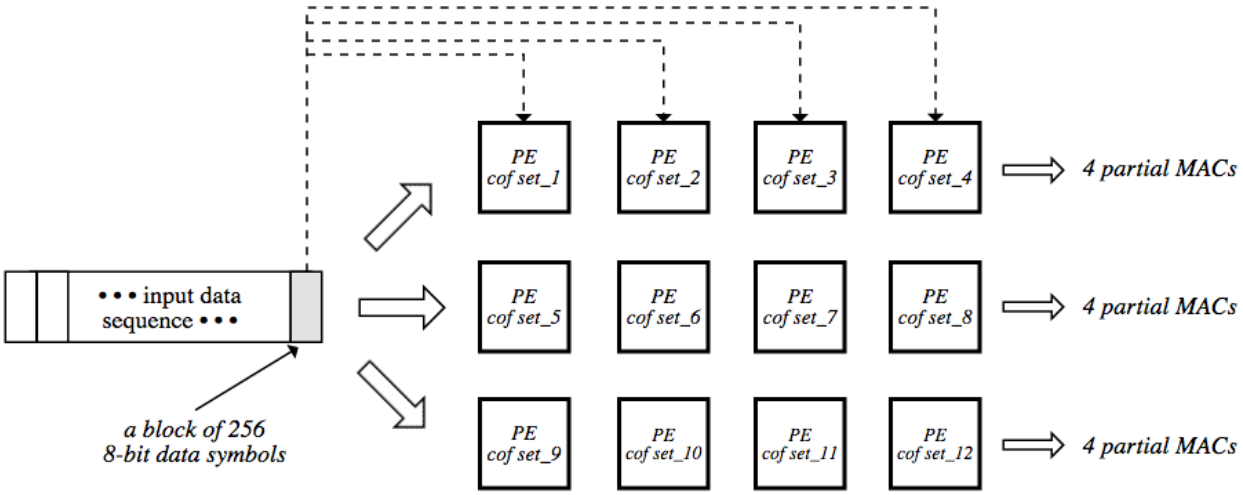


Figure 4.11: Mapping of FIR Filter

it is similar to the distributing and collecting model we use in previous applications. Another reason is that the coefficients (taps) are generated from Matlab, which is not suitable for streaming mapping. Therefore, incoming 8-bit data is distributed among the 3x4 array. The data is distributed in the unit of block, with each block having 256 8-bit radar data symbols considering data distribution overhead and computational complexity of every processing element. Each PE has a unique piece of source code that is capable of computing partial finite impulse response. Mapping of FIR filter is shown in Fig 4.11. The Collector will add up all the partial MACs together. Since there will be zero's inserted between different MAC's in the output of one row, therefore it is easy to differentiate output from PEs in the same row.

CHAPTER 5

MRAM-ENABLED COARSE-GRAINED RECONFIGURABLE ARRAY

In this chapter, the FPGA emulation platform for MRAM-enabled CGRA will be introduced and elaborated. The experimental approach will be introduced and the results will be presented and analyzed. From desktop-based simulation, we expect to obtain overall performance of implementing STT-RAM in 40 nm technology as well as the comparison against prevailing SRAM technology. From FPGA emulation and performance/power measurement, we expect to see the energy benefits of configuring the Instruction/Data memory and interconnect memory attached to the switchbox to adapt to time-varying physical environment. If an energy-saving configuration update is needed, a new configuration can be quickly swapped into compute blocks and interconnect switchboxes to minimize system down time. Our experiments show that the use of MRAM reduces overall application energy consumption by nearly 30% when dynamic reconfiguration is used.

5.1 Architecture

Our MRAM-enabled coarse-grained architecture is designed to support streaming applications with pre-scheduled communication paths. As shown in Fig. 5.1, the architecture is based on a two-dimensional array of ALU-based processing elements[1]. Each cell contains a 32-bit ALU, a 36K-word data memory, and a 24K-word configuration memory. These memories are directly connected to much larger multi-MB

[1]Nine PEs are shown for clarity, but much larger array sizes are feasible.

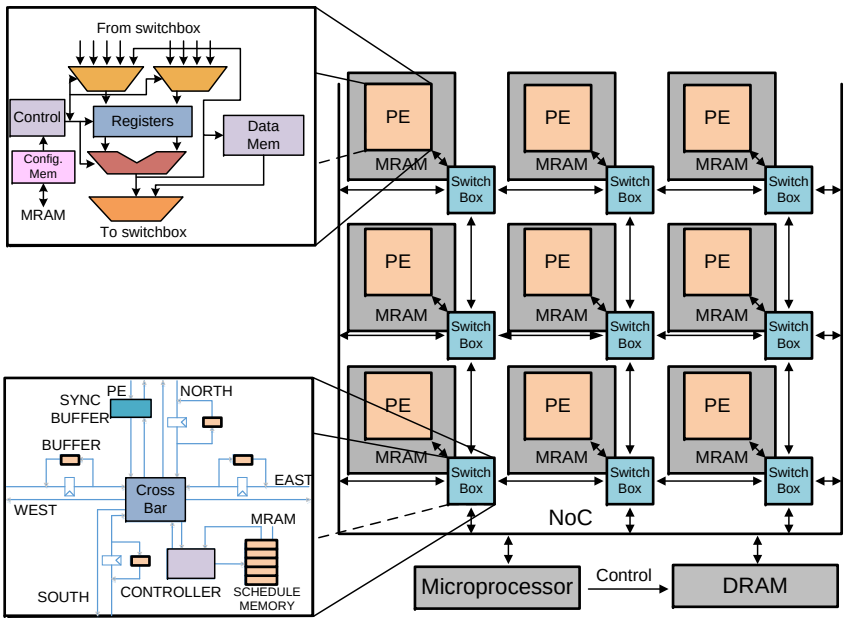


Figure 5.1: Coarse-grained reconfigurable architecture with interface to MRAM.

non-volatile memory (MRAM) blocks which can be used to change their configuration memory contents. The array interfaces to a network of multiplexer-based switchboxes to provide inter-element communication. Data are switched in 32-bit increments to lower the switch configuration memory overhead. A dedicated microprocessor is located on the bottom of the array to provide control over the loading of configurations to MRAM (if needed) and coordinating signal processing application data transfer to and from DRAM.

The switchbox for each processing element (Fig. 5.1) operates on a pre-determined schedule that coordinates communication with neighboring elements, which is elaborated in Chapter four. Schedule information for cycle-by-cycle configuration of the multiplexer-based crossbar is stored in the *schedule memory*. Buffers and flow control signals are provided on each interface port to prevent overflow. On each cycle for a port, data can be forwarded from the buffer or the neighbor. Each buffer contains storage for data from multiple independent streams (effectively, virtual channels). For our architecture, the clock speed of the switchbox and the PE is the same, so com-

munication between the two components is synchronous. As a data value becomes available for a stream, the PE or switchbox writes the value to the appropriate buffer location.

The key new architectural feature in this scalable multi-PE system is the use of MRAM to cache multiple PE and switchbox configurations. The use of MRAM, rather than more traditional SRAM, for configuration storage leads to several memory interfacing and organizational issues that can be addressed by examining the operation of time-varying signal processing applications. Localized embedded MRAM can dynamically reconfigure the configuration memories under control of the external controller enabling changes in signal processing algorithm implementations in response to the physical environment and/or signal characteristic changes (e.g. noise). Specific MRAM issues that must be considered include:

- The energy consumption and performance of MRAM are tied to the width and depth of the MRAM array. In this work we explore whether it is more efficient to dedicate one wide MRAM bank per PE or to share a common bank across multiple PEs

- The availability of large MRAM arrays allows for more rapid reconfiguration between applications and between multiple configurations of a single application. We examine how this increased flexibility reduces system energy consumption by adapting computation to environment factors.

5.2 Experimental Approach

To validate the energy and performance aspects of our approach, we use a combination of simulation and FPGA-based architectural emulation to accurately measure system energy and performance. NVSIM [18] is used to evaluate circuit-level area, performance, power, and energy for our STT-RAM (MRAM) implementations. This

simulator has been widely used in a variety of experimental research studies. In our work, the NVSIM LOP (low power) library is used to evaluate read/write access time and static and dynamic energy for varying MRAM array sizes. NVSIM was also used to model SRAM banks in the same 40 nm technology.

The processing element in our system, based on the ALU-based SPREE functional unit [59], and the switchbox were written in Verilog. The performance of the architecture for 12 PEs and switchboxes was validated using both RTL simulation and in-circuit emulation after design synthesis using an Altera DE4 FPGA board. Cycle-accurate performance and node toggle rate calculations for the applications were determined using the Altera Performance Counter Unit, instrumented in the FPGA hardware. This IP core could be activated by the microprocessor attached to the PE array, which is implemented by a NIOS processor. The performance counter starts when data is fed into the array and stops when all data has been processed and returned. Detailed power consumption was determined using a current measurement sensor included on the DE4 board. Another sensor is implemented to measured the voltage drop of the chip power supply. Both of the sensors are connected to Analog-Digital Converters(ADCs) which interface to FPGA via serial peripheral interface(SPI) bus. A dedicated NIOS processor, which has no connection to any other parts of the system, is instrumented in FPGA to read current and voltage and power numbers on terminal every second.

Power and performance were scaled from measured 40 nm FPGA values found in emulation to estimated 40 nm ASIC values. The switchbox and SPREE designs were analyzed by both Altera PowerPlay and Synopsys Design Compiler using the NanGate open core 40 nm library[2] to determine the scaling factors of 11.6 for performance and 5.0 for power. Both of the numbers are derived from ASIC simulation and FPGA

[2]www.nangate.com

emulation. Maximum clock frequency for ALU like SPREE processing element in ASIC is 1.388GHz, while for FPGA, SPREE is running at 120MHz, from which we infer the performance scaling factor. FPGA emulation power for SPREE is 60mW, ASIC simulation gives 12.138mW, which gives a 5.0 power scaling factor.

Power simulation from Altera PowerPlay is determined by simulating all three benchmarks with ModelSim Altera edition, dumping signal activities on each node and putting that information into PowerPlay for power evaluation. Since the static power number it returns is of the complete Stratix IV chip, therefore we infer the static power for our design based on the portion of resource utilization.

During emulation, input data was streamed to PEs in the array from DRAM (Fig. 5.1) using a NIOS microprocessor. Design performance was constrained by the processing rate of the PEs, rather than the DRAM interface. The implementation specifics for each benchmark are provided below:

RS decoding - In this application, each PE is assigned one RS decoder. The error-correcting capability of the decoder (e.g. number of k message symbols) is determined by the PE configuration. Our experimentation considers k values ranging from 217 to 239 out of $n = 255$ total symbols (e.g. RS(255,217) to RS(255,239)). A constant codeword error rate (CER) of 10^{-4} is used. The specific decoder configuration is selected based on channel noise. To simulate the behavior of a communication system, channel noise (SNR) was considered to vary at an accepted rate [4] of 1.5 seconds and a new noise value was randomly selected at this rate. If necessary, the CGRA configuration is updated in response to the change in noise, necessitating a PE configuration load from the attached configuration cache. For results, we consider 7 different decoder configurations. During processing, decoded data is streamed back to DRAM from each PE.

Motion estimation - For this application, a series of 1024×1024 pixel greyscale images are distributed to the PE array. Each of these images is part of a sequence

in which motion can be detected. Subsequent images in the sequence are split into windows and each window is tested by each PE for motion. In this implementation, the accuracy of the motion estimation algorithm is dependent on the window size. For higher accuracy, a smaller window size is used, leading to more windows and increased computation. For reduced accuracy, the opposite effect is observed. In our experimentation, we consider square window sizes of 14, 16, 18, 20, 24 and 26 pixels on a side. Reconfiguration is considered every 0.5s based on changes in expected motion.

FIR filtering - This application involves the implementation of a filter with tap counts ranging from 120 to 1920 taps. A stream of input data is sourced from the DRAM and the filtered data is streamed back to DRAM. For our experimentation, eight-bit sampled radar data is used [44]. Computation for the taps is distributed evenly across the PEs. An increased number of taps leads to more accurate filtered data. Reconfiguration is considered every 0.5s based on changes in expected data quality.

5.3 Results

There are three key issues that must be considered in evaluating the benefits of using MRAM versus SRAM for on-chip configuration storage:

1. The relative power consumption, both static and dynamic, of MRAM (in this case STT-RAM) versus SRAM.

2. The relative power consumption of the configuration storage versus the target application.

3. The power and energy benefits of using a configuration cache and performing dynamic reconfiguration versus always using the most powerful version of the target application and not performing reconfiguration.

Table 5.1: PE and switchbox configuration bit sizes for one configuration of each application (results were generated by Tedy Thomas)

	one PE			whole CGRA		
	PE	switchbox	Total	PE	switchbox	Total
RS	81,920	16,384	98,304	983,040	196,608	1,179,648
ME	49,152	16,384	65,536	589,824	196,608	786,432
FIR	40,960	16,384	57,344	491,520	196,608	688,128

Each of these three issues are addressed in our results.

5.3.1 MRAM and SRAM Energy Tradeoffs

Table 5.1 illustrates the configuration bit sizes required for the individual processing elements and switchboxes. A CGRA with 12 PEs is used for experimentation. The total for 12 PE/switchbox pairs is shown in the right columns of the table for each application. PE *configuration memory* bits set connections between the ALU and data memory. Switchbox configuration information, stored in the *schedule memory*, configures the routing crossbar on a per-cycle basis. As will be described later in this section, seven distinct Reed Solomon configurations, six distinct motion estimation configurations and five distinct FIR configurations are possible. As a result, a configuration cache must hold multiple copies of configuration information for each PE/switchbox, even though only one is loaded into the PE config. memory and switchbox schedule memory at a given time.

An important driver of MRAM, rather than SRAM, use for configuration cache storage is the substantially reduced leakage power of STT-RAM compared to SRAM. The amount of this reduction is apparent from Table 5.2. For 128KB, 1MB, and 4MB memory blocks, the ratio of leakage power for SRAM versus MRAM is 2.8, 2.4, and 8.3, respectively. In general, STT-RAM cells exhibit little leakage, so much of the array leakage is due to the decoders and output multiplexer. From the table, it can also be noted that the dynamic power of similar sized MRAM and SRAM blocks

Table 5.2: MRAM and SRAM parameter comparison for 128KB, 1MB and 4MB memory blocks with 32-bit output (results were generated by Tedy Thomas)

	MRAM			SRAM		
	128KB	1MB	4MB	128KB	1MB	4MB
Read time (ns)	1.10	1.30	1.67	0.31	0.60	1.23
Write time (ns)	5.24	5.36	5.88	0.28	0.49	0.82
Rd energy (pJ)	17.49	39.17	127.24	13.44	30.34	139.10
Wr energy (pJ)	17.05	29.74	105.71	10.00	22.81	116.45
Area (mm^2)	0.57	2.42	4.92	0.74	3.38	12.95
Leakage (mW)	0.61	3.56	3.87	1.71	8.52	31.98

Table 5.3: Reed Solomon decoder statistics after mapping to a 12 PE array

k	SNR (db)	Rate (Mb/s)	Energy per million bits decoded (mJ)
239	19.2-20.0	16.39	4.44
237	18.6-19.2	14.62	5.03
233	17.2-18.6	11.71	6.38
229	16.2-17.2	10.81	6.97
225	15.2-16.2	8.38	8.87
221	14.4-15.2	7.21	10.14
217	13.6-14.4	6.34	11.76

are similar. However, for our use of dynamic reconfiguraton where configuration update takes place on the order of 500 ms to 1.5 s, the dynamic power consumed by reconfiguration is minimal. For example, to reconfigure a PE and a switchbox requires 3,012 32-bit reads from the configuration cache and 3,012 32-bit writes to the PE configuration and switchbox schedule memories. For a 128KB MRAM, memory reading requires 52.7 nJ and for 128KB SRAM reading requires 40.48 nJ, a modest amount.

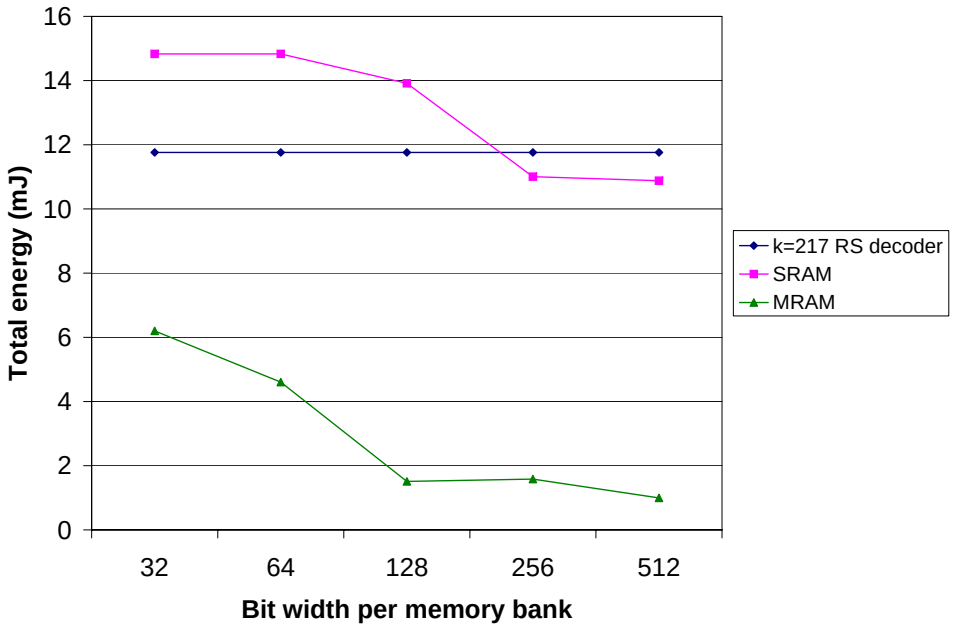


Figure 5.2: Energy consumption of a RS(255, 217) decoder and MRAM versus SRAM configuration cache storage for a selection of cache output bit widths. The total CGRA-wide configuration cache storage for each bit width is 12 MB.

5.3.2 Configuration Performance and Energy Tradeoffs

As mentioned in Experimental Approach section, RS decoders of differing error-correcting capability and power consumption can be used at different times based on channel noise level. Table 5.3 indicates appropriate message symbol counts (out of 255 total) for our implemented system. The results indicate that decode rate and energy consumption vary with k. As k changes from 239 to 217, decode rate is reduced from 16.39 to 6.34 Mb/s and the energy to decode one million bits increased from 4.44 mJ to 11.76 mJ. One approach to providing sufficient error correcting capability would be to use an RS(255,217) decoder at all times and avoid decoder reconfiguration. However, this approach limits opportunities for faster decode rates and reduced energy consumption.

Fig. 5.2 illustrates the energy cost of configuration cache storage in relation to Reed Solomon coding per one million decoded bits. The CGRA-wide energy for

the RS application is shown along with amount of energy needed to store 12 MB of configuration cache storage in either MRAM or SRAM. This storage is assessed for banks of memory with different output bit widths. If a 32-bit output is used, each PE/switchbox pair has a dedicated MRAM or SRAM cache of 1 MB. For a 512-bit output, a single 12 MB bank is used to configure all PE/switchbox pairs simultaneously. The single-bank option shows lower energy due to reduced decode and multiplexer logic in the memory. However, all individual PE/switchbox pairs must be configured at the same time.

It can be observed from the figure that if SRAM is used to cache configurations, its energy is *about the same* as the energy used by the CGRA PE and switchbox circuitry to perform the RS computation, an observation which echoes a previous finding of 43% of energy consumed by configuration caches [30]. However, across all memory widths, MRAM configuration cache energy is significantly lower than RS decoding energy (from about 50% less for 12 banks of 1MB to about 12× lower for 1 bank of 12MB). As mentioned in the previous subsection, the bulk (over 98%) of the MRAM and SRAM configuration cache energy consumption is due to static power since the RAMs are infrequently accessed. Although we do not consider voltage scaling for the SRAMs which possibly could help reduce their energy consumption, we also do not include the energy needed to initially load the SRAMs from off-chip non-volatile storage. This action is unneeded for non-volatile MRAM.

Similar results in terms of throughput and energy can be observed for the motion estimation application (Fig. 5.3) for search window sizes between 14 and 26. The figure indicates that the application throughput improves with increasing window size since fewer windows must be searched. Two configuration cache sizes per PE are considered, 128 KB and 1 MB. A 1 MB cache size is desirable since it is able to hold the configurations of all versions of all applications. A 128 MB cache size can hold all versions of ME. The configuration cache energy per frame is reduced as window

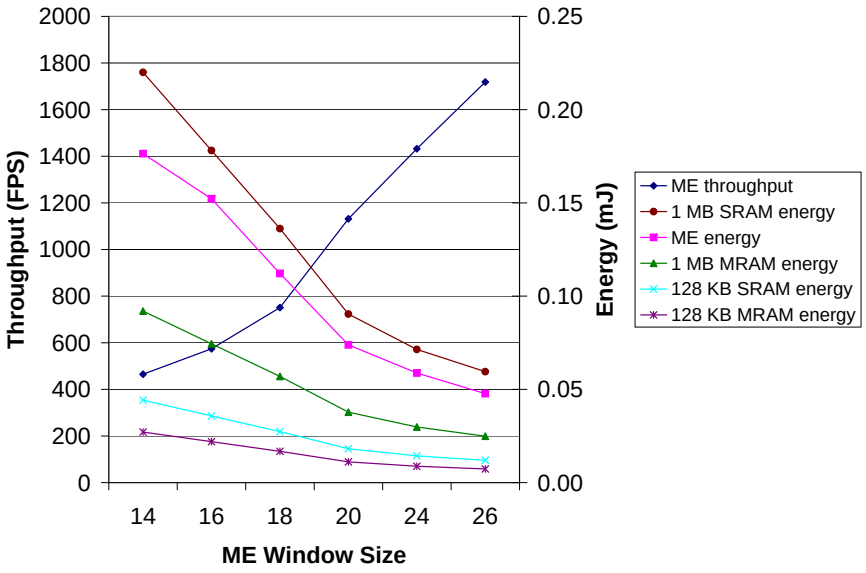


Figure 5.3: Energy consumption and throughput of motion estimation mapped to 12 PEs for different ME window sizes. The energy consumption of the application and the caches was calculated over one processed frame. All configuration caches are implemented as 12 individual banks of 1 MB or 128 KB (one bank per PE/switchbox pair) MRAM or SRAM.

size is increased because the frame requires less time for processing. For comparable cache sizes, MRAM once again requires about 2 to 3× less energy than its SRAM counterparts. It is notable that once again, the energy consumption of the SRAM-based configuration cache is greater than the processing and switchbox components of the application.

Similar throughput and energy results are seen for the floating point FIR application. The throughput difference between the 120 tap implementation (5.2 MB/s) and the 1920 tap version (0.56 MB/s) is almost an order of magnitude. For a 1MB per PE/switchbox configuration cache, the average energy per million decoded bits of the MRAM (29.20 mJ) is less than half the energy of the 960 tap FIR filter (74.47 mJ). However, an SRAM-based cache is nearly the same value (69.84 mJ).

Table 5.4: Results of dynamic reconfiguration for mapped applications using three 4MB (256Kx128) memory banks per CGRA.

	MRAM				
	Ave. pwr (mW)	Energy MRAM (mJ)	Ave. Ene. App (mJ)	Energy Total (mJ)	% Ene. imprv
RS	90.91	0.99	7.66	8.65	26
ME	94.31	0.013	0.106	0.11	38
FIR	97.91	7.11	59.31	66.42	33
	SRAM				
	Ave. power (mW)	Energy SRAM (mJ)	Ave. Ene. App (mJ)	Energy Total (mJ)	% Ene. imprv
RS	176.46	9.12	7.66	16.78	-43
ME	179.86	0.12	0.10	0.22	-22
FIR	183.46	69.84	59.31	129.15	11

5.3.3 Benefit of Dynamic Reconfiguration

For all three applications (Reed Solomon decoding, motion estimation, and FIR), energy consumption can be improved by adapting the application configuration to the measured environmental condition (e.g. noise in the communications channel for RS). In a final set of experiments, we examine the benefit of using dynamic reconfiguration of the CGRA with either an MRAM- or SRAM-based cache versus simply using the most powerful configuration of the application all the time (e.g. using the RS(255, 217) decoder for all RS decoding). A series of 10,000 random selections of decoder noise, required ME motion detection, and FIR accuracy were generated to represent changing environmental conditions. The application configuration which best met the requirements for these conditions were then chosen for each selection.

The average power and energy of the applications considering dynamic reconfiguration from an MRAM-based or SRAM-based cache is shown in Table 5.4. Energy is determined per one million processed bits for RS and FIR and per frame for ME. The percentage improvement is the energy improvement for the average case using reconfiguration versus the most computationally powerful configuration of the appli-

cation (e.g. RS(255, 217), window size 14 for ME, 1920 taps for FIR). For example, for RS using an MRAM cache, the RS(255, 217) decoder has a 11.76 mJ dissipation while the average energy with reconfiguration is 8.65 mJ, a 26% savings. Although not shown in the table, the throughput performance of the applications using the reconfigured average case versus the most computationally powerful configuration is 70%, 118%, and 94% for RS, ME, and FIR, respectively.

From Table 5.4, it is apparent that the use of SRAM is limiting since its leakage is high. The average energy consumption of the reconfigured RS and ME applications are *increased* versus the most computationally-powerful versions by 43% and 22%, respectively due primarily to this leakage. The throughput performance of the applications is improved with reconfiguration by the same amount as for the MRAM caches since reconfiguration time is so small (μs) versus the frequency of reconfiguration (s).

CHAPTER 6

CONCLUSION

Coarse-Grained Reconfigurable Array(CGRA) is known as a promising architecture for data-streaming application due to its natural support for pipelined as well as parallel applications. Each processing element in this array is capable of performing basic mathematical and logic operation. It is competitive in terms of reconfiguration overhead compared to its fine-grained peer, Field Programmable Gate Array. However, such energy reduction technique is not sufficient enough to shrink leakage power. 6-T SRAM cell consumes considerable amount of static energy(could be 43%).

In this thesis work, we have introduced the idea of using STT-RAM, the most promising candidate of MRAM, to implement a configuration cache on a coarse-grained reconfigurable array (CGRA). We have evaluated using a big chunk of MRAM which is shared by all processing elements and separated but smaller chunks of MRAM so that each PE will have private memory block. For CGRAs, leakage power can dominate configuration cache energy consumption. We show that the use of MRAM as a configuration cache is preferable to SRAM for a collection of three signal processing applications, due to reduced leakage. We also show that the reconfiguration overhead of MRAM and SRAM are the same. Therefore implementation of MRAM is beneficial in terms of energy. We also examine the area of both of the memory fabric and with an increasing size of memory chunk, which is the prevailing trend, MRAM will occupy less than half of the area of SRAM(4MB).

To further reduce energy consumption, we also explore the possibility of adapting CGRA to time-caring application requirements. By using dynamic CGRA reconfigu-

ration in response to environmental factors (e.g. noise), application energy consumption is reduced by about 30% with significant improvements in performance versus the constant use of the most power hungry application configuration.

In the future we plan to consider larger CGRA sizes so that we could map more complicated applications. A larger PE array will also require more memory support. We believe that, from studying the trend of memory performance, a larger MRAM chunk will bring more area efficiency as well as leakage power reduction compared to SRAM. We would also consider the use of dynamic voltage scaling to save configuration cache energy consumption. Since the configuration cache is read-biased and not always activated, the supply voltage of memory can be toggled so that energy could be further reduced.

BIBLIOGRAPHY

[1] Allen, Jonathan D. Energy efficient adaptive reed-solomon decoding system.

[2] Altera. Nios II Performance Benchmarks. `http://www.altera.com/literature/ds/ds_nios2_perf.pdf`.

[3] Altera. Performance Counter Core. `http://www.altera.co.jp/literature/hb/nios2/qts_qii55001.pdf`.

[4] Atieno, Lilian, Allen, Jonathan, Goeckel, Dennis, and Tessier, Russell. An adaptive reed-solomon errors-and-erasures decoder. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays* (2006), ACM, pp. 150–158.

[5] Bansal, Nikhil, Gupta, Sumit, Dutt, Nikil, and Nicolau, Alexandru. Analysis of the performance of coarse-grain reconfigurable architectures with different processing element configurations. In *Workshop on Application Specific Processors, held in conjunction with the International Symposium on Microarchitecture (MICRO), 2003* (2003), Citeseer.

[6] Bier, Jeff. Processors for embedded digital signal processing. `http://www.bdti.com/MyBDTI/pubs/200805_EECS124.pdf`, 2008.

[7] Cao, Liang, and Xinming, Huang. Smartcell: An energy efficient coarse-grained reconfigurable architecture for stream-based applications. *EURASIP Journal on Embedded Systems 2009* (2009).

[8] Carroll, Allan, Friedman, Stephen, Van Essen, Brian, Wood, Aaron, Ylvisaker, Benjamin, Ebeling, Carl, and Hauck, Scott. Designing a coarse-grained reconfigurable architecture for power efficiency. In *Department of Energy NA-22 University Information Technical Interchange Review Meeting* (2007).

[9] Chan, F., and Haccoun, D. Adaptive viterbi decoding of convolutional codes over memoryless channels. *Communications, IEEE Transactions on 45*, 11 (Nov 1997), 1389–1400.

[10] Chen, D, and Rabaey, J. Paddi: Programmable arithmetic devices for digital signal processing; vlsi signal processing iv, 1990.

[11] Chen, Deming, Cong, Jason, and Pan, Peichen. Fpga design automation: A survey. *Foundations and Trends® in Electronic Design Automation 1*, 3 (2006), 139–169.

[12] Chen, Yiran, (Helen) Li, Hai, Wang, Xiaobin, Zhu, Wenzhong, Xu, Wei, and Zhang, Tong. A nondestructive self-reference scheme for spin-transfer torque random access memory (stt-ram). In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010* (March 2010), pp. 148–153.

[13] Clarke, CKP. Reed-solomon error correction. *British Broadcasting Corporation, R&D White Paper* (2002).

[14] Compton, Katherine, and Hauck, Scott. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys (csuR) 34*, 2 (2002), 171–210.

[15] Costello, Daniel J, Hagenauer, Joachim, Imai, Hideki, Wicker, Stephen B, et al. Error control coding. In *Fundamentals and Applications, Printice Hall, Upper Saddle River, NJ* (2004), Citeseer.

[16] Cronquist, Darren C, Fisher, Chris, Figueroa, Miguel, Franklin, Paul, and Ebeling, Carl. Architecture design of reconfigurable pipelined datapaths. In *Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on* (1999), IEEE, pp. 23–40.

[17] Dhillon, Yuvraj Singh, Diril, Abdulkadir Utku, Chatterjee, Abhijit, and Lee, Hsien-Hsin Sean. Algorithm for achieving minimum energy consumption in cmos circuits using multiple supply and threshold voltages at the module level. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design* (2003), IEEE Computer Society, p. 693.

[18] Dong, Xiangyu, Xu, Cong, Xie, Yuan, and Jouppi, Norman P. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 31*, 7 (2012), 994–1007.

[19] Ebeling, Carl, Cronquist, Darren C, and Franklin, Paul. Rapid?reconfigurable pipelined datapath. In *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*. Springer, 1996, pp. 126–135.

[20] EECS 6.03, Massachusetts Institute of Technology. Covolutional Coding. `http://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf`, 2010.

[21] Eguro, Kenneth, and Hauck, Scott. Issues and approaches to coarse-grain reconfigurable architecture development. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on* (2003), IEEE, pp. 111–120.

[22] Fan, Kevin, Kudlur, Manjunath, Dasika, Ganesh, and Mahlke, Scott. Bridging the computation gap between programmable processors and hardwired accelerators. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on* (2009), IEEE, pp. 313–322.

[23] Friedman, Stephen, Carroll, Allan, Van Essen, Brian, Ylvisaker, Benjamin, Ebeling, Carl, and Hauck, Scott. Spr: an architecture-adaptive cgra mapping tool. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays* (2009), ACM, pp. 191–200.

[24] Goldstein, Seth Copen, Schmit, Herman, Budiu, Mihai, Cadambi, Srihari, Moe, Matthew, and Taylor, R Reed. Piperench: A reconfigurable architecture and compiler. *Computer 33*, 4 (2000), 70–77.

[25] Hosomi, M., Yamagishi, H., Yamamoto, T., Bessho, K., Higo, Y., Yamane, K., Yamada, H., Shoji, M., Hachino, H., Fukumoto, C., Nagao, H., and Kano, H. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International* (Dec 2005), pp. 459–462.

[26] HUTCHBY, Fames A, Cavin, Ralph, ZBIRNOV, Victor, Brewer, Joe E, and Bourianoff, George. Emerging nanoscale memory and logic devices: A critical assessment. *Computer 41*, 5 (2008), 28–32.

[27] Karmakar, Gour, and Dooley, Laurence S. *Mobile multimedia communications: Concepts, applications, and challenges*. Information Science Reference, 2008.

[28] Kawahara, T., Takemura, R., Miura, K., Hayakawa, J., Ikeda, S., Lee, Y., Sasaki, R., Goto, Y., Ito, K., Meguro, T., Matsukura, F., Takahashi, H., Matsuoka, H., and Ohno, H. 2mb spin-transfer torque ram (spram) with bit-by-bit bidirectional current write and parallelizing-direction current read. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International* (Feb 2007), pp. 480–617.

[29] Kim, Kinam, and Jeong, Gitae. Memory technologies for sub-40nm node. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International* (2007), IEEE, pp. 27–30.

[30] Kim, Yoonjin, Park, Ilhyun, Choi, Kiyoung, and Paek, Yunheung. Power-conscious configuration cache structure and code mapping for coarse-grained reconfigurable architecture. In *Proceedings of the 2006 international symposium on Low power electronics and design* (2006), ACM, pp. 310–315.

[31] Kumar, Rakesh, Zyuban, Victor, and Tullsen, Dean M. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on* (2005), IEEE, pp. 408–419.

[32] Lam, C.H. The quest for the universal semiconductor memory. In *Electron Devices and Solid-State Circuits, 2005 IEEE Conference on* (Dec 2005), pp. 327–331.

[33] Lemieux, G., Lee, E., Tom, M., and Yu, A. Directional and single-driver wires in fpga interconnect. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on* (Dec 2004), pp. 41–48.

[34] Li, Hai, Xi, Haiwen, Chen, Yiran, Stricklin, J., Wang, Xiaobin, and Zhang, Tong. Thermal-assisted spin transfer torque memory (stt-ram) cell design exploration. In *VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on* (May 2009), pp. 217–222.

[35] Liang, Jian, Swaminathan, Sriram, and Tessier, Russell. asoc: A scalable, single-chip communications architecture. In *Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on* (2000), IEEE, pp. 37–46.

[36] Manne, Srilatha, Klauser, Artur, and Grunwald, Dirk. Pipeline gating: speculation control for energy reduction. In *ACM SIGARCH Computer Architecture News* (1998), vol. 26, IEEE Computer Society, pp. 132–141.

[37] Mei, Bingfeng, Lambrechts, A., Mignolet, J-Y, Verkest, D., and Lauwereins, R. Architecture exploration for a reconfigurable architecture template. *Design Test of Computers, IEEE 22*, 2 (March 2005), 90–101.

[38] Mirsky, Ethan, and DeHon, Andre. Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on* (1996), IEEE, pp. 157–166.

[39] Mosaic Research Group, University of Washington. Mosaic Project. `http://www.cs.washington.edu/research/lis/mosaic/`.

[40] Natarajan, S., Chung, S., Paris, L., and Keshavarzi, A. Searching for the dream embedded memory. *Solid-State Circuits Magazine, IEEE 1*, 3 (Summer 2009), 34–44.

[41] Pan, Y., Li, Y., Sun, H., Xu, W., Zheng, N., and Zhang, T. Exploring the use of emerging nonvolatile memory technologies in future fpgas. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21*, 4 (April 2013), 771–775.

[42] Park, Yongjun, Park, Jason Jong Kyu, and Mahlke, Scott A. Efficient performance scaling of future cgras for mobile applications. In *FPT* (2012), Citeseer, pp. 335–342.

[43] Rouffet, Denis, and Schier, P. Convergence and competition on the way towards 4g. In *Radio and Wireless Symposium, 2007 IEEE* (2007), IEEE, pp. 277–280.

[44] Sanchez-Barbetty, M., Jackson, R.W., and Frasier, S.J. nterleaved sparse arrays for polarization control of electronically steered phased arrays for meteorological applications. *IEEE Transactions on Geoscience and Remote Sensing 50*, 4 (2012), 1283–1290.

[45] Simmons, S.J. Breadth-first trellis decoding with adaptive effort. *Communications, IEEE Transactions on 38*, 1 (Jan 1990), 3–12.

[46] Singh, Hartej, Lee, Ming-Hau, Lu, Guangming, Kurdahi, Fadi J, Bagherzadeh, Nader, and Chaves Filho, Eliseu M. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. *Computers, IEEE Transactions on 49*, 5 (2000), 465–481.

[47] Soares, R., Silva, I.S., and Azevedo, A. When reconfigurable architecture meets network-on-chip. In *Integrated Circuits and Systems Design, 2004. SBCCI 2004. 17th Symposium on* (Sept 2004), pp. 216–221.

[48] Swaminathan, Sriram, Tessier, Russell, Goeckel, Dennis, and Burleson, Wayne. A dynamically reconfigurable adaptive viterbi decoder. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays* (2002), ACM, pp. 227–236.

[49] Sylvester, Joel. Reed solomon codes. *Elekrobit., January* (2001).

[50] Tanizaki, H., Tsuji, T., Otani, J., Yamaguchi, Y., Murai, Y., Furuta, H., Ueno, S., Oishi, T., Hayashikoshi, Masanori, and Hidaka, H. A high-density and high-speed 1t-4mtj mram with voltage offset self-reference sensing scheme. In *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian* (Nov 2006), pp. 303–306.

[51] Tessier, Russell, and Burleson, Wayne. Reconfigurable computing for digital signal processing: A survey. *Journal of VLSI signal processing systems for signal, image and video technology 28*, 1-2 (2001), 7–27.

[52] Texas Instruments. Multicore DSP+ARM KeyStone II System-on-Chip (SoC). `http://www.ti.com/lit/ds/sprs865b/sprs865b.pdf`. Revised Jan 2014.

[53] Theodoridis, George, Soudris, Dimitrios, and Vassiliadis, Stamatis. A survey of coarse-grain reconfigurable architectures and cad tools. In *Fine-and Coarse-Grain Reconfigurable Computing*. Springer, 2008, pp. 89–149.

[54] Van Essen, B., Wood, A., Carroll, A., Friedman, S., Panda, R., Ylvisaker, B., Ebeling, C., and Hauck, S. Static versus scheduled interconnect in coarse-grained reconfigurable arrays. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on* (Aug 2009), pp. 268–275.

[55] Van Essen, Brian, Wood, Aaron, Carroll, Allan, Friedman, Stephen, Panda, Robin, Ylvisaker, Benjamin, Ebeling, Carl, and Hauck, Scott. Static versus scheduled interconnect in coarse-grained reconfigurable arrays. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on* (2009), IEEE, pp. 268–275.

[56] Wikipedia. Viterbi algorithm. http://en.wikipedia.org/wiki/Viterbi_algorithm.

[57] Wolf, Stuart A., Lu, Jiwei, Stan, M.R., Chen, E., and Treger, D.M. The promise of nanomagnetics and spintronics for future logic and universal memory. *Proceedings of the IEEE 98*, 12 (Dec 2010), 2155–2168.

[58] Xu, Wei, Sun, Hongbin, Wang, Xiaobin, Chen, Yiran, and Zhang, Tong. Design of last-level on-chip cache using spin-torque transfer ram (stt ram). *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 19*, 3 (March 2011), 483–493.

[59] Yiannacouras, Peter, Steffan, J Gregory, and Rose, Jonathan. Application-specific customization of soft processor microarchitecture. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays* (2006), ACM, pp. 201–210.