SmartCell: An Energy Efficient Reconfigurable Architecture for Stream Processing

by
Cao Liang

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Electrical and Computer Engineering

April, 2009

Approved:

Prof. Xinming Huang
ECE Department, WPI
Dissertation Advisor

Prof. Fred J. Looft
ECE Department Head, WPI

Prof. Berk Sunar
ECE Department, WPI
Dissertation Committee

Prof. Russell Tessier
ECE Department, UMASS Amherst
Dissertation Committee

Abstract

Data streaming applications, such as signal processing, multimedia applications, often require high computing capacity, yet also have stringent power constraints, especially in portable devices. General purpose processors can no longer meet these requirements due to their sequential software execution. Although fixed logic ASICs are usually able to achieve the best performance and energy efficiency, ASIC solutions are expensive to design and their lack of flexibility makes them unable to accommodate functional changes or new system requirements. Reconfigurable systems have long been proposed to bridge the gap between the flexibility of software processors and performance of hardware circuits. Unfortunately, mainstream reconfigurable FPGA designs suffer from high cost of area, power consumption and speed due to the routing area overhead and timing penalty of their bit-level fine granularity.

In this dissertation, we present an architecture design, application mapping and performance evaluation of a novel coarse-grained reconfigurable architecture, named SmartCell, for data streaming applications. The system tiles a large number of computing cell units in a 2D mesh structure, with four coarse-grained processing elements developed inside each cell to form a quad structure. Based on this structure, a hierarchical reconfigurable network is developed to provide flexible on-chip communication among computing resources: including fully connected crossbar, nearest neighbor connection and clustered mesh network. SmartCell can be configured to operate in various computing modes, including SIMD, MIMD and systolic array styles to fit for different application requirements. The coarse-grained SmartCell has the potential to improve the power and energy efficiency compared with fine-grained FPGAs. It is also able to provide high performance comparable to the fixed function ASICs through deep pipelining and large amount of computing parallelism. Dynamic reconfiguration is also addressed in this dissertation.

To evaluate its performance, a set of benchmark applications has been successfully mapped onto the SmartCell system, ranging from signal processing, multimedia applications to scientific computing and data encryption. A 4 by 4 SmartCell prototype system was initially designed in CMOS standard cell ASIC with 0.13 μm process. The chip occupies 8.2 mm^2 and dissipates 1.6 mW/MHz under fully operation. The results show that

the SmartCell can bridge the performance and flexibility gap between logic specific ASICs and reconfigurable FPGAs. SmartCell is also about 8% and 69% more energy efficient and achieves 4x and 2x throughput gains compared with Montium and RaPiD CGRAs.

Based on our first SmartCell prototype experiences, an improved SmartCell-II architecture was developed, which includes distributed data memory, segmented instruction format and improved dynamic configuration schemes. A novel parallel FFT algorithm with balanced workloads and optimized data flow was also proposed and successfully mapped onto SmartCell-II for performance evaluations. A 4 by 4 SmartCell-II prototype was then synthesized into standard cell ASICs with 90 nm process. The results show that SmartCell-II consists of 2.0 million gates and is fully functional at up to 295 MHz with 3.1 mW/MHz power consumption. SmartCell-II is about 3.6 and 28.9 times more energy efficient than Xilinx FPGA and TI's high performance DSPs, respectively. It is concluded that the SmartCell is able to provide a promising solution to achieve high performance and energy efficiency for future data streaming applications.

Acknowledgements

This work would not have been possible without the support and help of many people.

First of all, I want to thank my advisor Prof. Xinming Huang, whose support and guidance over the last a few years has been invaluable for this work. He was always ready to provide me with inspiring ideas and advices, and guided me to the right direction. His devotion and enthusiasm on research will affect me strongly in my future career. I would also like to thank my dissertation committee members, Prof. Fred J. Looft, Prof. Berk Sunar and Prof. Russell Tessier for their valuable time and suggestions, which significantly improved this dissertation.

I am grateful to my fellow graduate students in the Embedded Computing Lab, Wenxuan Guo, Kai Zhang, Yanjie Peng, Chen Shen, Hongkui Zhu for their friendship and supports. It was with them that made my Ph.D. a rich and enjoyable experience. I'll remember the many inspiring discussions we had that improved this project.

I would also like to acknowledge Defense Advanced Research Projects Agency (under DARPA grant W911NF-07-1-0191-P00001) and National Science Foundation (under NSF grant ECS-0725522) for the financial support.

I would like to take this opportunity to thank all my teachers and professors who kept me inspired and motivated, and broadened my perspective. Many thanks go to Robert Brown, Manager of Computational Facilities at ECE department, who helped us setup the Synopsys design tools on the server and kept them reliable and accessible all the time. I also would like to thank the administrative assistants Catherine Emmerton, Colleen Sweeney and Brenda McDonald for their kind assistance and coordinations.

At last, and most of all, I would like to express my deepest gratitude to my parents Xingwu Liang and Yarong Liu, my wife Wenxin Zhou for their love, support and continuous encouragement. My little son Eric always makes my life happy and enjoyable. I would like to say thanks to all of my family members and friends who have helped me over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Data streaming applications, such as signal processing, multimedia applications and data encryptions, are the dominant workload in many electronic systems. The real time constraints of these applications are that these devices often require relatively high performance with stringent power budget, especially for portable devices. Many other military applications, including real time synthetic aperture radar imaging, automatic target recognition, surveillance video processing, optical inspection, and cognitive radio systems, have similar needs and constraints. General purpose solutions, such as programmable digital signal processors (DSPs), are widely used in conventional data-path oriented applications due to their flexibility and ease of use. However, they can not meet the increasing requirements on performance, cost and energy in the data streaming application domains due to their sequential software execution. The application specific integrated circuits (ASICs) become inevitable a customized solution to meet these ever increasing demands for highly repetitive parallel computations. It is reported that they are potentially two to three orders of magnitude more efficient than the processors in terms of combined performances of computational power, energy consumption and cost [54]. Although ASIC can

provide the best performance for specific applications, it is not desirable for all circuitry designs. ASICs generally have fixed data flow with predefined functionalities that makes them unable to accommodate new system requirements or changes in standards. The long design cycle and high Non-Recurring Engineering (NRE) cost also become an obstacle to meet stringent cost and time-to-market requirements.

Reconfigurable architectures (RAs) have been proposed as a way to achieve a balance between flexibility and ease of use of processors, and cost/performance efficiency of ASICs. The hardware based RA implementation is able to exploit the spatial feature of the computing tasks involved in the targeted applications. It also avoids the instruction fetching, decoding, executing overhead of the software implementations, which results in a power efficiency and performance gain over general purpose processors. On the other hand, RAs maintain the post fabric flexibility to be configured, either off-line or in the real time, to accommodate new system requirements or protocol updates that is not feasible in ASIC implementations. Also, the flexibility provided by RAs can improve the fault tolerance and reliability of the designs. Design bugs can be easily fixed by loading new configuration bits, and malfunctioned circuitry can be excluded from other parts to achieve system recovery and prolong a product's lifetime.

Nowadays, field programmable gate arrays (FPGAs) are still the dominating semiconductor technology in the reconfigurable computing area. The most common SRAM-based FPGAs decompose complex logic functions into smaller ones and map them onto Lookup Tables (LUTs) or other on-chip embedded resources, such as multipliers and block memories. The island-style routing fabrics can be configured to form any desired application datapath. The bit-level fine granularity is suited to implement a large variety of functions directly onto its rich hardware resources. However, this flexibility comes at a significant cost in terms of device area, power consumption and speed, due to its huge routing area overhead and timing penalty. Furthermore, due to the fine-grained nature of most FPGAs, the compilation and configuration of FPGAs takes much longer time than those in general purpose processors. As a result, the FPGAs are mainly adopted in system prototyping

and high-end communication market, where power consumption is less of a concern.

Recognizing of these issues, several projects over the past decade have introduced more coarse-grained reconfigurable operators as the basis for reconfigurable computing architectures as summarized in [44]. Benefiting from reduced computing and routing overhead, these coarse-grained reconfigurable architectures (CGRAs) have the potential to improve FPGA area, power and energy efficiency, while maintaining high performance and flexibility.

1.2 Contributions

Fig. 1.1 shows the design objectives. In our research, we aimed at developing a computing architecture that is able to pride high performance, low power and flexibility at the same time. We proposed SmartCell as a novel CGRA system targeting applications with inherent high data-parallelism, high computing and communication regularities that are usually found in data streaming applications. Toward this end, SmartCell integrates a large number of tiny processor cores (cells) onto the same chip. The cells are interconnected with three levels of programmable switching fabrics and can be reconfigured to operate in various modes, such as SIMD, MIMD and systolic array styles. A prototype SmartCell system with 64 processing elements is designed in standard cell ASICs and is evaluated based on a set of various benchmark applications.

This dissertation makes several contributions outlined as follows:

- We present SmartCell as a successful CGRA design. The proposed architecture is discussed in detail, including the design of processing element, cell structure, on-chip interconnections, and system control and configuration schemes. Dynamic reconfigurability is also developed in two modes: coarse-grained cell broadcasting and fine-grained ID based configurations to adapt to different datapath control requirements. SmartCell is aimed to provide high steam processing capacity to achieve high performance and energy efficiency meanwhile maintaining efficient reconfigurability.

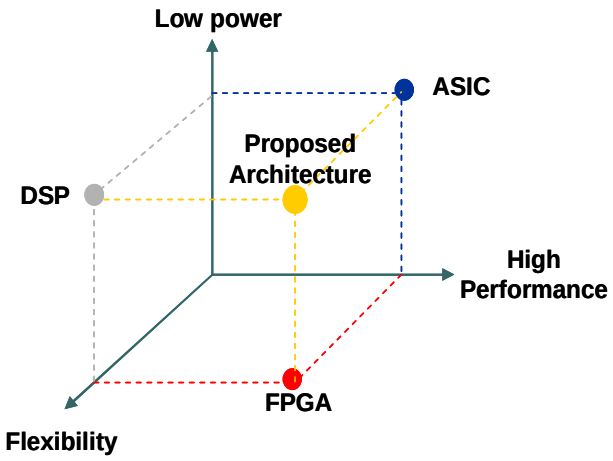


Figure 1.1: Targeted design objectives for the proposed architecture.

- A set of application benchmarks are designed and mapped onto the SmartCell prototype system. These benchmarks represent a wide range of real-time applications from signal processing, multimedia applications to scientific computing and data encryption, which also exploit the temporal and spatial parallelism achieved in SmartCell architecture. The benchmarks are simulated in hardware and are verified through software simulations. A software developing environment, named Smart_C, is proposed to help generate the configuration contexts based on application description and hardware configuration.

- This dissertation provides SmartCell performance evaluations with respect to area, power consumption, system throughput and energy efficiency. The same benchmarks are also implemented and evaluated on other computing platforms, including commercial FPGAs, DSPs, ASICs, and other CGRAs for performance comparison. To the best of our knowledge, only limited performance comparisons are available in the literature for the existing CGRA designs to exploit their architectural advantages. Our results demonstrate that SmartCell has the potential to bridge the performance and energy efficiency gap between FPGA and ASICs. For the tested benchmarks, SmartCell also shows favorable performance advantages over RaPiD and Montium CGRA designs.

- The dissertation also presents a novel parallel FFT algorithm that distributes the transform task onto multiple processor environment for parallel computing to reduce the processing time. The proposed algorithm achieves balanced workload and fixed data flow pattern across all processing units, which improves the scalability and reduces the communication overhead. The proposed algorithm can be scaled to implement FFT of any size as long as the on-chip memory fits. This parallel FFT algorithm is further mapped onto the prototype SmartCell system and evaluated against other implementations, including DSP, FPGA, NoC and MorphoSys platforms.

1.3 Dissertation Statement

The dissertation addresses the design of SmartCell for a research implementation and evaluation of an efficient CGRA system. SmartCell exploits both temporal and spatial parallelisms involved in the computing tasks and can be configured into different operation modes for different system requirements. The dissertation also presents detailed performance evaluations and comparisons that demonstrates the potential advantages the coarse-grained architecture could offer to bridge the performance efficiency gap between fine-grained FPGA and fixed function ASICs.

1.4 Outline

The dissertation is organized as follows. Chapter 2 presents an overview of the general characteristics of the targeted application domain. It also reviews the existing computing models for data streaming applications and introduces the concept of coarse-grained reconfigurable architecture. Chapter 3 presents the design details of the SmartCell system, including processing unit, cell structure, instruction format, switching fabrics, and configuration schemes. It also distinguishes SmartCell from other CGRA systems through

architectural comparisons. Chapter 4 presents the design and verification methodology in our research and evaluates SmartCell performance based on a set of benchmark applications. Although experiments show favorable results for SmartCell system, they also expose some design limitations. Chapter 5 presents the major modifications being made to the second generation SmartCell, called SmartCell-II, along with the proposed software developing environment Smart_C. The design and mapping of matrix multiplication and FFT benchmarks onto SmartCell-II is presented in Chapter 6. Chapter 7 discusses the SmartCell-II synthesis results and compares its performance with some other computing platforms. At last, we draw the conclusions and future work in Chapter 8.

Chapter 2

Background

In this chapter, the general characteristics of stream processing are described. After that, we talk about the existing computing models for stream processing applications. Finally, the concept of coarse-grained reconfigurable architecture is discussed as a mechanism to achieve better performance while improving high energy efficiency.

2.1 Data Streaming Applications

In data streaming applications, a set of data inputs is streamed into the computational units to perform a series operations, called kernel functions. Typically, one kernel function is applied to all elements in the data stream, which is called uniform streaming. Example application domains include digital signal processing, multimedia applications, scientific computing, and data encryptions. Each of these applications shares important characteristics[74]: computing intensity, parallelism and data locality, which can be explorted to improve data processing speeds. In real time, the workload involved in data streaming applications often require high computational performance, low power consumption and high energy efficiency, which build the key goals for stream processing architecture designs. Each of these characteristics will be briefly explained below.

2.1.1 Computing Intensity

The computing intensity refers to the fact that data streaming applications require a larger number of arithmetic operations for each memory reference when compered with traditional general purpose applications. In [74], Rixner studied the computing intensity for data streaming applications, including stereo depth extractor, video encoder/decoder, polygon renderer and matrix QR decomposition. The computing intensity factors are ranging from 58 to 473 arithmetic operations per memory reference for these applications. On the other hand, it is studied in [58] that for traditional desktop applications defined by SPEC2000 benchmark, memory accessing accounts for up to 80% of total processing time, which leads to a computing intensity factor of less than 2 operations per memory reference. This data demonstrate that the traditional general purpose processor (GPP) will likely not meet the computing intensity requirements of data streaming applications from the fundamental architecture point of view.

2.1.2 Parallelism

Another important characteristic for data streaming applications is computing parallelism, which means the computational task has the potential to be distributed across multiple processing components. In general, computing parallelism can be classified into two categories: temporal parallelism and spatial parallelism. In temporal parallelism, the pipeline technology is adopted at either the instruction level or at the task level, in which the instruction code or computing task is separated into multiple stages. Several instructions or tasks are overlapped in the same pipeline at different stages to improve system throughput. On the other hand, spatial parallelism distributes the data and tasks onto different computational nodes to process in parallel. In data streaming applications, data-level parallelism (DLP) can often be exploited since there are no data dependencies between different input data blocks. In this case, single instruction multiple data (SIMD) computational style is widely used to apply the same kernel functions to different data elements.

Similarly, task-level parallelism (TLP) are usually exploited to execute different application threads in data streaming applications. In many cases, the computing task involved in stream processing can be decomposed into multiple stages. These stages can be overlapped into multiple computing resources to concurrently process different data sets through the pipeline. Given plentiful parallelism, it is a key requirement that the computing architecture design for data streaming applications should be able to efficiently exploit and map the parallelism onto available hardware resources.

2.1.3 Data Locality

Data locality usually refers to the fact that the input data and coefficients can be shared or reused among multiple processing components. The intermediate results calculated in one processor can be passed directly into the next processing element through tightly coupled interconnection fabrics, which do not require frequent processor-memory communications compared to the traditional single processor Von Neumann architectures [87]. Similarly, the input data can be propagated and reused temporally among multiple kernel functions. Data locality reduces the on-chip memory utilization and requirements, which in turn has the potential to improve the system performance, chip area efficiency and energy efficiency.

2.1.4 Regular and Deterministic Data Flow

The last characteristic of data streaming applications is regular and deterministic data flow [20]. For most data streaming applications, the datapath involved in the kernel functions follows specific regular pattern, which makes it possible to use a relatively simple on-chip communication scheme to achieve high area and energy efficiency. Also, data streaming applications often have deterministic computation and communication throughout the entire processing stage that can be scheduled at the compilation time. This alleviates the stress of dynamic control and data routing requirements.

2.2 Computing Models for Stream Processing

After discussing the characteristics, proper computing models are needed to be designed to provide high performance and, more importantly, energy efficiency for the targeted application domain. Currently, how to achieve high performance while maintaining energy efficiency has become the key challenges in this field. Innovative computer architecture designs are necessary to be address for these challenges. This section reviews the existing computing models that could be potentially used in stream processing. A summary of key features for some major computing models is listed in Table 2.1. Each of these generic models (processor, DSP, FPGA, GPU) will be discussed below.

Type	Product	Architecture	Frequency	Word width
CMP	AMD Athlon 64X2	Deep pipeline	2.4 GHz	64-bit
DSP	TI C64X	8-way VLIW	0.5∼1 GHz	32-bit
FPGA	Xilinx Virtex 4	Mix Granularity	∼500 MHz	4-bit LUT, 18-bit DSP
GPU	NVIDIA GTX280	SIMD	1.2 GHz	32 or 64-bit

Table 2.1: List of the key features of some existing computing models

2.2.1 Microprocessor

Microprocessors are targeted at high performance, memory intensive general purpose applications and are widely used in desktop and laptop computers. Traditionally, deep pipeline and process shrinking are the key techniques to improve system performance, but have become more and more challenging nowadays. As the inserted Flip-Flop delay is comparable to the combinational logic delay, the benefits from deep instruction level pipeline (ILP) becomes less and less significant for frequency improvement. Process shrinking is an efficient way to reduce the transistor size for both performance and integration capacity improvement. However, it is predicted that process shrinking will meet its limitation in the near future either due to the physical limits when the transistor size approaches the size of an atom or due to fabrication process limits. The high power dissipation is another constraint for microprocessor to be used in data streaming applications. The thermal

design power for modern CPUs are normally ranging from tens of Watts to hundreds of Watts [1].

2.2.2 Digital Signal Processor

Programmable digital signal processors (DSPs) are designed to provide high computation performance for digital signal applications. Compared with microprocessors, DSPs usually provide more dedicated computing resources, such as multipliers and accumulators, and higher memory access bandwidth. More recently, some new features are also introduced to DSP architecture, which include very long instruction word (VLIW) structure and hierarchical cache memories to improve computing capacity. For example, TI's 8-way VLIW TMS320C64X DSPs [19] can achieve up to 8k mega instructions per second (MIPS) performance at a power budget of 6 W.

DSPs provide an energy efficiency gain compared with microprocessors due to shorter pipeline length, less communication overhead and more parallel favorable architectures. However, the VLIW DSP shares the control structure and uses a global register file for data sharing among different processing units, which makes it hard to be scaled to include a large number of processing units. Besides ILP, DSPs can only exploit limited amount of DLP, similar to microprocessors. It is studied in [54] that they are potentially two to three orders of magnitude less efficient than application specific circuits in terms of area and energy consumption.

2.2.3 FPGAs

Traditional FPGA architectures use static streams to configure the functional units and routing resources to perform user specifications. The data parallelism and flexible on-chip communications are essential to meet the high performance requirements of the computations being performed. Due to direct mapping of application tasks onto hardware resources, FPGAs are able to complete one operation in a single clock cycle, which avoids

the instruction fetching, decoding, executing overhead as of in the software processors. The most common SRAM-based FPGAs decompose complex logic functions into smaller ones and map them onto the Lookup Tables (LUTs) or other on-chip embedded resources. Most FPGAs use the island-style routing fabrics for on-chip data communication. The bit-level fine granularity is suited to implement a large variety of functions directly onto its rich hardware resources. However, this flexibility comes at a significant cost in terms of area, power consumption and speed, due to its huge routing area overhead and timing penalty. Furthermore, due to the fine-grained nature, the compilation and configuration of FPGAs takes much longer than those in general purpose processors.

In recognition of these problems, commercial FPGA vendors have introduced more coarse-grained components in addition to the fine-grained LUTs in their newly developed FPGAs. These components include dedicated logics for arithmetic multiply-add functions and some on-chip memory blocks. For example, the Virtex-4 and 5 series [2, 3] are among the latest Xilinx FPGAs, which have a mixed granularity of basic logic cells with coarse-grained DSP slices (DSP48) to enhance the signal processing capacity and power consumption performance. Similar features can be also found in Altera's Stratix II FPGAs [4]. In addition, 6-input and 8-input LUTs are also introduced to the Virtex 5 and Stratix II FPGAs, respectively, as the substitute for the traditional 4-input LUT. These features help to reduce routing overhead and ease the configuration process. But the SRAM-based FPGAs have some fundamental limits that hamper them becoming the mainstream computing media for data streaming applications. The system configuration SRAM cells are power and area intense. The rich on-chip programmable interconnection provides flexible routing ability at a cost of high power consumption and large die area. It is studied in [62, 85] that the programmable interconnection account for up to 60% to 70% total power consumption of FPGA. These FPGAs also consume about 14 times more dynamic power and is about 35 times larger than equivalent ASICs on average when only logic elements are used [60, 61]. Furthermore, FPGAs do not support instruction sequencing. As a result, it is difficult or very costly to make configuration changes on the fly.

In reality, FPGAs are mainly used for system prototyping and high-end communication markets, where power consumption is less of a concern.

2.2.4 Systolic Arrays

Another approach to data streaming processing is to use systolic array architectures [23], in which the data processing units (DPUs) are arranged in an array structure with local connectivity. The operations of DPUs are scheduled by regular data flows in a pipelined manner. The input/output pipeline and the concurrent data execution supports extremely high throughput. Systolic arrays also provide high system scalability due to their regular structure. However, systolic arrays have some drawbacks. For example, due to the fixed configurations, the cost of introducing new pipeline patterns in traditional systolic arrays is very high. In addition, the process synchronization may be very complicated in both hardware and software in a systolic array implementation.

2.2.5 Multiple Processors on a Chip

Chip multi-processor (CMP) has been proposed as a general purpose processor architecture design to achieve higher performance with even lower frequency through parallel computing. Introduced first by AMD's Athlon 64 2X [16], CMP integrates a relatively small number of microprocessors onto the same chip. With a lower operating frequency and supply voltage, CMP is able to provide higher energy efficiency compared with single processor systems. However, traditional CMPs require extensive control logic and large global memories for even general purpose applications, which in turn results in poor scalability. Furthermore, existing microprocessor architectures cannot take advantage of the computing intensity and parallelism inherent with stream processing, since they are primarily optimized for ILP and have limited computing capacity for parallel processing.

More recently, several multi-core systems have been developed that integrate a large number of simplified processor cores onto a single chip, which are targeted at comput-

ing intensive applications, such as video/image processing, gaming, super computing and network applications. Compared with CMP, multi-core system usually involves more processing units with a specific communication structure among them. For example, Intel 80-core system [86] integrates 80 tiles in a 10 by 8 2D mesh structure, with each tile contains one computing element and a 5 port router unit. A dynamic message passing protocol is implemented to provide high speed and robust Network-On-a-Chip data communications, which is much more scalable than today's CMP interconnect. It is reported that Intel's 80-core can achieve a peak performance of 1.28 tera floating point operations per second (TFLOPS) at 181 Watts. Other multi-core systems include Stream Processor's Storm-1 [55], IBM/Sony/Toshiba's CELL [73], Mathstar's FPOA [9], and RAPPORT's KC256 [12]. Multi-core structure is a promising solution to achieve high computing capacity for data streaming applications. But based on the processor structure, the multi-core systems often have high power consumption, which usually can not meet the stringent power consumption requirement especially for portable devices.

2.2.6 Graphics Processing Unit

The graphics Processing Units (GPUs) were originally designed to provide a dedicated graphics rendering device for personal computers and game consoles. A GPU integrates a number of graphic primitive operations, such as interpolation, rasterization, shader processing, and texture mapping. Because most of these computations involve matrix and vector operations, researchers are adapting computationally intensive general-purpose algorithms to run on GPUs, which forms the idea of general purpose GPU (GPGPU). The applications that achieve the best performance on GPU are typically those with high arithmetic intensity and relatively low memory access requirements. Modern GPUs feature a large number of tightly integrated streaming processors to achieve high computing capacity. For example, NVIDIA's GeForce 8800 GTX [5] has 128 on-chip streaming processors with a peak performance of 518 GFLOPS with about 185 Watts power consumption.

Advances in the programming model and tools has become a key challenge for GPGPU to balance high level program flexibility and low level hardware access. Although various computing engine tools have been developed by GPU vendors to provide both low level and higher level APIs, including NVIDIA's CUDA [17] and AMD's BROOK+ [10], these tools can not be easily used for stream computing without expertise knowledge. AMD proposed another aggressive architecture, called FUSION [6], to combine the CPU and GPU onto a single chip, with the CPU responsible for the sequential tasks and system controls and GPU for the parallel computing tasks. In the FUSION processor (available in 2011), the applications can be specified in general programming languages and a unified compiler can be used to partition the workload into the CPU or GPU either statically or on the fly. Despite these advantages, similar to multi-core systems, the power consumption of GPUs still remains a concern if they are for the adoption in data streaming applications and meet stringent energy efficiency requirements.

Some other models are also available for high performance computing, such as vector processor [75, 59], and SIMD processor [72]. However, the power efficiency and system flexibility are still open issues when applied to data streaming applications. On the other hand, the logic specific ASICs matches well to the stream processing characteristics, but the high performance and energy efficiency is achieved at a cost of non post-fabrication flexibility. As a result, there is an urgent demand to exploit an innovative computing architecture to bridge the performance efficiency gap between fixed function logics and programmable processors.

2.3 Coarse-Grained Reconfigurable Architecture

Reconfigurable computing architectures have been proposed as a way to bridge the performance gap between ASICs and GPPs, while maintaining the flexibility and ease of use of processors. Nowadays, the FPGAs are still the dominating semiconductor technology in the reconfigurable computing area. But as discussed before, the bit level fine-grained

FPGA architecture results in a significant cost in terms of area, power and speed due to its huge routing area overhead, poor routability and timing penalty [37].

On the other hand, coarse-grained reconfigurable architecture includes word-level (16-bit) or double word-level (32-bit) components as the basic computational and communication units [40, 69, 79, 80, 83, 68]. Benefiting from much lower computational and routing overhead, coarse-grained reconfigurable architectures (CGRAs) have the potential to improve upon FPPGA power and energy efficiency while providing high system performance. CGRAs take advantage of the fact that many computing tasks operate on multi-bit data with intensive computation requirements. The involved computing component does not need to be as complex and powerful as microprocessors. The simplified functional blocks can usually achieve higher area and power efficiency and can improve system scalability.

Based on the on-chip hardware resources, CGRAs can also be categorized into heterogenous system and homogenous system. Heterogenous CGRAs integrate a mixture of computing blocks (ALU, multiplier, specific functions, and etc.) and memories onto the same chip. Homogenous CGRA use the same computing blocks and unified communication structure throughout the entire chip. The computing and communication regularity of the homogenous CGRA has the potential to improve system scalability and to ease the design and testing efforts. This partially explains the trend in CGRA research to move away from heterogeneity to homogeneous architectures [61].

Within the context of a houmogeneous CGRA, this dissertation presents SmartCell as a novel CGRA system, which integrates a large number of tiny processor cores (cells) onto a single chip. A three level hierarchical interconnection network is developed for data exchange among the cell units. It can be configured to operate in various computing styles such as SIMD, MIMD, and systolic array fashions, which is well suited for the targeting data streaming applications. Based on evaluated benchmarks, the SmartCell performance is analyzied and compared with other computing architectures, including ASICs, FPGAs, DSPs and some other CGRA systems.

2.4 Summary

In this chapter, we presented the characteristics of the targeted data streaming application domain for our research. Existing computing models were then reviewed with their possibilities and limitations relate to stream processing. At last, we briefly discussed the CGRA concept and its potential to achieve high computing capacity and energy efficiency.

Chapter 3

SmartCell Architecture

A novel coarse-grained reconfigurable architecture, called SmartCell [64], is proposed and designed in our work, mainly targeted at applications with inherent data-parallelism, and high computing and communication regularities. SmartCell integrates a large number of processing units on the same chip, along with configurable interconnection fabrics. The computing tasks can be distributed across different processing elements (PEs) to achieve task/data level parallelism for high performance.

In this chapter, we extract the key features of the proposed SmartCell system, followed by the architecture design details, including processing element design, on-chip interconnection design and configuration schemes. The comparison with some other CGRA systems is also presented in this chapter.

3.1 Architecture Overview

Fig. 3.1 depicts the components and organization of the integrated SmartCell architecture. In a typical SmartCell architecture, a set of cell units is organized in a tiled structure. Each cell block consists of four processing elements (PEs) along with the functional control and data switching fabrics. The PE can be configured to perform basic logic, shift and arithmetic functions with 16-bit granularity. Multiple PEs can be chained together

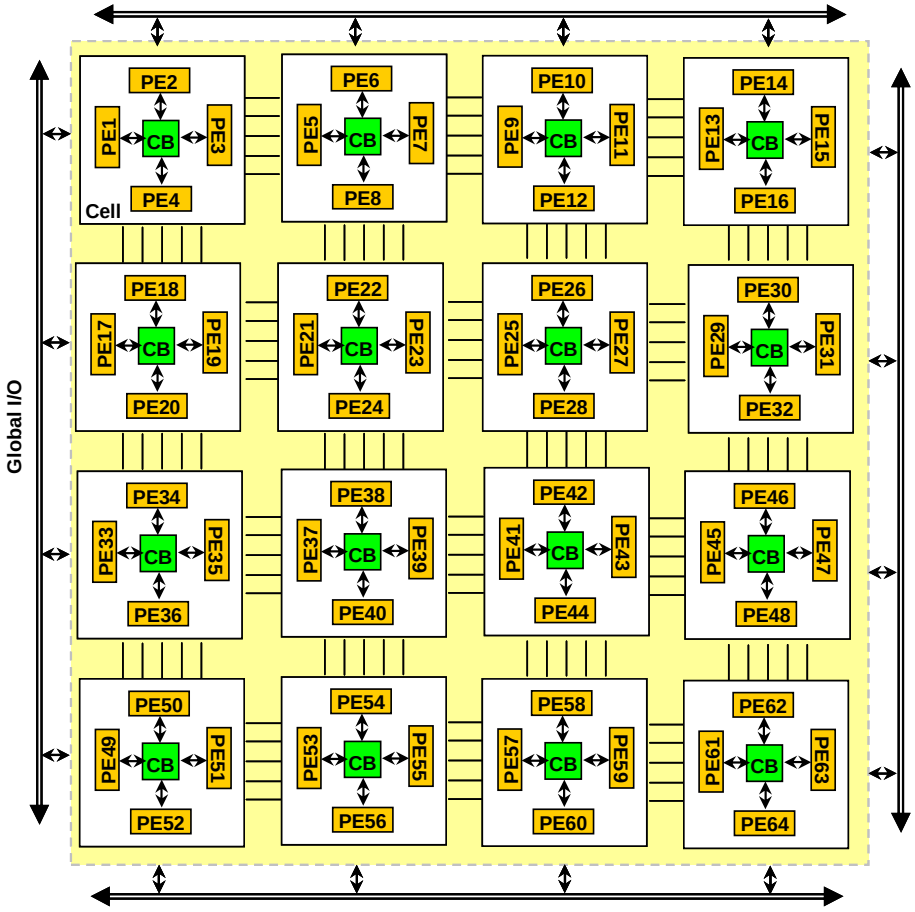


Figure 3.1: Overview of the SmartCell architecture. The SmartCell architecture is featured in a 2D tiled structure that consists of cell units, layered interconnection networks and high speed global data I/O.

for more complex tasks. A three-level layered interconnection network is designed for the on-chip processer communications, which includes fully connected crossbar unit in each cell, nearest neighbor connection among adjacent cells and a hierarchical concentrated mesh (CMesh) network for non-adjacent cells. Distributed instruction memories are also designed for each PE to store the configuration contexts for both computing and communication. A serial peripheral interface (SPI) is designed that chains the instruction memories in a linear array fashion to efficiently load instruction contexts into active processing units. These design aspects will be discussed in details in the following sections.

3.2 Key Features

Several key features distinguish SmartCell from other reconfigurable architecture designs. Fig. 3.2 shows the SmartCell key features and their expected benefits, which serves as our initial design considerations.

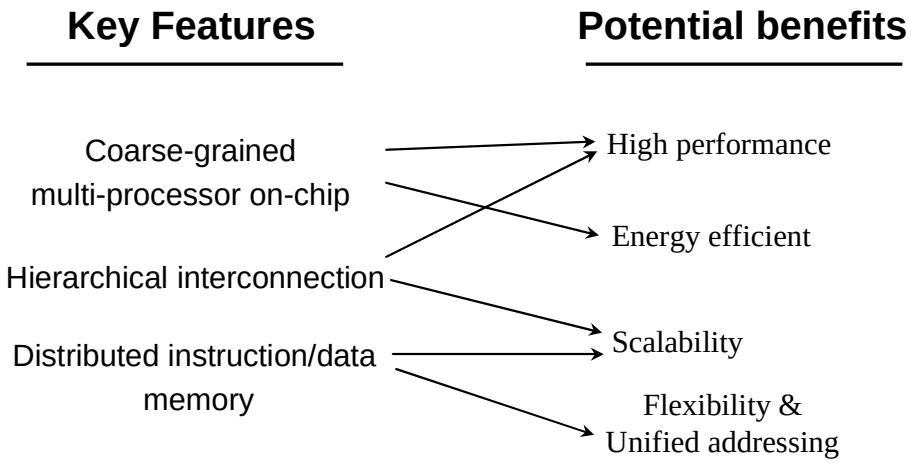


Figure 3.2: Key features and potential benefits of SmartCell architecture.

Some important features of the SmartCell architecture are summarized as follows:

- Coarse-grained multi-processor on-chip:

  Integrating multiple processors on the same chip has the potential to partition tasks onto different on-chip computing resources and to process them in parallel for high system performance. The temporary results from one processor can be forwarded directly to the next one through tightly coupled interconnections, which reduces the on-chip memory utilization and requirements. The coarse-grained computing and communication components are designed in SmartCell to avoid high execution overhead compared with fine-grained architecture. Thus SmartCell has the potential to achieve high performance and high energy efficiency.

- Hierarchical interconnections:

  As CMOS technology scales down, interconnect has become an increasingly important issue for circuit design due to its increased impact on system delay and power

consumption. Shared bus connections with high bandwidth are usually adopted in modern multicore CPU designs. But the lack of scalability and high power consumption make it not favorable for data streaming applications. In our design, we limit the communication flexibility based on different data locality levels. Each PE has the full visibility of the temporary results from all PEs in the same cell. On the other hand, cells are grouped into clusters with shared switching components to provide limited connection among non-adjacent cell units. This hierarchical interconnection efficiently alleviates the long wire delay impacts meanwhile maintaining good scalability for different system dimensions.

- Distributed instruction and data memories:

  To fully expand the on-chip resource utilizations, distributed instruction memory is attached to each PE for both computing and communication configurations. In our experiments, a relatively small number of instructions in a memory was found to be enough for most targeting DSP and data streaming applications due to their regular control and data flow characteristics. Besides the instruction memory, it is studied in [18] and [71] that the on-chip memory can easily contribute to more than 60% of the total chip area for modern processor designs. Thus the area, performance and energy efficiency can be improved significantly if the utilization of the on-chip data memory can be minimized. In our design, limited register banks or data memories (in SmartCell-II) are distributed into each processor. This is well suited to our targeted stream processing applications, characterized with continuous data flow from one processor to another without much data feedback and reutilization. The distributed memory structure is adopted in our design to provide unified addressing space and high scalability.

- Deep pipeline and parallelism:

  Two levels of pipeline are exploited by SmartCell - the instruction level pipeline (ILP) in processing element and the task level pipeline (TLP). Data parallelism is

also achieved in SmartCell to concurrently execute multiple data streams, which in combination ensures a high computing capacity.

- Flexibility:

  Due to the rich computing and communication resources, numerous computing architectures can be mapped onto the SmartCell architecture, including SIMD, MIMD, and 1D or 2D systolic array structures. This also expands the range of applications that can be implemented by SmartCell.

- Dynamic reconfiguration:

  By loading new instruction codes into the configuration memory through the SPI structure, new operations can be executed on the desired PEs without interrupting other PEs. The number of PEs involved in the application is also adjustable for different system requirements.

- Fault tolerance:

  Fault tolerance is an important feature to improve production yields and to extend device lifetime. In the SmartCell system, defective cells, caused by manufacturing fault or malfunctioned circuits, can be easily turned off and isolated from the functional ones to achieve good fault tolerance.

- Hardware virtualization:

  In our design, distributed context memories are used to store the configuration signals for each PE. The cycle by cycle instruction execution supports hardware virtualization that is able to map large applications onto limited computing resources, which is not feasible in traditional fixed context systems.

- Explicit synchronization:

  A program counter (PC) is designed to schedule instruction execution time for each PE on the fly. Variant delays are also available for input/output signals inside each

PE. Therefore, the SmartCell can provide explicit synchronization that eases the exploration of computing parallelisms.

3.3 Cell Unit and Processing Element Design

The reconfigurable cell units build the key components of SmartCell system, which are aligned in a 2D mesh structure as shown in Fig. 3.1. Each cell consists of four identical PEs. The PE is composed of an arithmetic unit and logic unit, I/O muxes, instruction controller, local data registers and instruction memories, as shown in Fig. 3.3. It can be configured to perform basic logic, shift and arithmetic functions. The arithmetic unit takes two 16-bit vectors as inputs for basic mathematic functions to generate a 36-bit output without loss of precision during multiply-accumulate operations. It also includes some logic and shift operators, usually found in targeted data streaming applications. The basic operations supported by SmartCell processor are listed in Table 3.1. Multiple PEs can be chained together through the programmable on-chip connections to implement more complex algorithms.

	Basic operations
Arithmetic Unit	add, sub, mult, MAC, abs sum
Logic Unit	and, or, not, xor, nand, compare, etc.
Shift Unit	shift right, shift left, circular shift

Table 3.1: List of basic operations supported by SmartCell processors

An up to 4-stage pipeline structure is developed in each processor, as denoted in different colors in Fig. 3.3. The *Src* select stage inputs data from the on-chip connection calculated by other PEs or itself and stores the data into its local register banks. The execution stages (*Exe 1* and *Exe 2*) occupies two clock cycles for basic multiply-add and other logic operations. The *Des* select stage selects the output result and sends it back to the on-chip interconnections. Unlike traditional pipelined processor design, the pipeline stages are not fixed in SmartCell. The bypass path can be selected in every stage except

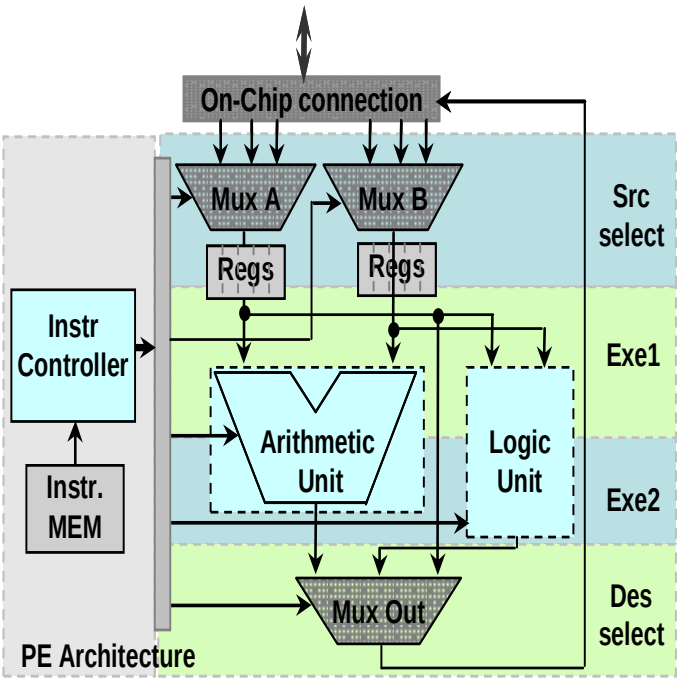


Figure 3.3: Processing element architecture. The PE component can be configured to perform 16-bit basic arithmetic operations, including logic, shift, adder, multiplier, and etc. An instruction controller is designed for the cyclic configuration of the computational units and data flows.

for *Src* select to allow fast passing through of input data or intermediate results to the next operating unit to reduce unnecessary processing delays. The traditional decoding stage is replaced by an instruction controller, which generates all control and scheduling signals in parallel with the 4 pipeline stages. An instruction code, pre-stored into the instruction memory, is loaded into the instruction controller on a cycle by cycle basis to provide both functionality and datapath control for a specific algorithm. Additionally, the instruction code can be dynamically reconfigured in various modes to adapt to different application requirements. Therefore, SmartCell is able to provide comparable energy efficiency as an ASIC while maintaining dynamic programmability as a DSP.

The instruction code is designed in a 64-bit frame format, as listed in Table 3.2. A 9-bit program counter control (PC control) section is used to indicate execution time of the current opcode, next instruction address and valid memory ranges for active instruc-

	64 bits/instruction code					
# of bits	9	20	7	10	11	7
Format	PC *Control*	Datapath *Control*	I/O *Delay*	Operation *Control*	NoC *Control*	*RESV*

Table 3.2: Frame format of the instruction code

tion codes. The datapath and operation control signals specify the configuration of data flow and computing units, while the I/O delays are used for synchronization scheduling among multiple computing units. An 11-bit Network-on-Chip (NoC) control signal is designed to configure the on-chip communication network. A 7-bit undefined section is reserved for future functional extension. New instructions are able to be input to the computing processor in pipeline with current ones. The instructions are accessed in a cyclic manner that supports periodical execution of a set of operations. In our first SmartCell implementation, a 20 by 64-bit instruction memory block is attached to each PE.

In a cell unit, four PEs placed in the east, west, south and north directions form a quad structure with a fully connected crossbar switch box located at the center, as shown in Fig. 3.4. The crossbar network supports arbitrary non-blocking connections among PEs in the same cell. Instruction memories are attached to each PE and are chained in a linear array fashion by serial peripheral interface (SPI) for configurations. The data exchange controls are stored in the instruction memory and are changed only upon loading of new instruction context.

3.4 Configuration Scheme

As discussed in previous section, the instruction code determines what operation each PE performs and how data is routed among multiple PEs. The initial instructions are loaded to the memory at compile time and can be updated to accommodate new applications or performance requirements at run time. A serial peripheral interface (SPI) is designed to configure the instruction memories, as shown in Fig. 3.4. In this structure, the instruction

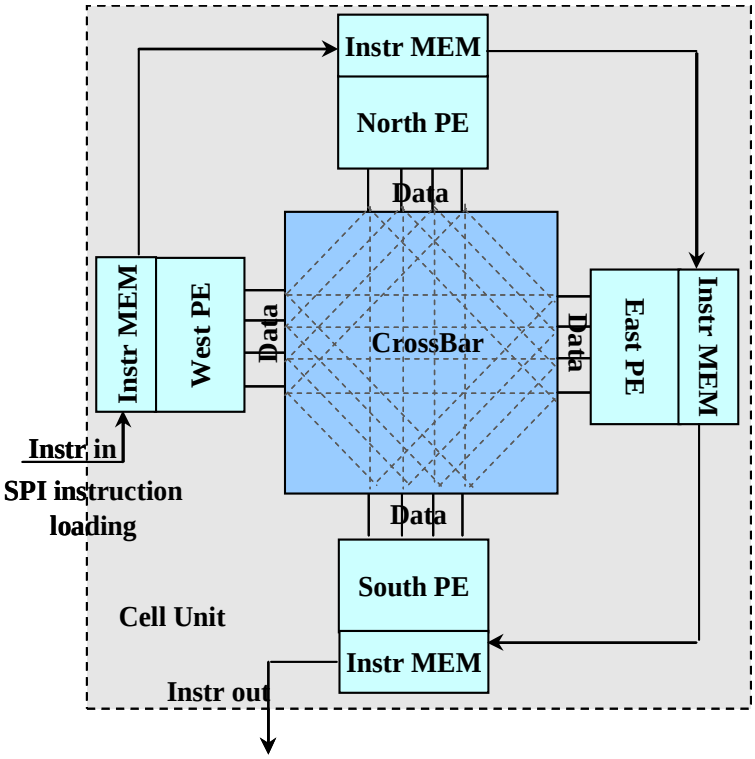


Figure 3.4: Cell unit structure. Each cell consists of 4 PEs and 4 instruction memories in pairs. A configurable crossbar is designed for intra-cell data exchange. The SPI structure is designed for instruction loading and dynamic reconfiguration.

memories are linked in a ripple array fashion with the inputs and outputs chained one to another. At cell level, each cell is able to receive one instruction code from the previous cell and forward it to the next cell after local propagation. By this means, only one unified configuration port is exposed to the outside world, which eliminates a large amount of global configuring wires to provide better performance and scalability. In the second generation of our SmartCell (SmartCell-II), we introduce two dynamic configuration modes to mitigate the unbalanced configuration delay, which will be discussed in greater details in Chapter 5.

The initial configuration procedure is depicted in Fig. 3.5. The instruction contexts are loaded into the first PE's instruction memory and is then shifted down to the second one and so on. This procedure stops after the last active PE is configured. The run time reconfiguration can be achieved by the same SPI structure, as shown in Fig. 3.6. The new instruction code and the ID of the PE to be configured are sent into the SPI instruction memory chain. The PE bypasses the information to the next one if the transmitted ID doesn't match its own ID. This procedure continues until it reaches the desired PE. By this means, only the execution of the reconfigured PE is temporarily suspended. The other PEs remain unaffected and keep operating during the reconfiguration procedure. A potential limit for this reconfiguration scheme is the unbalanced configuration delay of the PEs along the SPI chain: the nearer to the input port, the faster can the configuration be done. A variety delay ranging from 1 to 64 clock cycles is observed to load a new instruction code into a PE for a 4 by 4 SmartCell system. In SmartCell-II, a cell broadcasting and memory partitioning scheme is proposed and implemented to overcome this limit. For the sake of simplicity and proof-of-work, these features were not implemented in the first version of SmartCell implementation.

Fig. 3.7 depicts the dynamic control flow for a processing unit. At run time, a configuration context is loaded into the instruction register and is then partitioned into interconnection and functionality controls. In general, each instruction selects two operands from local register banks or network inputs. The basic arithmetic, logic and shift opera-

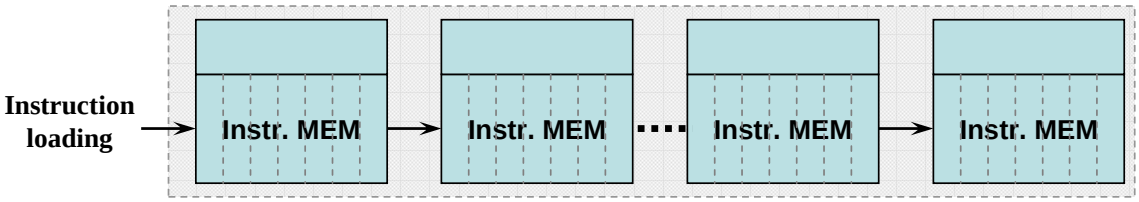


Figure 3.5: Instruction format of SmartCell architecture.

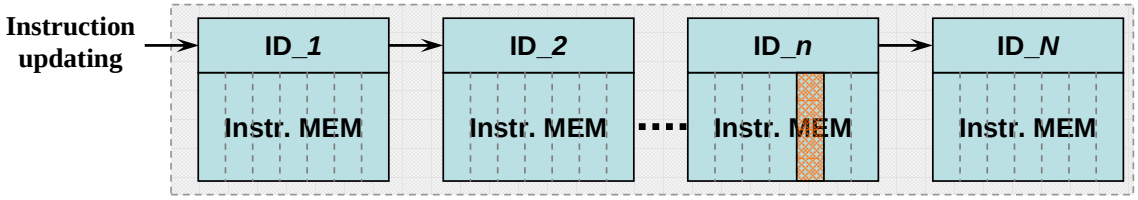


Figure 3.6: Illustration of dynamic reconfiguration in SmartCell.

tions are then executed concurrently on these operands. Finally, one result is selected and forwarded to the destination. Cyclic data flows can be configured through the looping of instructions in the memory.

3.5 Interconnection Design

3.5.1 Hierarchical on-chip Connection

As the CMOS technology scaling down, interconnect has become an increasingly important issue for integrated circuit design. In many signal processing applications, the system throughput is significantly affected by communication costs. The design of efficient data exchange scheme has become a key feature for high performance systems. Shared bus connections with high bandwidth are usually adopted in modern multicore CPU designs. The bus topology is simple, but the lack of scalability and high power consumption and timing penalty make it not attractive in our design. Other solutions are available for on-chip switch topology, such as fully connect crossbar and island-style mesh networks. The cross-bar network provides the flexibility to connect any components in the network with limited transfer delays. Despite these advantages, crossbars suffer from high silicon area

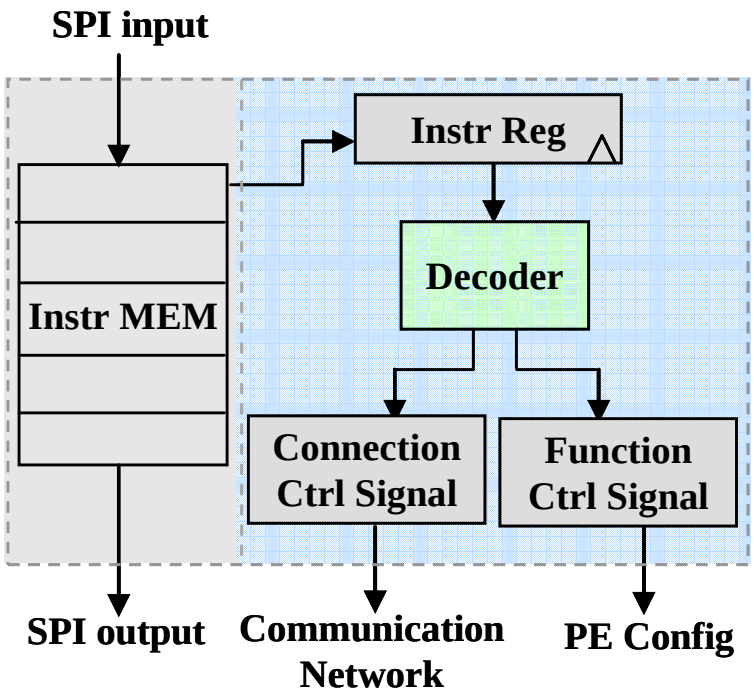


Figure 3.7: Diagram of configuration controller architecture. The control signals for the PE functionality and the data communications are decoded based on the current instruction code stored in the instruction register. The SPI links the instruction memory in ripple array style for instruction loading and updating.

costs, high power consumption and low scalability. On the other hand, island-style mesh networks are often used in FPGAs, in which each computing unit is attached with its own switch fabrics to transmit/receive data or to relay data to adjacent nodes. The tile-based mesh network offers regular structure and is easy to scale. It is widely adopted by many other reconfigurable architecture designs, including RAW [83], Trips [53], and AsAP [91]. However, intermediate relays are necessary for data transfers with larger than one hop distance in these systems, which involves longer delays and relatively complex control and synchronization logics. The proper application mapping scheme to reduce long data switching has become a crucial aspect in mesh only systems, which in turn also reduces system flexibility. In realizing of these facts, a mixed hierarchical routing structure with three-level networks is designed in SmartCell: the fully connected crossbar unit for intra-cell data exchange, the static nearest neighbor connection for inter-cell communications, and the reconfigurable modified CMesh network for concurrent data communication among non-adjacent cell units.

Crossbar Inter Cell Connection

Initially, a centralized shared register memory (SRM) block were designed for the intra-cell communications. But it was abandoned due to its high area and power costs and complex memory access controls. In the current design, the PEs and instruction memories are placed at the four edges in a cell. A fully connected crossbar unit is able to provide an efficient non-blocking connection for data exchanging. Compared to the SRM implementation, the control logics are substantially simplified in the crossbar connection, which in turn results in better timing and area performance.

Intra Cell Nearest Neighbor Connection

In our system, the homogeneous cell units are tiled in a 2D mesh structure. Thus the adjacent cells can be connected directly through short wires. Since four PEs in a cell are placed at four edges, each PE can be directly linked to the nearest PE located in the

adjacent cell, as shown in Fig. 3.1. This static network supports single cycle bi-directional data transmission of 2 16-bit and 1 36-bit signals between connected PEs. These signals are aligned with the cell's internal signals and are later sent out to the PE's inputs. Thus no extra synchronization process is involved. This low latency and self synchronization feature is critical to exploit the task level parallelism among cell units in many multimedia and signal processing applications.

Hierarchical CMesh Network

Besides local connections, it was realized that some applications also require dynamic data exchanges between nonadjacent cells, such as Radix-2 FFT, 2D DCT. After examining the major existing on-chip interconnection techniques, a modified CMesh network was adopted in our SmartCell architecture. It is studied in [22] that the CMesh network has the potential to provide the best performance in terms of average latency and network efficiency among other NoCs, including Mesh, Torus, and FTree. As shown in Fig. 3.8(a), the modified CMesh segments the network into clusters, with 4 cell units sharing the same switching component. The switch architecture, depicted in Fig. 3.8(b), is designed to connect four local cells with adjacent cluster networks. Each cell in the same cluster can input(output) a 36-bit signal from(to) the switch fabric to form a local I/O interface. The switch component also receives two sets of 4 36-bit inputs from horizontal and vertical directions, and outputs the same amount of data in these two directions. A routing arbitrator component is designed to dictate the proper data transmission that can be configured by the NoC control bits in the instruction code. A simple Dimension Order Routing (DOR) is implemented to route data firstly in one direction and then in another for multi-hop data transmissions. Because no closed loop can be generated, the DOR scheme guarantees no traffic contention exists. Instead of arranging the routers in a ring style as in traditional CMesh network, high level routers that connect four local routers are designed and chained together to form another CMesh network. It is so called hierarchical CMesh network. This hierarchical CMesh network reduces the number of long wires compared

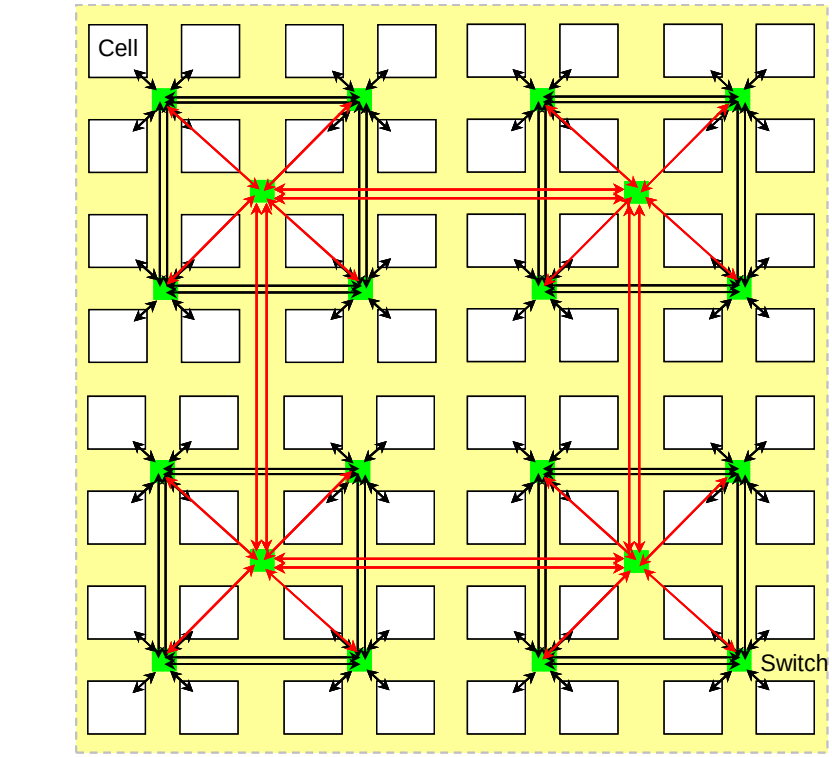
with traditional CMesh. Also, the same routing components can be easily added to or reduced from the SmartCell architecture for system scalability. In our design, each cell can receive a 36-bit signal through the CMesh network every clock cycle, which achieves a single hop system throughput of 57.6 Gbits/s for a 4 by 4 SmartCell operating at 100 MHz.

3.5.2 Propose of a Dynamic Routing Scheme: Adaptive First Routing

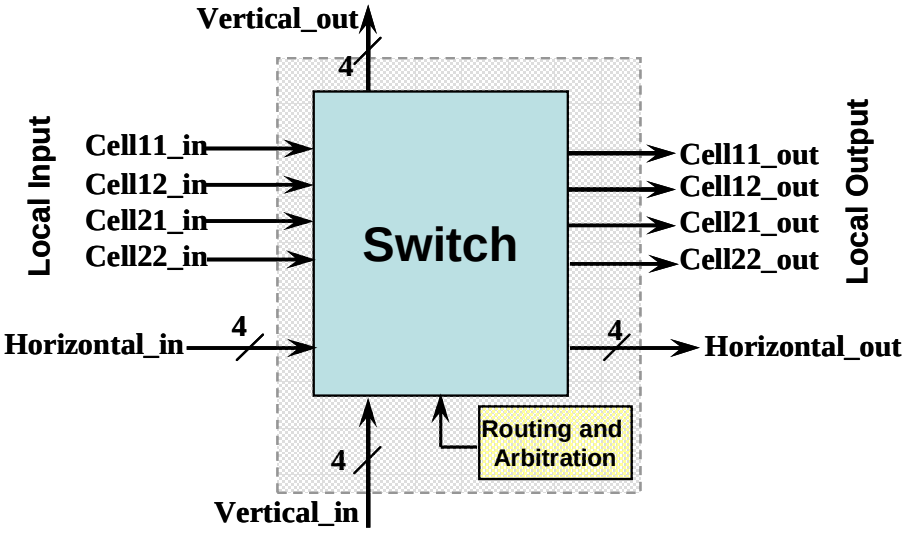
The routing protocols for on-chip data communications can be divided into two categories: Static Routing and Dynamic Routing. The data flow for static routing is decided at compilation time with only one path available from source to destination. On the other hand, dynamic routing supports multiple paths from source to destination. The actual data path in use is decided at run time based on network conditions. The key advantage of the static routing is that for applications with predictable traffic, it can provide an efficient solution with low redundancy and small area and communication delays. Static routing are suited for most DSP and data streaming applications. In our current SmartCell design, static routing is adopted for the data path scheduling of the hierarchical on-chip network. As the system size increases and more complex applications are to be mapped, the static configuration can no longer satisfy the communication requirements or is too complicated to schedule. Dynamic routing becomes inevitable. In this section, we discuss a fully adaptive routing protocol for a store-and-forward computing network [42] that could be adopted in our future design. More details can be found in [63].

Adaptive First Routing

Turn model [41] is a well known dynamic routing scheme that offers partial adaptivity with simple circuitry controls. The basic idea of this scheme is to prohibit the minimum number of turns that break all of the cycles in the channel dependency graph. Thus both



(a)



(b)

Figure 3.8: Reconfigurable hierarchical CMesh network: (a) Modified CMesh Architecture; (b) Switch fabrics of the CMesh network

deadlock and live lock can be avoided. The allowable turns are shown in Fig. 3.9 for the west first (WF) turn model. Two turns to the west direction are prohibited. Unfortunately the degree of adaptivity provided by turn models is highly uneven. For instance, in the WF turn model, if the destination is located towards the west of the source, only one path is available between source and destination, while full adaptivity is achieved if the destination is at east of source node. In 2000, Chiu proposed the odd-even routing scheme [33], which extends the WF turn model to provide more balanced adaptivity at a cost of more complex control rules.

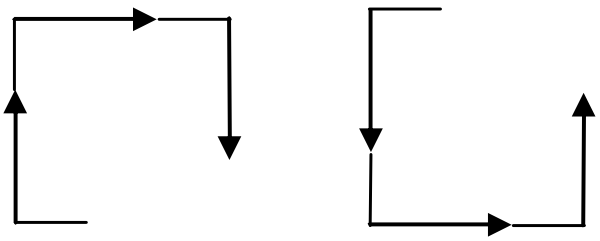


Figure 3.9: West first turn model for dynamic routing [41]

Motivated by these turn model schemes, an Adaptive First (AF) routing is proposed to achieve full adaptivity for all source-destination pairs. Similar to WF routing, three other routing schemes, named east first (EF), north first (NF) and south first (NF), are developed and can be assigned to different packets, based on source-destination location. A specific routing scheme is chosen for a packet at the source node based on the location of the destination. Let S(S_x, S_y) and D(D_x, D_y) be the location of the source and destination nodes respectively. Also we denote $\Delta x = |S_x - D_x|$ and $\Delta y = |S_y - D_y|$. Routing scheme from S to D is chosen according to Algorithm 1.

The principle idea for the AF routing is to independently choose the routing scheme with least restrictions for each packet based on source/destination locations to achieve maximal adaptivity. Fig. 3.10 shows two routing path examples generated by AF routing.

Algorithm 1 Adaptive First Routing

```
1: if Δx ≤ Delta_y then
2:
3:    if S_y ≤ D_y then
4:       Choose West First routing
5:    else
6:       Choose East First routing
7:    end if
8: else
9:
10:   if S_x ≤ D_x then
11:      Choose North First routing
12:   else
13:      Choose South First routing
14:   end if
15: end if
```

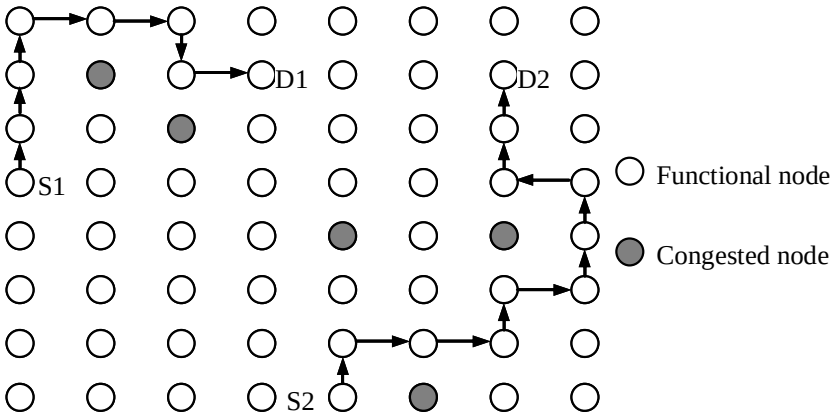


Figure 3.10: Example of Adaptive First routing

Analysis of Livelock and deadlock in AF routing

Livelock and deadlock are the two major problems that need to be addressed in adaptive routing. Livelock occurs when a packet is forwarded endlessly in the network without arriving at its destination. With AF algorithm, a turn model based routing scheme is assigned to each packet. Two turns are prohibited for each scheme, which prevents forming a closed circle path for each packet. Thus it is impossible for a packet to be routed endlessly in a finite dimension mesh network, which guarantees no livelock exists.

On the other hand, deadlock is caused by cyclic waiting for the resources in different nodes. Thus no packet is able to be forwarded anymore. Because a combination of multiple routing schemes are used in AF routing to achieve maximal adaptivity, deadlock-free is not inherited from the turn models. A buffer dependency graph (BDG) is usually used to analyze the deadlock conditions. Two types of deadlocks are observed in a 2D mesh network: one is called acyclic deadlock where multiple nodes are waiting for each other in an acyclic manner; the other is called cyclic deadlock where deadlock is caused by multiple nodes waiting in a circle. Then the deadlock necessary conditions is summarized as: (i) there is at least one node that has only 1 or 2 directions to forward all packets in it; (ii) if 2 directions are available, they are in 90 degree phase angle as shown in Fig. 3.11(b), which is called connected directions. In our design, a hierarchy routing structure is proposed to facilitate deadlock freedom, which can break the deadlock necessary conditions.

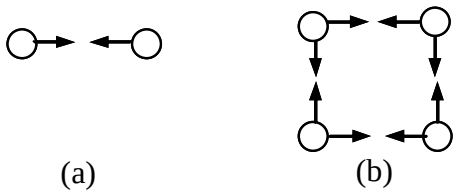


Figure 3.11: Buffer dependency graph for two types of deadlocks: (a) acyclic deadlock; (b) cyclic deadlock.

The hierarchy routing structure for each node is shown in Fig. 3.12. The packets following the AF routing algorithm are stored in the central buffers. When an output channel is available, a packet is chosen by the output controller and is directly transferred to next node's central buffer. We assume a new packet can be only generated in a free central buffer. The network defined by the central buffers is considered as the primary network, which is fully adaptive. Besides the central buffer queue, up to four edge buffers are introduced in every node, with one for each output channel. When the central buffer queue of a node is filled up and the packets satisfy the deadlock necessary conditions, a closest packet to its destination is put into the edge buffer to perform the DOR, such as XY routing [48]. Once a packet enters the edge buffer, it is required to be routed

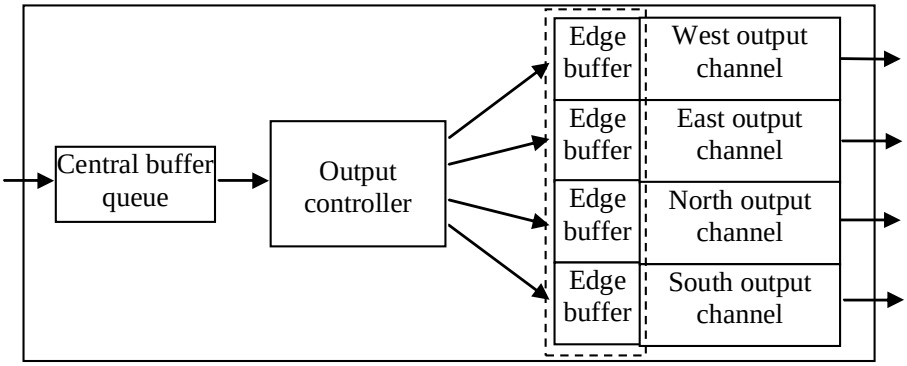


Figure 3.12: Hierarchical routing structure to avoid deadlock in AF routing.

within the edge buffers by DOR scheme. The network defined by the edge buffers is called the secondary network. It is well known that the DOR algorithm is deadlock free by prohibiting a turn from the one dimension to the other dimension. By introducing the hierarchy routing structure, the deadlock necessary conditions are always broken in the primary network and the secondary network is guaranteed deadlock free. Also the packets are routed independently to their destinations in the primary and secondary networks. Thus the deadlock freedom is achieved in this hierarchy routing structure.

Simulation results of AF routing

Although the Adaptive First routing is not included in our current SmartCell design, it has been intensively simulated in software to evaluate its performance. A 10 by 10 mesh network with uniform data traffic pattern is modelled in MatLab with different routing schemes, including Adaptive First routing, West First routing and XY routing. The package delivery rate and network coverage are compared among these routing schemes with randomly selected congested nodes at the beginning of each packet delivery. We assume the congested nodes are not able to participate into any package transfer for the entire delivery process.

Fig. 3.13 shows the packet delivery rate among evaluated routing schemes. The results indicates that the AF algorithm always outperforms the WF and XY routing

algorithms with the presence of congested nodes in the network. At high congestion ratio, the successful delivery rate in AF routing is more than 20% and 30% of those in the WF and XY routing, respectively. This is because the AF algorithm provides full routing adaptivity from source to destination, which maximizes the probability for a packet to be successfully delivered.

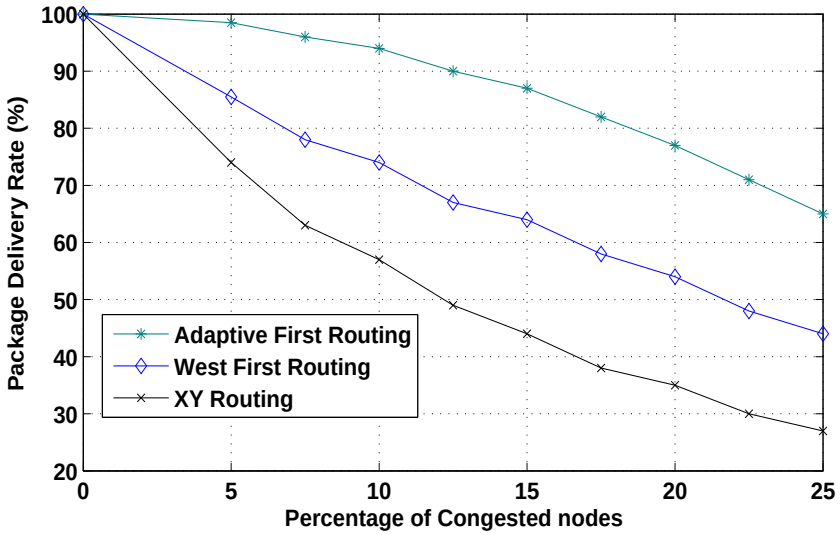


Figure 3.13: Package delivery rate versus different percentages of congested nodes

The network coverage range is used to evaluate the average accessible range in a network as calculated in Eq. 3.1.

$$Coverage = \frac{\sum_{delivered} (|S_x - D_x| + |S_y - D_y|)}{N} \tag{3.1}$$

where the numerator is the summation of the one-norm distance from sources to destinations for all delivered packets. N is the number of delivered packets. The simulation results is shown in Fig. 3.14. For AF algorithm, the degradation of network coverage is within 1 hop range even at high network congestion rate. Comparing to the much lower coverage range achieved in WF and XY routing, this is mainly benefited from full adaptivity achieved by AF routing.

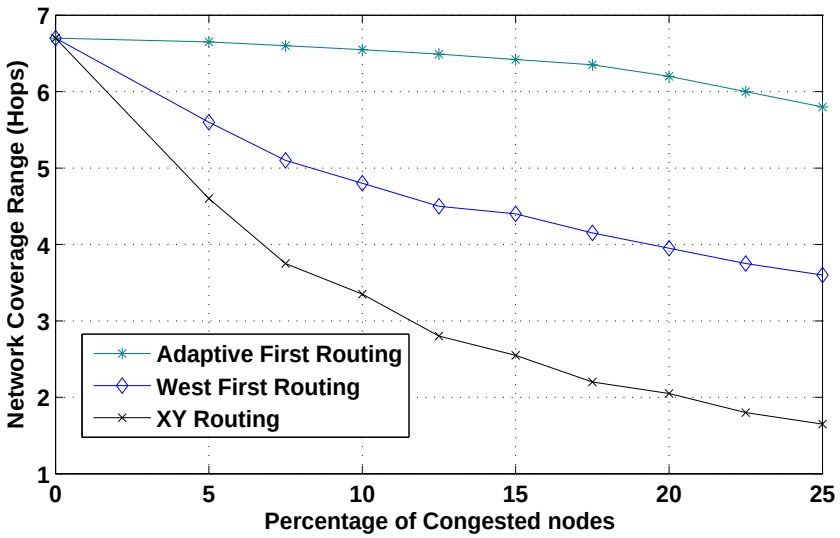


Figure 3.14: Network coverage range versus different percentages of congested nodes

3.6 Related Work

Fig. 3.15 depicts one possible computing system taxonomy, based on homogeneity and granularity. Since SmartCell falls into the category of coarse-grained reconfigurable architecture, it is important and meaningful to compare it with other CGRAs. The related work is discussed and an architectural level comparison is given to distinguish SmartCell from others.

3.6.1 Existing CGRA Designs

Researches has been focused on exploring of efficient CGRA designs as summarized in [44]. In this section, some modern CGRA systems will be briefly discussed. These architectures, as shown in Fig. 3.16, organize a large number of coarse-grained computing units and configurable interconnection fabrics into a 2D mesh or 1D linear array structure. They share the same features of configurability, coarse granularity, scalability, targeting for computing intensive applications, and etc. But they also differ in many fundamental basics. In this section, we also differentiate SmartCell with other CGRAs from the architecture point of view.

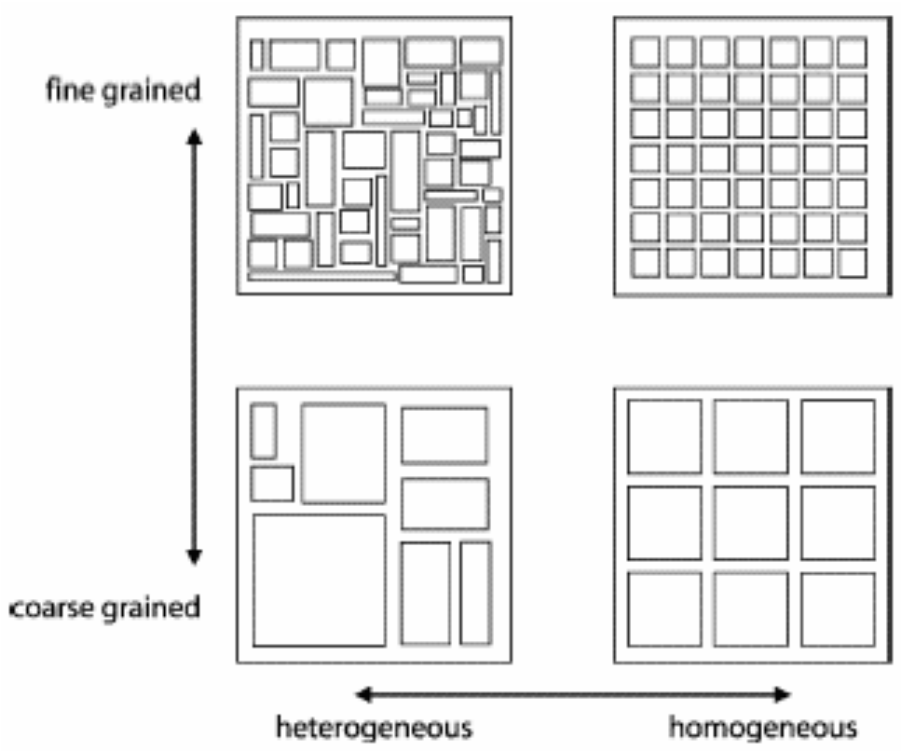


Figure 3.15: Category of different computing systems based on the degree of homogeneity and granularity [27].

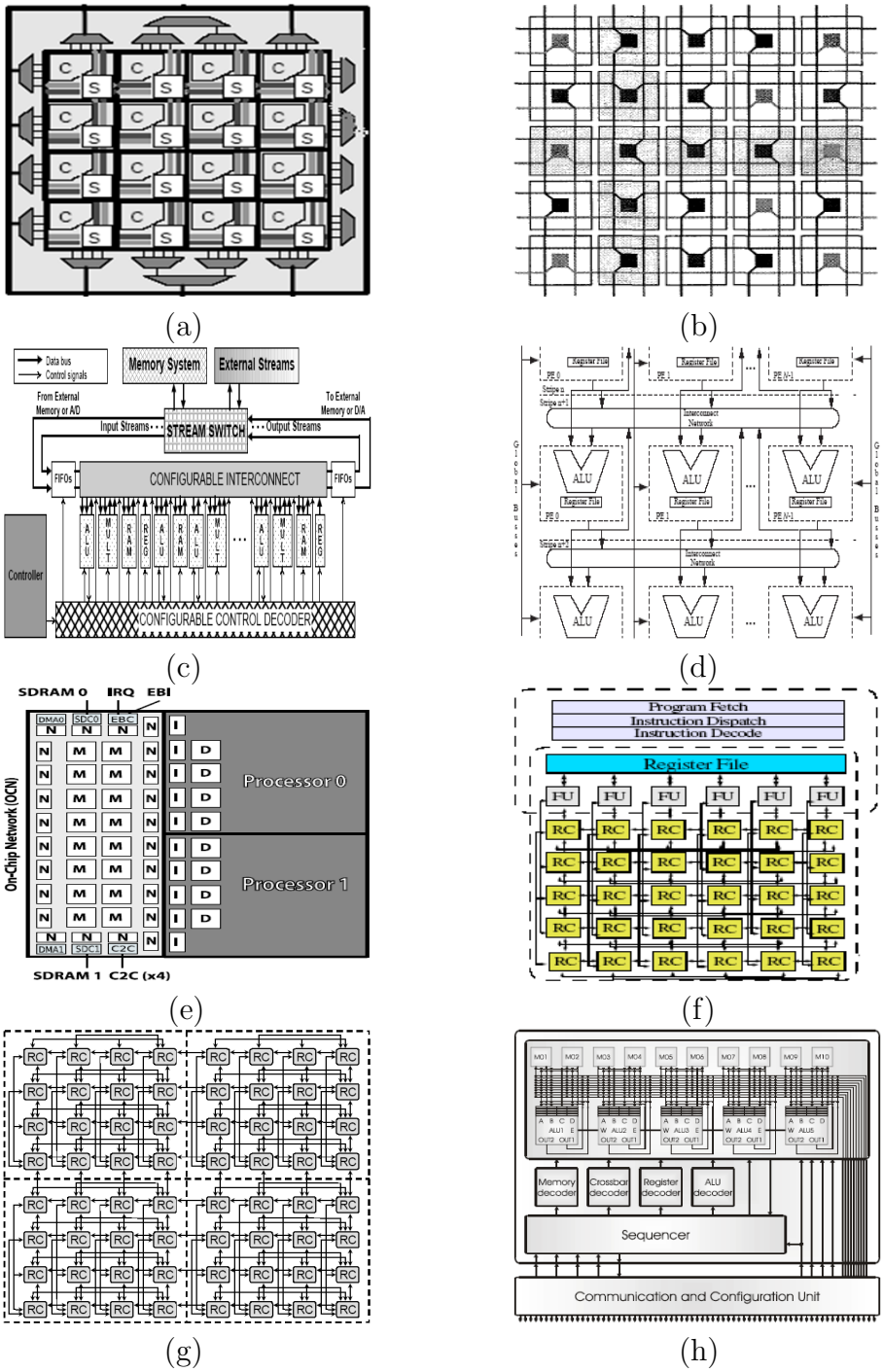


Figure 3.16: Diagram of related CGRA systems. (a) RAW 2D mesh structure [83]; (b) MATRIX 2D mesh structure [69]; (c) RaPiD 1D array structure [40]; (d) PipeRench 1D stripe architecture [79]; (e) TRIPS 2D mesh structure [76]; (f) ADRES 2D mesh Structure [28]; (g) MorphoSys 2D mesh structure [80]; (h) Montium 1D array structure [81].

RAW [83] was designed at MIT in the late 90's. It incorporates 16 simplified 32-bit MIPS processors in a 2D mesh structure to provide high parallel computing capacities. Besides the data parallel applications, it can also be configured to perform irregular or general purpose tasks. The static and dynamic mesh networks are designed to exchange data among processors. Due to distributed instruction memories, the RAW system can be organized in a MIMD manner that has the ability to perform multiple processing streams simultaneously. However, the 6-stage ALU components exhibit a processor execution style involving instruction fetching, decoding, and execution process. A 4 by 4 RAW system was implemented and evaluated in IBM 180 nm CMOS technology. In [84], it is reported that the RAW prototype occupies 331 mm^2 with an average core power consumption of 18.2W at 425 MHz. However, RAW is not able to provide dynamic reconfiguration. This makes it inflexible to adapt to any changes on the fly.

RaPiD [40] was developed at University of Washington in the early 2000's. It links hundreds of heterogenous components in a 1D linear array, including ALUs, multipliers, and RAMs. The potential applications for RaPiD are those of a linear systolic nature or applications that can be easily pipelined among the computational units. Functional specific components can also be involved for application specific designs. In RaPiD, several parallel segmented buses are designed that can be dynamically changed for different data communications. A program sequencer is involved to control the processing units in VLIW fashion. Unlike the RAW processor, the RaPiD uses a combination of static and dynamic control logics, but the dynamic control is very expensive in timing and chip area. It is studied in [80] that the linear array nature exemplifies provision of datapath parallelism in the temporal domain, and also makes it not very appropriate for block based applications, such as 2D systolic array applications.

PipeRench [79] was developed at CMU in the early 2000's. In the PipeRench system, several reconfigurable pipeline stripes are offered as an accelerator for data streaming applications. Each stripe consists of multiple computing units organized in a 1D array structure. Limited configurable interconnection fabrics are developed, including a lo-

cal network inside a stripe, unidirectional nearest neighbor connection between stripes and some global buses. The motivation of PipeRench is to dynamically reconfigure the data path of the targeted application for efficient calculations. A pipeline reconfiguration scheme [78] was developed to achieve the hardware virtualization that relies on fast partial dynamic configuration and run time scheduling of configuration contexts and data paths. By this means, an application that is not physically fit on the hardware can still be executed on PipeRench. This dynamic configuration is built on a cycle by cycle basis that allows the configuration of a pipeline stage in every clock cycle. However, similar to the RaPiD system, the linear array structure makes PipeRench inefficient for the block based applications (for example, 2-D signal processing tasks). Additionally, with limited interconnection between stripes, the PipeRench system does not support computation of feedback loops, and it may waste available parallelism when squeezing wide graphs into a linear sequence of stripes [31]. A prototype PipeRench was fabricated in 180 nm CMOS technology with about 55 mm^2 die size. In FIR filter test benches, the chip consumes 519 mW without virtualization and 675 mW with virtualization operating at 33 MHz.

The MIT MATRIX [69] incorporates a large number of Basic Functional Units (BFUs) in a 2D mesh structure. An 8-bit ALU module is designed inside the BFU to generate coarse-grained computing resources. The routing fabrics provide 3 levels of 8-bit bus connections among the BFUs, and may be configured to perform in SIMD, MIMD, or VLIW fashion. A novel concept of dynamic instruction generation as data signals is introduced by MATRIX for dynamic system reconfiguration. This allows the MATRIX system to source some configuration context statically, while maintaining others dynamically over the routing structure within the computing components. However, the 3-level hierarchical data switching network involves variable interconnection delays, which could become a limiting factor for the scheduling of stream processing among multiple BFUs. Furthermore, the 8-bit granularity may not be sufficient for today's computational applications. Multiple BFUs are needed to support wider operations, which in turn increases the computing and communication overhead and results in relatively complex control logic.

TRIPS [76] was developed at UT Austin in early 2000's. Similar to the RAW system, TRIPS adopts large processing cores targeting to provid high performance across a wide range of applications. Based on different application features, three level parallelism of ILP, TLP and DLP can be configured in TRIPS to achieve high performance. A prototype TRIPS system was fabricated in 130 nm technology in 2007 [77]. The chip contains two cores and a separate on-chip network and occupies 336 mm^2. It can operate at 366 MHz and consumes about 36 W power. This high power consumption is mostly related to the non-negligible overhead introduced by the large core and complex configuration scheme designed to support general purpose applications.

MorphoSys [80] was developed at UC Irvine in early 2000's. It is an integrated and configurable system-on-chip, targeted for high throughput and parallel data applications. It incorporates a reconfigurable processing array, a modified RISC processor, and an efficient memory interface unit on the same chip. The main component of MorphoSys is an 8 by 8 reconfigurable array (RA). The RA is clustered into four 4 by 4 mesh structures with bus connections of variable widths. Two sets of Frame Buffer (FB) are provided to minimize data I/O overhead by overlapping data load and store with computations. The RISC processor with extended instruction set is adopted as the system controller that is responsible for the RA configurations and dynamic context memory updating. The configuration words are broadcast to the RA in either column-wise or row-wise mode. This constrains the computing components in the same column (or row) to perform the same operations. Thus the MorphoSys system is designed to be operated in the SIMD style.

Power consumption is another important aspect of reconfigurable architecture designs. More recently, some other CGRA systems have also been developed to provide ultra low power consumption, such as ADRES [28] and Montium [81] with limited computing resources. The ADRES architecture introduces two functional views, which includes a VLIW processor to execute the non-kernel code with ILP and the reconfigurable matrix to accelerate the parallel kernels. It is reported in [29] that a 4 by 4 ADRES can achieve a power efficiency of 0.24mW/MHz. On the other hand, the Montium tile processor links 5

identical ALUs in a linear array style to communicate through 10 local SRAM memories. A hardware sequencer is designed to dynamically reconfigured the ALUs in VLIW mode. Montium achieves a power consumption of 0.58 mW/MHz in FFT [45].

Several other reconfigurable architectures have been implemented with various technologies [24, 38, 57]. Most of them have been focused on the exploring of computational models or efficient design with respect to area and performance. For example, Processor-In-Memory (PIM) based systems [92, 39] integrate the processing logic and memories into the same chip and try to perform the computations directly in memory, which greatly reduces data transfer overhead between CPU and main memory.

3.6.2 Architectural Comparison With other CGRAs

A quantitative comparison of the SmartCell system with other evaluated systems is difficult due to the differences in implementation technology, application details, system setup, and so forth. Furthermore, the performance details for the implemented benchmarks are not disclosed in the literature for most designs, which makes it impossible to compared with.

Table 3.3 summarized the targeted applications and objectives of a few CGRA systems, along with their key features. RAW and TRIPS are aimed to develop a universal system as flexible and configurable as possible to fit a wide range of applications, which involves complex connection and large core components. On the other hand, the other systems are targeted at domain specific applications, such as signal processing and multimedia applications. They take advantage of more specific architectures and datapath controls to reduce overhead and achieve better efficiency in the targeted application domains.

Table 3.4 lists the homogeneity, processing style and interconnection networks among the compared reconfigurable systems. Most designs involves homogeneous resources. RaPiD adopts the heterogenous style which links the processor, memories and specific function units in a linear array to make it efficient for certain applications, but it results

	Targets/Objectives	Key features
RAW[83]	General purpose application	complex static/dynamic route
RaPiD[40]	pipeline application	reconfigurable pipelined datapath
PipeRench[79]	DSP application	dynamically configurable datapath
MATRIX[69]	systolic array processing	layered connection
TRIPS[76]	general purpose application	complex configurable cores
MorphoSys[80]	DSP application	SoC structure
Imagine[54]	multimedia application	stream architecture
Montium[45]	multimedia application	1D array & low power consumption
SmartCell	Data streaming applications	hierarchical connection

Table 3.3: Comparison of targeted application and key features among selected CGRAs.

	Homogeneity	Processing style	Interconnection
RAW[83]	homogeneous	MIMD	static/dynamic 2-D mesh
RaPiD[40]	**heterogeneous**	VLIW	1-D pipelined datapath
PipeRench[79]	homogeneous	SIMD	1-D linear array
MATRIX[69]	homogeneous	MIMD	hierarchical buses
TRIPS[76]	**heterogeneous**	EDGE	dynamic routed network
MorphoSys[80]	homogeneous	SIMD	row/column connection
Imagine[54]	homogeneous	VLIW	global/local switches
Montium[45]	homogeneous	MIMD	communication unit
SmartCell	homogeneous	MIMD	3-level hierarchical routing

Table 3.4: Comparison of homogeneity and interconnection scheme among selected CGRAs.

in an irregular layout and poor scalability.

Processing style can be generally categorized into three types - SIMD, MIMD and VLIW. SIMD uses data level parallelism (DLP) to accelerate computing process. Besides DLP, task level parallelism (TLP) is also available in MIMD systems. VLIW uses long instruction code to control multiple on-chip processors. In TRIPS [76], the EDGE structure uses instruction blocks for system configuration. The interconnection design also varies greatly, ranging from bus connections, 1D array structure, row/column broadcasting to 2D mesh structure.

Finally, the system prototyping results are listed in Table 3.5, including process technology, maximum frequency, chip area and reported power consumption. A direct com-

	Process (nm)	Max. freq. (MHz)	Area (mm^2)	Power (mW/MHz)
RAW[83]	180	425	331	428
RaPiD[40]	500	100	5.07	N/A
PipeRench[79]	180	120	55.5	20.5
MATRIX[69]	500	100	1.8/element	N/A
TRIPS[76]	130	366	336	98
Montium[45]	130	100	1.83	0.5
SmartCell	130	123	8.2	1.6
SmartCell-II	90	295	5.0	3.1

Table 3.5: Comparison of chip implementation results among selected CGRAs.

parison of these metrics may not be very meaningful due to different on-chip resources, implementation process, evaluation benchmark, and system setup. Furthermore, performance details are not disclosed in most designs. For this reason, we will take the system throughput and energy efficiency as the evaluation metrics in Chapter 4 when comparing SmartCell with other systems, including FPGA, ASIC, RaPiD and Montium.

SmartCell integrates some prominent features in previous systems. The 16-bit granularity of the basic operations is efficient for the data parallelism exploration, while keeping the cost of computing and communications low. It can be configured to operate in SIMD, MIMD and systolic array styles due to the distributed contexts and hierarchical on-chip connections with uniform delays to ease the scheduling of the stream processing among multiple cell units. In combination of these features, we say that the SmartCell system is a unique approach in the CGRA family.

3.7 Summary

In this chapter, we presented the SmartCell architecture as a novel CGRA system targeting stream processing domain. The architecture design, including cell structure, processing element, control logics, on-chip interconnection and dynamic configurations, were presented in details. A dynamic routing algorithm was also proposed to provide a fully adaptive routing protocol for future system expansion. At last, several related CGRA systems

were discussed and reviewed. These CGRAs were also compared with SmartCell from the architecture point of view.

Chapter 4

SmartCell Experimental Results and Evaluations

In this chapter, a prototype SmartCell with 64 processing elements was designed in CMOS standard cell ASICs with TSMC 0.13 μm technology. The design and verification methodology used in our research is briefly discussed. We then present the mapping structure of a set of benchmark applications onto the SmartCell system. The power/energy consumption and system throughput results are then compared with other computing platforms, including FPGA and standard cell ASICs. Finally, we compare the energy efficiency and system throughput performance with other CGRA systems, including RaPiD [35] and Montium [45]. The reason to choose RaPiD is that it shares similar hardware resources with our design and the interested performance for a common set of benchmarks are disclosed in details. Montium is among several recently developed ultra low power CGRA systems. The system throughput and energy consumption is compared among these CGRAs.

4.1 Design and Evaluation Methodology

Typically, standard cell ASIC design methodology is best suited for a small design team and short time-to-market cycle in contrast to full-custom design methodology that could lead to optimized timing and performance. It is studied in [36, 32] that full custom design could result in more than 3 times improvement in both area and timing performance compared with standard cell design. But due to the small design team and proof-of-concept purpose, we are restrict to follow the CMOS standard cell ASIC design flow.

The design flow and evaluation methodology is depicted in Fig. 4.1. After the architectural design is finalized, a functional register transfer level (RTL) model is written in hardware description language (VHDL or Verilog) and the behavior simulation is performed in MentorGraphics ModelSim for functionality verification. The RTL model is then translated and mapped into the standard cell library using Synopsys logic synthesis tool DesignCompiler. The timing performance is analyzed by Synopsys PrimeTime. Any timing violations are fixed by incremental resynthesizing with new timing constraints or by restructuring the logic design. After synthesis, a netlist file is generated and simulated at the gate level with proper wire load modes. ModelSim monitors the signal switching activity and creates a value-change-dump (vcd) file that can be used by Synopsys PrimePower for power consumption evaluations of the major component on the chip.

4.2 Verification Methodology

Functional verification methodology is drawn in Fig. 4.2. For the evaluated benchmarks, the input signals are generated in MatLab based on application specs. These inputs are then quantized and scaled to provide hardware stimulus signals. The chip level simulations are then performed using the input file and manually generated test bench configurations. The same application is emulated in MatLab to mimic data flow paths as in hardware simulations. The end-to-end hardware/software outputs are compared within certain tol-

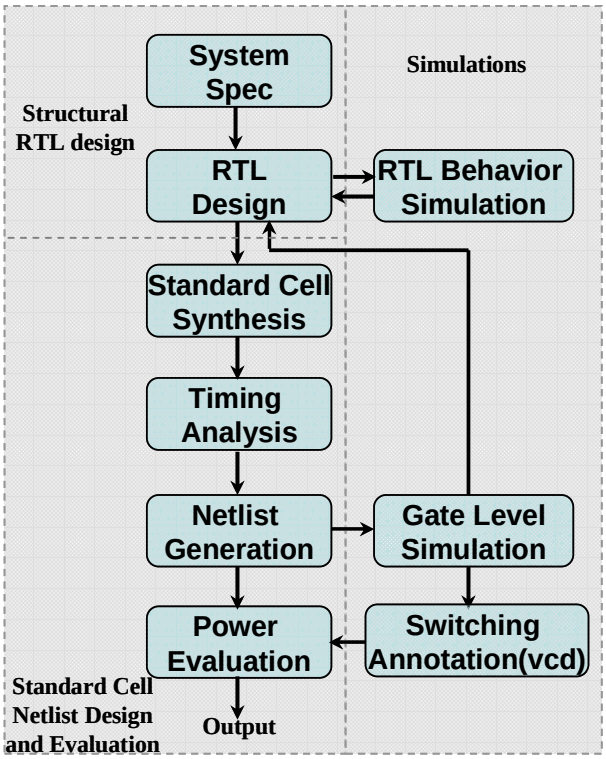


Figure 4.1: Physical design flow and evaluation methodology involved in SmartCell research.

erance range. When mismatch happens, we trace along the data path back to the first module with unnegligible mismatches between hardware and software simulations. The subblock input can then be extracted from software emulation to provide isolated block level stimulus for hardware debugging. Furthermore, both RTL behavior and netlist gate level simulations are performed and compared against each other to make sure no synthesis errors or timing violations are introduced during synthesis.

4.3 Application Mapping

The eight benchmark applications listed in Table 4.1 have been manually mapped onto the SmartCell system. These benchmarks represent a wide range of real-time applications from signal/image processing to scientific computing. The application mapping structure is described in this section.

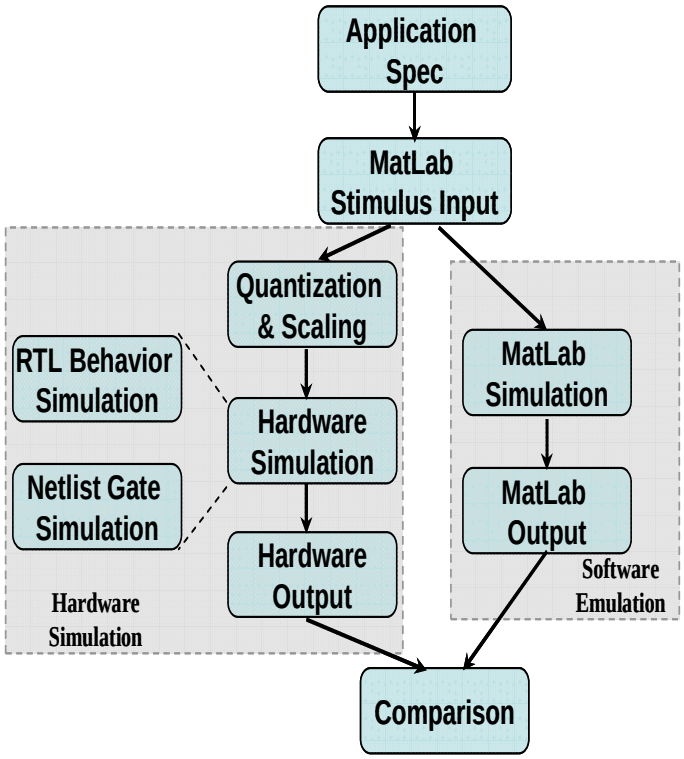


Figure 4.2: SmartCell prototype verification methodology.

Application domain	**Test Benches**
Signal processing	1. 64-tap FIR 2. 32-tap IIR 3. 64-point FFT
Multimedia and image processing	4. 8 by 8 2D-DCT 5. 8 by 8 Motion Estimation(ME) in 24 by 24 area
Scientific computing	6. 128 by 128 MMM 7. 64^{th}order Polynomial Evaluation(PoE)
Data encryption	8. RC5

Table 4.1: Application domain and test benches mapped onto the SmartCell prototype system.

4.3.1 Finite Impulse Response (FIR) Filter

Linear-phase digital filters are frequently used in communication, image processing, and speech processing algorithms. The output of a general N-tap FIR-filter is calculated as:

$$y[n] = \sum_{i=0}^{N-1} (x[n-i] \times h[i]) \tag{4.1}$$

where x and h are the input signals and the filter coefficients, respectively. The systolic array structure is considered the most efficient solution for parallel FIR-filter designs. The structure of a 4-tap FIR-filter is shown in Fig. 4.3, which is able to be mapped onto four PEs in the same cell unit. The input data is propagated through the cascaded data buffers between PEs. The multiply-accumulate (MAC) operation is executed in every PE. Higher order FIR-filters can be implemented by chaining multiple cell units in a linear array fashion. A fully pipelined FIR-filter implementation can be mapped onto the SmartCell architecture by loading a unified MAC configuration code to the instruction memories of all active PEs. The filter coefficients can be preloaded into an on-chip memory and can be dynamically changed upon request. The prototyping of the pipelined FIR-filter leads to a system throughput of one output per clock cycle after initial setup.

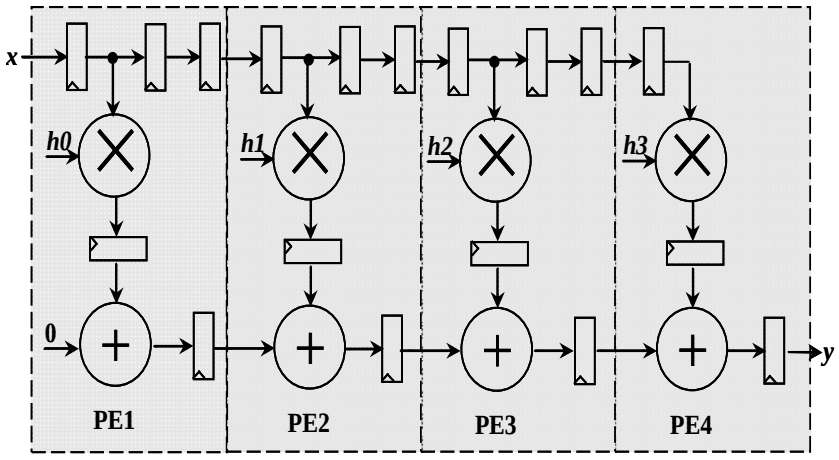


Figure 4.3: Mapping of a 4-tap FIR-filter onto 4 PEs in systolic array structure.

The hardware and software simulation results for a 64-tap FIR-filter are compared in

Fig. 4.4, with the output index number in x axis and magnitude in y axis. The input signals, scaled by a factor of 2^{12}, and the filter coefficients are generated in MatLab and are then read into the hardware test bench in ModelSim. Software simulation results are computed by MatLab built-in functions. The differences between the fixed point hardware results and the floating point software results are all within 10^{-3} range, which proves the correct functionality of the SmartCell prototype.

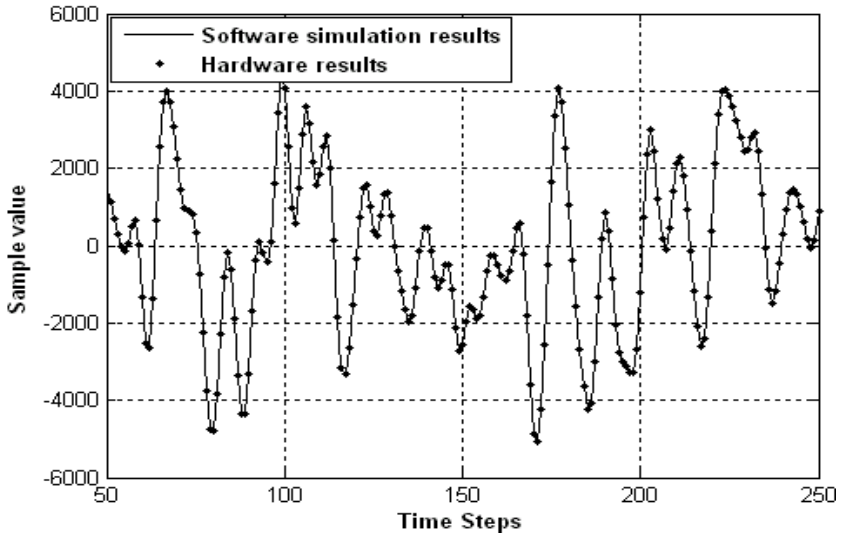


Figure 4.4: Comparison of hardware and MatLab software simulation results of FIR filter design.

4.3.2 Infinite Impulse Response (IIR) Filter

The IIR-filter is another primary class of digital filters with feedback loops. It can achieve a much steeper frequency response with fewer stages compared with the FIR-filter. The output for an N^{th} order IIR-filter is calculated as:

$$y[n] = \sum_{i=1}^{N}(a_i y[n-i]) + \sum_{i=0}^{M}(b_i x[n-i]) \tag{4.2}$$

where a and b are the coefficients for the output feedback signal y and the input signal x, respectively. In general, the feedback order N is no less than the input order M. Due to

its internal feedback feature, the IIR-filter is much more susceptible to the intermediate computing and round off errors than the FIR-filter, especially in high order filter designs. It is well known that the high order IIR system can be factored into a cascade of multiple second-order subsystems, called Biquad sections. Such that its system response function can be rewritten as:

$$H(z) = \prod_{i=1}^{k} H_i(z), \text{ where } H_i(z) = \frac{1 + b_i 2z^{-1} + b_i 3z^{-2}}{1 + a_i 2z^{-1} + a_i 3z^{-2}} \tag{4.3}$$

The hardware structure for the basic Biquad section core can be implemented in one cell unit as illustrated in Fig. 4.5. Similar to the FIR-filter design, every PE unit primarily performs the MAC operation. Due to the fixed point mapping onto the SmartCell system, a shift-right scaling operator is necessary for PE3 and PE4 to scale down the products of the feedback output and the feedback coefficient. By cascading multiple Biquad sections along the adjacent cell units, higher order of IIR-filters can be designed efficiently. The filter coefficients are loaded directly into the hardware registers and can be updated at run time. The shift-right operation is performed in PE3 and PE4 due to the scaling factor involved in the fixed point implementation, as shown in Fig. 4.5.

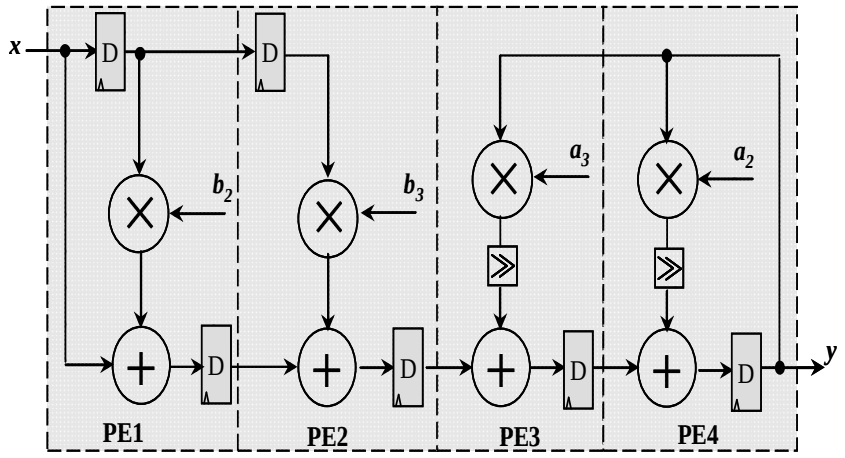


Figure 4.5: Mapping of Biquad IIR-filter onto 4 PEs for cascaded IIR design.

The hardware and software simulation results for a 32-tap IIR-filter are compared in Fig. 4.6. The random signals (with a scaling factor 2^{12}) and the filter coefficients are

generated in MatLab and are then fed into ModelSim for hardware simulations. The simulation shows that the hardware results matches well with the software simulation (within 10^{-2} difference range), which validates the IIR-filter design on the SmartCell system.

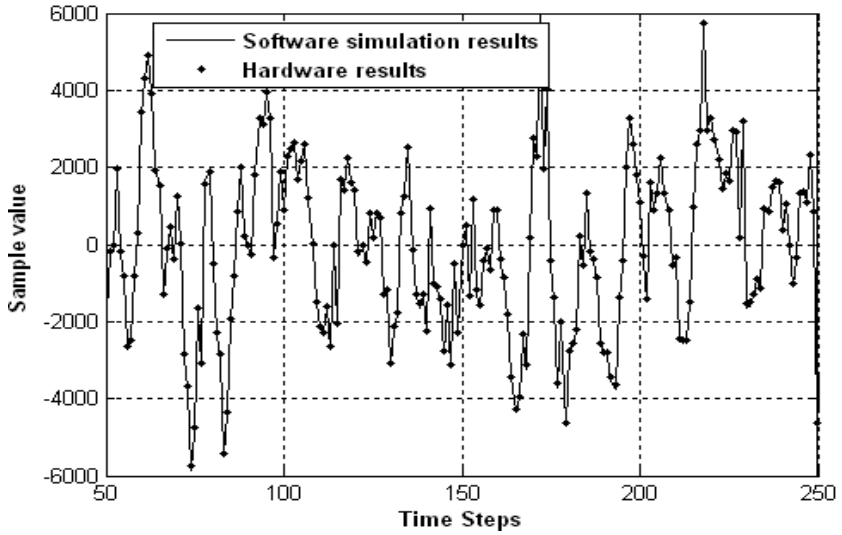


Figure 4.6: Comparison of hardware and MatLab software simulation results of IIR filter design.

4.3.3 2D Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is an important mathematic algorithm that has been used in many real-time applications from audio filters to video compression hardware. 2D DCT is one of the core functions for JPEG image compressor. The calculation of an N by N 2D DCT can be calculated as:

$$X_{i,j} = a_i b_j \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} x_{k,l} cos[\frac{(k+1/2)i\pi}{N}] cos[\frac{cos(k+1/2)j\pi}{N}], \text{ where } 0 \leq i,j < N \quad (4.4)$$

where $x_{k,l}$ is the matrix element located at i^{th} row and j^{th} column. a and b are the normalization and scale factors.

Due to the high degree of computational complexity involved in the 2D DCT, directly

mapping of the 2D computation is not efficient for hardware implementations. Benefitting from the separability property, the calculation can be greatly simplified by separating the 2D DCT into two 1D DCTs that involves only convolution operations, as shown in Fig. 4.7(b). The first 1D DCT is performed row-wise on the input matrix and the second one is performed column-wise on the outputs of the first 1D DCT. This decomposition scheme reduces the complexity of the calculation by a factor of four [26]. Fig. 4.7(a) shows the hardware mapping and data flow of a 4 by 4 DCT system on the SmartCell architecture.

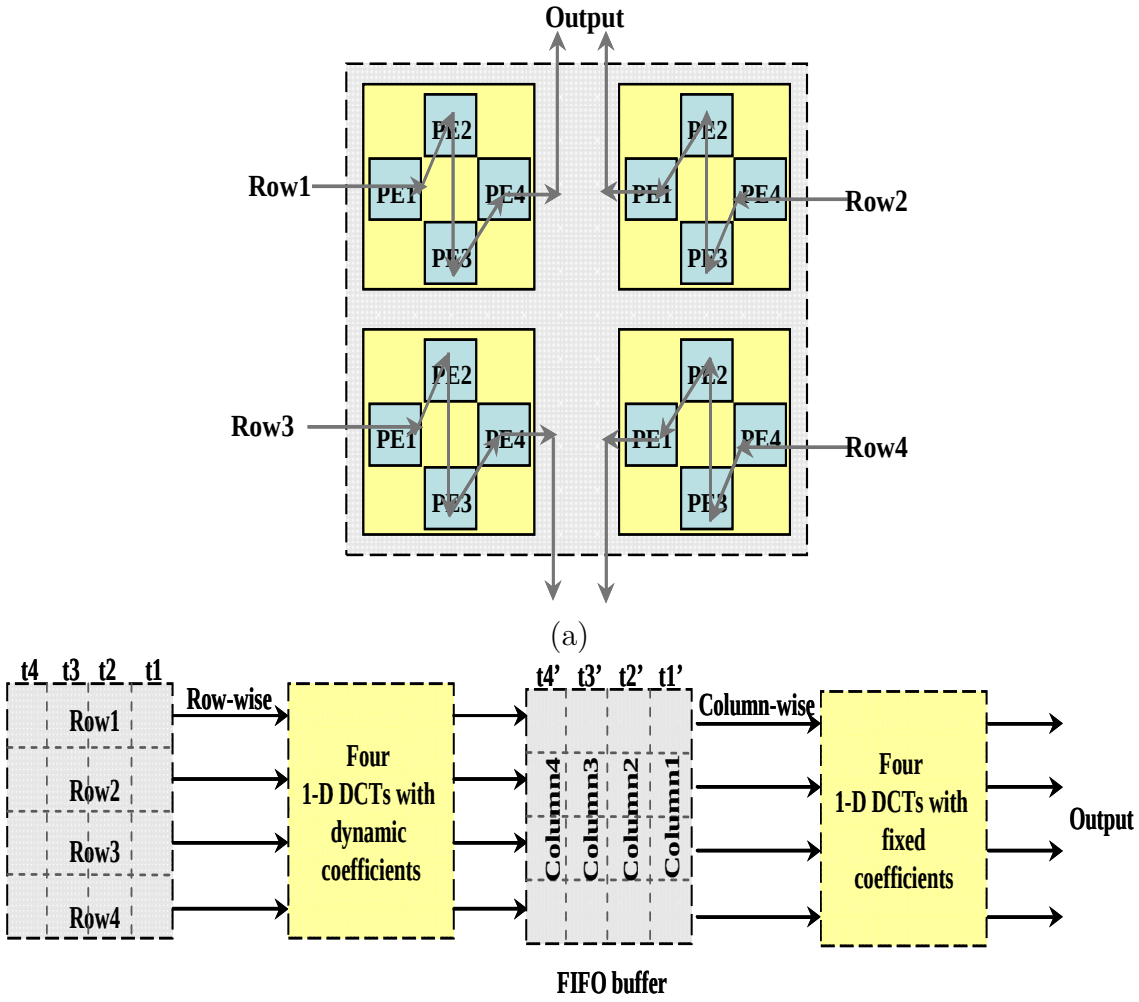


Figure 4.7: 2D DCT mapping and re-scheduling scheme: (a) Hardware mapping structure. (b) Fully pipelined implementation of the 2D DCT through two 1D DCTs with input retiming scheme.

Because the same hardware resources are used to calculate both DCT operations, the design of a proper scheduling for the input signals are essential to exploit the computing pipelines and improve system throughput. In our design, we column-wisely input the initial matrix into the first DCT. In this case, the first column of the matrix is input at $t1$ and second column is input at $t2$, as shown in Fig. 4.7(b). By this means, it is guaranteed that the corresponding DCT values for the row elements of the original matrix will be output at the same time spets, as denoted at time $t1'$, $t2'$, $t3'$ and $t4'$. These vectors can be immediately input to the second stage DCT without waiting for the completion of the first one. Thus a fully pipelined structure can be achieved between the first and second DCTs. A system throughput increase of about 22% is achieved from this input retiming scheme. A 16-bit 8 by 8 2D DCT has been mapped onto the SmartCell system. The simulation results show that 64 clock cycles are required to complete the transform of one image block in pipelined execution.

4.3.4 RC5 Data Encryption

Nowadays, the computing capacity has become a limit factor for many data encryption designs. The coding/decoding process usually demands a huge computing bandwidth to achieve the security and performance requirements. As a case study, we mapped the RC5 [30] algorithm onto the SmartCell system to exploit its capability and performance in the data encryption domain. RC5 is a block cipher notable for its simplicity and flexibility. It only involves basic logic and shift operations and can take a variety of input block sizes.

In our design, four PEs in a cell unit are used to implement half round of RC5 as shown in Fig. 4.8. Each PE performs specific operation as illustrated in the figure. The same calculations are performed in the next cell unit to compute one round RC5 operation. The data dependent rotation inside a single round is the key feature for the RC5 algorithm. The encryption process continues until the completion of the last round. Multiple rounds of the RC5 can be easily cascaded to generate a pipelined structure. Therefore, the blocks

of the plaintexts are able to be continuously fed into this pipe structure to improve the system throughput. The number of involved cells can also be dynamically changed to adapt to different security requirements.

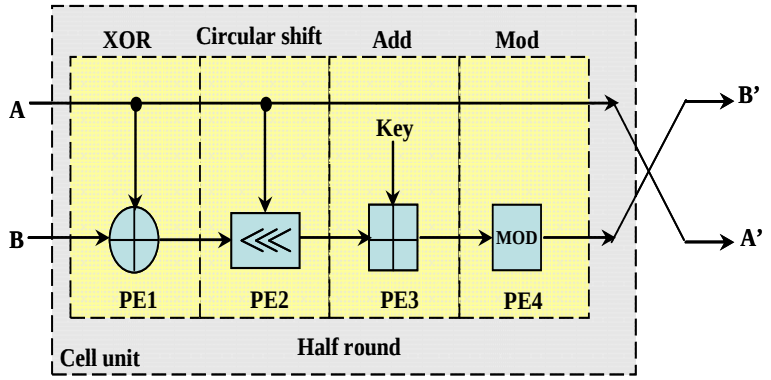


Figure 4.8: Mapping of half round RC5 onto 1 cell unit.

A fully pipelined 8-round RC5 coder/decoder machine has been mapped onto the SmartCell system that occupies 16 cell units. 128 cycles are needed to finish the encryption or decryption process for each block. In the pipelined structure, one output can be generated every 2 clock cycles. The hardware simulations are verified by the MatLab software simulations.

4.3.5 Matrix-Matrix Multiplication (MMM)

Matrix operation algorithms are often used in scientific computing applications. The matrix-matrix multiplication (MMM) is chosen to be mapped on the SmartCell architecture. As of other benchmarks, there are several mapping schemes available. The 2D systolic array is designed to implement the MMM application. For demonstration purpose, a simple 2 by 2 matrix multiplication is developed and mapped onto 4 cell units as illustrated in Fig. 4.9. The notations used in the figure are indicated in the following Eq. 4.5. Five time steps are needed to complete the 2 by 2 MMM operations as shown in Fig.

4.9.

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{4.5}$$

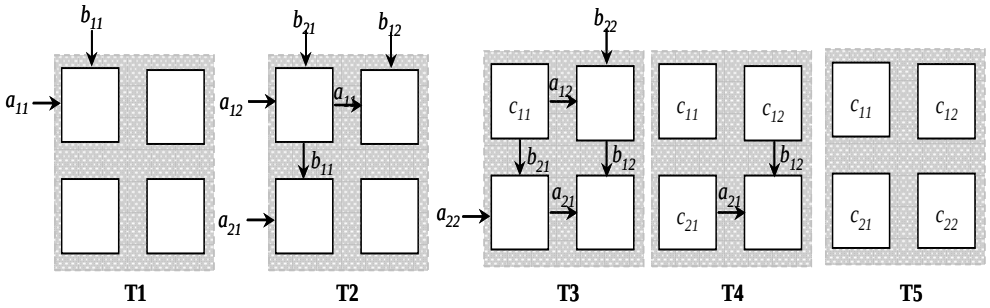


Figure 4.9: Systolic array mapping of 2 by 2 matrix multiplication onto SmartCell

A 16-bit 4 by 4 matrix multiplication is developed and mapped onto the SmartCell architecture. The simulation shows that 24 clock cycles are needed to generate the product result of two matrices. While through full pipeline, only 4 cycles are needed to produce one output result. The hardware results are validated with the software simulations.

In this systolic array structure, the scalability becomes the system bottleneck, since the number of processing units increases quadratically with the matrix dimension. For example, 64 cells are required for the direct mapping of an 8 by 8 MMM applications. To address the scaling issue, a block recursive partitioning algorithm can be used to decompose large matrix multiplication into several smaller ones. Thus the N by N matrix multiplication can be decomposed as:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{4.6}$$

where each element in Eq. 4.6 represents a N/2 by N/2 matrix. By this means, the product of large dimensional matrices can be calculated by the products of several smaller one. In our design, a 4 by 4 2D systolic array is generated with each cell to be a single operation unit. A more efficient mapping scheme is introduced in SmartCell-II when

on-chip data memory is provided.

4.3.6 Polynomial Evaluation (PoE)

Polynomial evaluation is often used as a primitive operation in the scientific computing and data encryption applications. Eq. 4.7 calculates the n^{th} order expression of a polynomial evaluation. By providing a set of coefficients a, the output y can be calculated as the system response with respect to the input signal x. The direct calculation involves a computing complexity of $O(n^2)$ for expensive multiplications, and error may accumulate to degrade the precision of final results. Fortunately, there exists an alternative way to calculate the polynomial evaluation in a reduced computing complexity of $O(n)$, as shown in Eq. 4.8.

$$y = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \tag{4.7}$$

$$y = (((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a0 \tag{4.8}$$

The modified equation is also very efficient for hardware implementations. Fig. 4.10 depicts the mapping of a 4^{th} order polynomial evaluation onto one cell unit. This structure is very similar to the FIR filter design, except that the MAC output from the previous PE is connected to the multiplication operator instead of the addition operator in the next PE. Similarly, one output can be generated every clock cycle after certain initial setup time.

4.3.7 Motion Estimation (ME)

The last but not least bench mark being implemented on the SmartCell system is the motion estimation. The motion estimation plays an important role in video compression algorithms to remove temporal redundancies between successive image frames. It usually requires a high computing capacity. The block-matching algorithms are the most popular methods for the motion estimations because of their simplicity in hardware im-

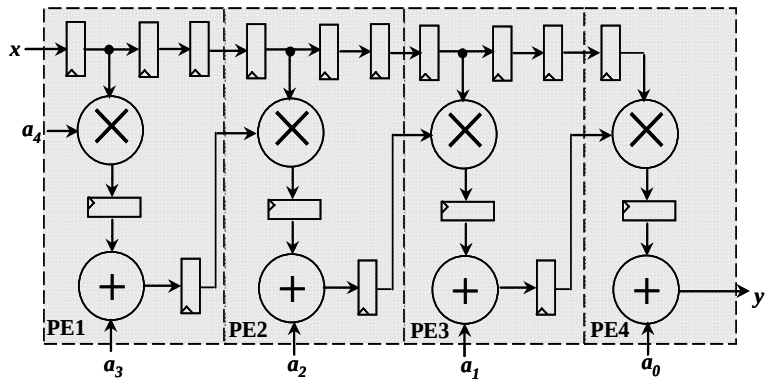


Figure 4.10: Linear array mapping of a 4^{th} order polynomial evaluation onto a cell unit.

plementations. The full search block matching (FSBM) [46] is able to achieve an optimal performance by evaluating all possible displaced candidate blocks in the search area. But it also involves the maximum computing cost. The mean absolute difference (MAD) is usually adopted as the criterion in FSBM that can be calculated as:

$$MAD(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{n} |S(i+u, j+v) - R(i,j)|, \text{ where } -p \leq u, v \leq p \qquad (4.9)$$

R and S are the reference and candidate blocks of size n by n. (u, v) represents the candidate's displacement vector with a maximum displacement value of p. The FSBM algorithm calculates the MAD values of all possible candidate $(2p+1)^2$ blocks and selects the one with the minimum MAD value as the best match in the motion estimation.

The FSBM calculation can be mapped onto the 4 cell units in a 1D array style. The reference and candidate blocks are fed into the 1D array to start the matching process. The partial results are accumulated from cell to cell. Different delays are scheduled for the input data among different cell units, which is scheduled during test bench design. In a pipeline style, a MAD value can be generated every 4 clock cycles. Furthermore, 4 pipelines can be concurrently executed in a 4 by 4 SmartCell system that achieves an average throughput of 1 output per cycle for a 4 by 4 block matching. Based on this mapping scheme, a more realistic system with an 8 by 8 reference block and a 24 by 24

System dimension	4 by 4
Design tools	ModelSim, Synopsys CAD tools
Library	TSMC 0.13 μm process
Synthesis Environment	Worst case condition
Voltage	1 V
Simulation frequency	100 MHz

Table 4.2: System design environment and simulation parameters.

search area is implemented on the SmartCell architecture.

4.4 Design Environment and Comparison Scheme

The prototype SmartCell was developed and synthesized with standard CAD tools. A functional RTL model was first designed in HDL hardware description language and was then synthesized in Synopsys DesignCompiler to generate the CMOS standard cell ASICs using TSMC 0.13 μm technology. No custom optimization was applied during this process. The area and timing results were also generated by DesignCompiler using worst case condition. The synthesized design was then annotated via a set of benchmark simulations for power consumption estimation in Synopsys PrimePower. The energy efficiency was evaluated as the number of operations executed per second per watt (GOPS/W). Some experimental setup is listed in Table 4.2.

For power consumption and system throughput evaluations, all benchmarks are simulated at the same operating frequency of 100 MHz. The same simulation frequency was also used by RaPiD for its power consumption analysis. Because the RaPiD was designed in 0.5 μm process and was operated at 3.3 V, a fair comparison requires scaling down its power consumption by a reasonable factor. In our study, full scaling [51] is performed that scales down the power consumption from 3.3 V to 1 V by a factor of 3.3^2. By this means, the process dimension is also scaled down to 0.15 μm. To compensate the effect of dimension scaling, constant voltage scaling [51] is then performed to scale up the power consumption by a factor of 1.7. Therefore, the RaPiD power consumption is scaled down

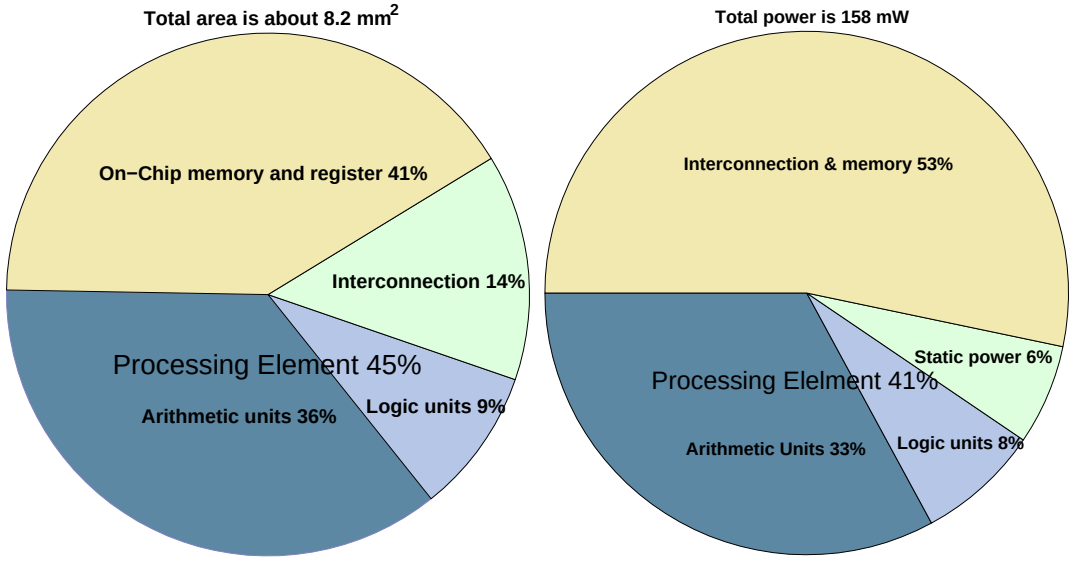
by an overall factor of 9.34.

The same benchmarks are also directly implemented on the FPGA platform to provide performance comparison. The Altera's Stratix II FPGA with 90 nm process technology is selected as the benchmark platform. In particular, an EP2S30 FPGA device is used, since it is the smallest Stratix II FPGA that contains the same number of multipliers as in the SmartCell system. The benchmarks are designed in Altera's Quartus II 6.1 CAD tool and simulated at 100 MHz in ModelSim. The PowerPlay Analyzer is used to evaluate the power consumption based on the switching annotation generated from the gate level simulations. For fair comparison, only the core power consumption is recorded in the FPGA implementation, since the I/O and aux power is not included in others.

The ASIC implement is also generated for every test bench using the same HDL code in the FPGA designs. It is expected to provide the best power/energy performance at a cost of non-flexibility. We use the same 0.13 μm process technology as in the SmartCell. Due to the large set of benchmarks under test, standard cell circuits are generated automatically by the Synopsys CAD tools without custom optimizations. We estimate the power consumption of the ASICs based on the gate level simulations at 100 MHz. Similarly, only the logic core power is recorded.

4.5 Synthesis Results

The area of the SmartCell system according to the synthesis tool is about 8.2 mm^2, which is about 1.6 million gates. The system area, separated into PE, on-chip memory and registers, and interconnection, is shown in Fig. 4.11(a). The area of processing elements are further decomposed into arithmetic and logic units. The interconnection area is calculated by subtracting the area of the processing units and on-chip memories from the total area. The synthesis results indicate that the processing units contribute to about 45% of the total area, with 36% for arithmetic units and 9% for logic units. The on-chip memory and register together comprise about 41% of the area, mainly due to



(a) (b)

Figure 4.11: Area and average power consumption of a 4 by 4 SmartCell prototype system: (a) Area breakdown. (b) Average power consumption breakdown at 100 MHz.

the long instruction format and intensive controlling requirements. Furthermore, custom optimizations or library components are likely to reduce the on-chip memory size. The three-level hierarchical on-chip interconnections roughly take 14% of the total area.

The SmartCell can operate at a maximum frequency of about 123 MHz. Further investigation reveals that the single cycle MAC unit inside the arithmetic component takes about 5.5 ns of the total critical path delay, with 3.2 ns for the 18-bit multiplier and 2.3 ns for the 36-bit adder. Again, custom optimization can be performed to improve the timing result, such as pipelined multiplier and carry lookahead adder. In SmartCell prototype system, the full chip configuration can be completed within 13 microseconds at 100 MHz, which is much faster compared with the fine-grained FPGA configurations.

The power consumption of the SmartCell for different benchmarks is estimated in PrimePower based on netlist annotation from gate level simulations. Table 4.3 lists the power consumption (dynamic power P_{Dyn} and total core power P_{Core}) and energy efficiency (E_{Eff}) performance for eight benchmark applications. All figures are generated

	FIR	IIR	MMM	2D DCT	ME	FFT	PoE	RC5
P_{Dyn} (mW)	143	180	135	156	142	165	141	132
P_{Core} (mW)	152	189	143	165	151	174	150	140
E_{Eff} $(GOPS/W)$	42.1	33.9	11.2	38.8	44.1	18.3	42.7	45.7

Table 4.3: SmartCell power consumption and energy efficiency of different benchmarks at 100 MHz

at 100 MHz. Clock gating is automatically added by synthesis tool to dynamically turn off the clock for unused registers. This requires an enable signal, indicating whether the register is in use or not, to be attached to each register during the design stage. Fig. 4.11(b) shows the power dissipation for different components of the SmartCell system. The processing units consume about 41% of total power, with 33% for ALUs and 8% for logic components. The on-chip memories and interconnections consume another 53% of total power. On average, the SmartCell consumes about 158 mW at 100 MHz.

Finally, the energy efficiency, evaluated by the total number of operations per second per watt, is also calculated, as shown in Table 4.3. A peek performance of 45.7 GOPS/W is achieved in the RC5 data encryption application. SmartCell provides an average 34.6 GOPS/W energy efficiency from all benchmarks under test. The matrix multiplication shows only 11.2 GOPS/W energy efficiency. This is because the systolic array mapping scheme only uses 1/4 of the on-chip computing resources, while the other PEs still consumes power to bypass data. Another mapping scheme is studied to decompose the matrix multiplication into smaller row-column sub-blocks to be processed independently. It has the potential to improve the performance and energy efficiency. However, a closer look reveals that on-chip data memory is needed for the data reuse and propagation among the parallel computing. The mapping of sub-block matrix multiplication will be discussed in SmartCell-II in Chapter 6.

4.6 Comparison with FPGA and ASICs

In this section, we compare the SmartCell power and energy consumption performance with other computing platforms, including FPGA and ASIC. Table 4.4 gives the power consumption and system throughput of each benchmark, all generated at 100 MHz. Due to similar algorithm mapping structures and the same simulating frequency, the evaluated platforms can achieve the same system throughput for all benchmarks except 64-point FFT. Pipelined FFT is developed in both FPGA and ASIC implementations, which compute the 64-point FFT through 6 pipeline stages and generate 1 output per clock cycle after system initialization. While in our design, a parallel structure is mapped onto the SmartCell system, with 32 butterfly units running concurrently. Consequently, 60 clock cycles are required in the SmartCell to complete 1 block of 64-point FFT, which yields a throughput of 107 MS/s. Fig. 4.12 compares the power consumption results of these three platforms, which has been normalized to the results of ASIC implementations. As expected, the ASIC implementations outperform both SmartCell and FPGA. SmartCell is about 2.7 $\sim$ 15.6 less power efficient than ASICs. The maximum gap is observed in the RC5 application, which is mainly because only logic and addition operators are involved in ASIC implementation while multipliers are enabled in SmartCell even the results are not used. On the other hand, SmartCell outperforms FPGA by a factor of 2.7 $\sim$ 4.8 in terms of power consumption.

A more meaningful figure is depicted in Fig. 4.13 that compares the average energy efficiency (GOPS/W) among the evaluated platforms, normalized to FPGA result. As expected, the ASICs are the most energy efficient among the evaluated platforms. It provides an average 26.1x energy efficiency gain compared with FPGA results. However, this performance gain is achieved at a cost of no post fabrication flexibility and high engineering design cost. The energy efficiency of SmartCell falls somewhere in between. It is about 4.1x more energy efficient than FPGA but about 6.4x less than the ASIC implementations. The result demonstrates that the coarse-grained architecture is able to

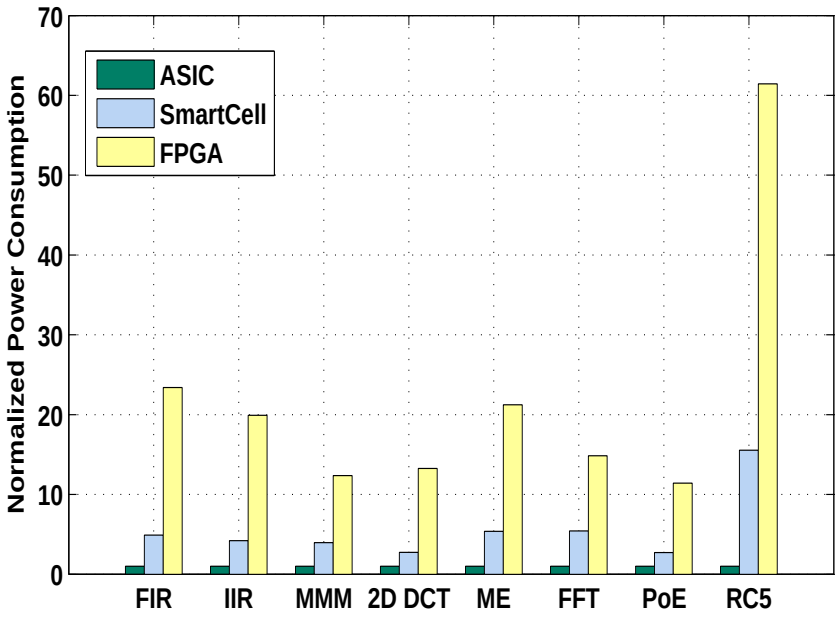


Figure 4.12: Diagram of power comparisons among SmartCell, FPGA and ASICs, normalized to ASIC results

fill the energy efficiency gap between the fine-grained FPGAs and logic specific ASICs.

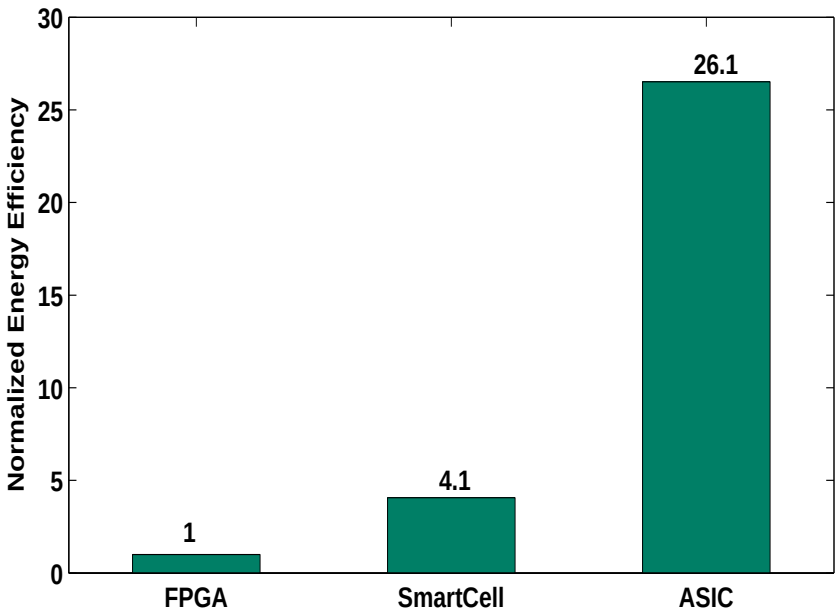


Figure 4.13: Diagram of power and energy efficiency comparisons among SmartCell, FPGA and ASICs, normalized to FPGA result.

	SmartCell		FPGA		ASIC	
	Power (mW)	Throughput	Power (mW)	Throughput	Power (mW)	Throughput
FIR	152	100 MS/s	725	100 MS/s	31	100 MS/s
IIR	189	100 MS/s	896	100 MS/s	45	100 MS/s
MMM	143	763 Metrics/s	445	763 Metrics/s	36	763 Metrics/s
2D-DCT	165	2.8 MBlocks/s	795	2.8 MBlocks/s	60	2.8 Mblocks/s
ME	145	3.5 MBlocks/s	573	3.5 MBlocks/s	27	3.5 Mblocks/s
FFT	174	107 MS/s	475	100 MS/s	32	100 MS/s
PoE	150	100 Ms/s	628	100 MS/s	55	100 MS/s
RC5	140	50 MBlocks/s	553	50 MBlocks/s	9	50 MBlocks/s

Table 4.4: SmartCell power consumption and energy efficiency of different benchmarks at 100 MHz

4.7 Comparison with other CGRAs

In this section, we compare SmartCell with some other CGRA systems, including Montium and RaPiD. Montium [45] occupies about 1.8 mm^2 silicon area with the same 0.13 μm technology as in SmartCell, while RaPiD [35] consumes about 5.7 mm^2 in 0.5 μm technology. The power consumption (PW in mW/MHz) of different benchmarks is given in Table 4.5. On average, Montium consumes 3.2x and 7.5x less power than SmartCell and RaPiD, respectively. However, direct comparison of power consumption does not mean much, due to different system configurations, hardware resources, computing precision, memory sizes, and etc. For the same reason, the amount of actual operations per second can not be easily generated to compare the energy efficiency as calculated in Section 4.6. Instead, a more realistic way is to compare the total energy consumption for the same amount of tasks. It provides a fair comparison of the relative energy efficiency among evaluated systems. The system throughput is also calculated and compared based on the number of clock cycles required to compute the same task.

As listed in Table 4.5, five benchmarks have been mapped onto the SmartCell system. Three of them are shared by the RaPiD and Montium, individually. Montium achieves the best power consumption performance, since only 5 ALUs are provided as on-chip computing resources. The cycle column (Cyc) in the table denotes the number of clock

	SmartCell			RaPiD[35]			Montium[45, 81]		
	PW[1]	Cyc[2]	EN[3]	PW	Cyc	EN	PW	Cyc	EN
2D DCT	1.65	36	59	4.29	64	275	0.5	96	48
ME	1.46	1156	1688	2.35	1156	2717	-	-	-
FFT	1.74	60	104	-	-	-	0.541	192	104
MMM	1.46	33K	48K	4.28	131K	561K	-	-	-
20-tap FIR	1.47	341	501	-	-	-	0.42	2057	864

Table 4.5: Power and Energy Comparison Among the Evaluated CGRA Systems

cycles needed to compute one data block, except for the FIR filter design. In the 20-tap FIR benchmark, 2 blocks of 512 samples are used to generate the cycle and energy figures, as done in [45]. The results demonstrate that in most applications, the SmartCell requires less clock cycles to finish the same amount of task comparing with the RaPiD and Montium implementations. This is benefitted from more processing parallelism provided by SmartCell with reduced computational and configuration complexity. For example, in SmartCell, three data pipes can be created to process the 20-tap FIR filter in parallel. On the other hand, a recursive processing scheme was adopted in Montium, since at most 5-tap FIR can be calculated at the same time. This recursive scheme also involves extra control and data exchange overhead. The energy consumption (EN in nJ) is also compared in Table 4.5, which is computed as the product of average power consumption and the processing time calculated by the cycle counts.

The normalized energy consumption is depicted in Fig. 4.14. When Montium is compared, SmartCell dissipates the same amount of energy in the 64-point FFT and consumes about 18.6% more energy in the 8 by 8 2D DCT. For the 20-tap FIR benchmark, 42.0% energy saving is observed in the SmartCell implementation. On the other hand, SmartCell always outperforms RaPiD with respect to energy consumption. A maximum 11.7x energy efficiency gain is achieved in the 128 by 128 matrix multiplication. On average, SmartCell is about 7.8% and 69.3% more energy efficient than Montium and RaPiD, respectively, for evaluated benchmarks. RaPiD integrated heterogenous computing and data storage components, in which case the power consumption may not be well balanced

among different modules. Its segmented interconnections and single control decoder may also involve high power consumption and low energy efficiency.

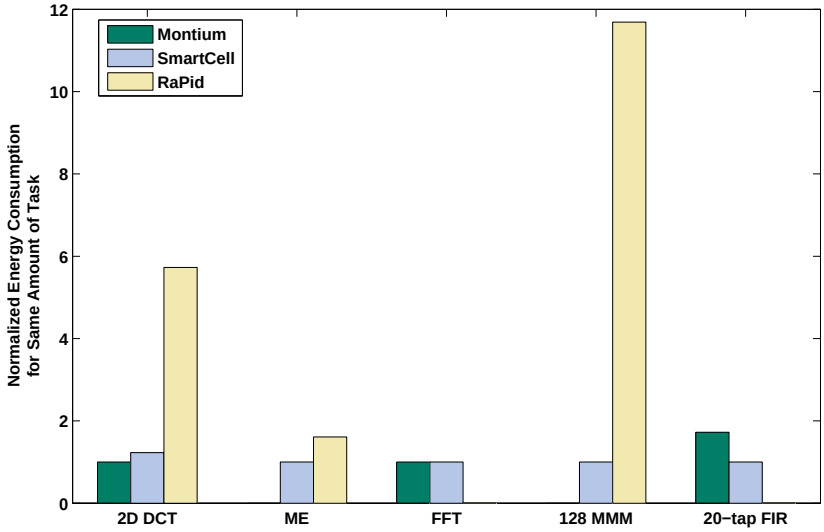


Figure 4.14: Diagram of energy consumption and throughput comparison among Montium, SmartCell and RapiD, normalized to SmartCell results.

Fig. 4.15 compares the normalized system throughput of different platforms. SmartCell and RaPiD provide same throughput in motion estimation application, due to similar algorithm mapping structures. SmartCell outperforms both Montium and RaPiD in all other benchmarks regarding to system throughput. In the FIR application, SmartCell is about 6x faster than Montium system. SmartCell also shows a maximum throughput gain of 4.2x over RaPiD system in the matrix multiplication implementations. Averagely, SmartCell provides about 4.0x and 2.2x throughput gains against the Montium and RaPiD implementations, respectively.

4.8 Summary

This chapter presented the synthesis results and performance evaluations of the first SmartCell design. A prototype system with 64 PEs was implemented with TSMC 0.13

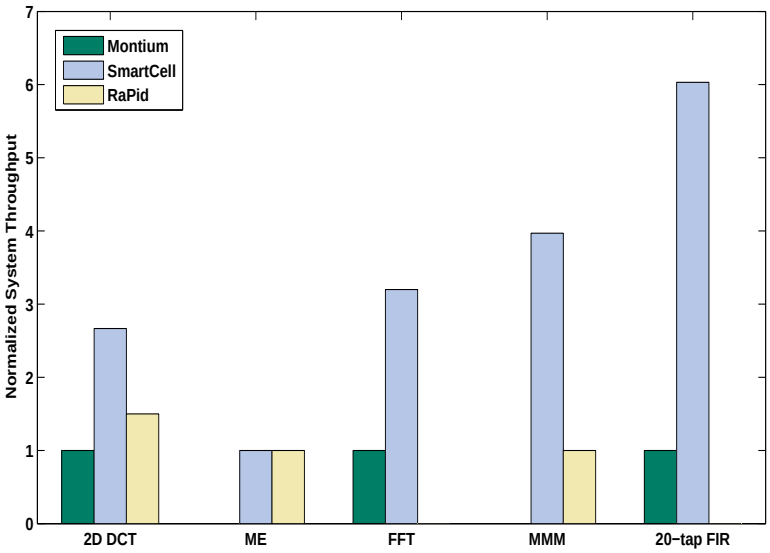


Figure 4.15: Diagram of throughput comparison among Montium, SmartCell and RapiD.

μm technology. It consisted of about 1.6 million gates with an average power consumption of 1.6 mW/MHz for the evaluated benchmarks. The results showed that the SmartCell is able to bridge the energy efficiency gap between the fine-grained FGPAs and customized ASICs. SmartCell achieves 4x and 2x throughput gains and is about 8% and 69% more energy efficient than Montium and RaPiD, respectively. The results demonstrate that SmartCell is a promising reconfigurable and energy efficient platform for stream processing.

Chapter 5

Design of SmartCell-II

The design of the first SmartCell prototype system with various application mapping experiences and performance evaluations demonstrates the powerful computing capacity and flexibility achieved in SmartCell. However, the design experience also reveals some architecture limitations. For example, lack of on-chip data memory makes it difficult and not efficient to map large applications directly onto SmartCell, which require memory storage of intermediate results. Also, the dynamic reconfiguration is not fully studied in our first SmartCell prototype. Based on these experiences, some modifications have been made in our second SmartCell design, called SmartCell-II. In this chapter, we firstly summarize these new features developed in SmartCell-II. The design details are presented thereafter.

5.1 New Features Developed in SmartCell-II

The initial motivation of SmartCell project is to design a reconfigurable architecture targeting high performance and energy efficient steam processing domain with high data parallelism and communication regularity. Although numerous mapping experiences showed that SmartCell is able to provide a promising solution for the targeted stream processing domain, some limitations were observed during our design, which hamper the exploration

of full capability of SmartCell. The on-chip data memory is not included in the first prototype, based on the fact that in data streaming applications, the intermediate results can generally be passed directly to next processing unit without local storage requirement. However, our experience shows that totally eliminating on-chip memory may become an obstacle for efficient application mapping or for implementation of large size applications. One example is found in matrix multiplication design. Due to no data memory available, a 2D systolic array structure is adopted that maps matrix multiplication onto a 4 by 4 systolic array structure. In this case, only 1 out of 4 PEs in the same cell is contributing to the real computation, while other PEs are bypassing data to the next cell. This systolic array structure does not take full advantage of the spatial computing ability provided by SmartCell and results in poor energy efficiency compared with other benchmarks.

The initial FFT mapping showed another example of the limitation without data memory. 32 Radix-2 butterfly units can be generated in the 4 by 4 SmartCell system to compute 64-point FFT in parallel. But due to lack of on-chip memories, for larger size FFTs, the intermediate results have to be output to the off-chip memories (test bench in our simulation) at the end of each processing stage and to be read back from memories at the beginning of the next stage. This structure greatly reduces the FFT scalability and degrades its performance. Besides the on-chip memory requirement, dynamic configuration is not fully studied and evaluated in our first SmartCell implementation. These issues are addressed in SmartCell-II.

The new features developed in SmartCell-II is summarized as follows:

- Distributed on-chip data memory:

  The targeted stream processing has a noticeable characteristic of limited memory requirement compared with traditional general tasks. The temporary results are consumed among processing units and do not need to be frequently exchanged through external memories. In SmartCell-II, each PE incorporates a 1K bits SRAM data memory, which can be addressed as 64 16-bit memory locations. Distributed data

memory structure is adopted to provide unified memory addressing modes and high scalability.

- Restructure of instruction format into sections:

  A new field is added to existing instruction code to handle the data memory read/write and address generation. As the control logic getting more complicated, the instruction format is breakdown into four sections: operation control, on-chip network control, program control and data memory control. Due to the control intensive nature, a prototype software compiler, named Smart_C, is proposed to facilitate the generation of configuration contexts. Since multiple components may share one or more instruction fields, different configuration fields can be separately generated in Smart_C. These fields can then be selected and combined together to form the entire instruction code for each PE.

- Two reconfiguration modes with memory partitioning for dynamic configuration:

  The dynamic reconfiguration is an important feature in CGRA designs. SmartCell-II revisits this issue and develops two configuration modes: coarse-grained cell broadcasting mode for SIMD operation and fine-grained ID based configuration for fine control of each processor in MIMD style. To further alleviate the unbalanced configuration delay along the SPI chain, a memory partitioning scheme is also developed to load new configuration contexts into unused instruction memories without interruption of current execution. A global select signal is then used to switch between different configurations in one clock cycle.

5.2 Design Details

5.2.1 Design of On-Chip Data Memory

In SmartCell-II, a limited 1Kb data memory is attached to each PE for data storage, since the processing data are generally resided and consumed locally for the targeted application domain. As shown in Fig. 5.1(a), the data memory is controlled by the same configuration context reading from the local instruction memory. The input and output of the data memory are connected to the on-chip connection. To be more specific, the PE and memory connection is depicted in Fig. 5.1(b). Each PE has full control of writing and reading of its own data memory. Moreover, the data read from a memory can be shared by up to two PEs to provide more data flow flexibility. A two-to-one mux is used to select one memory data as the PE's input, based on a 1-bit memory input control from the instruction code. The synchronous dual-port static RAM (DW_ram_r_w_s_dff), generated by Synopsys DesignWare library [13], is adopted to reduce area and power consumption of the on-chip memory blocks.

Multiple memory addressing modes are developed, as listed in Table 5.1. Four modes are supported to address the data memory. The direct memory addressing uses the 6-bit instruction address input ($addr$) to access the memory location, while the indirect memory addressing adds an reference address ($addr\_ref$) to $addr$ input. The immediate addressing mode adds $addr\_ref$ to the current memory location ($current\_addr$). At last, an immediate swap addressing mode is developed to switch memory accesses between lower and higher sub-blocks with an offset of half memory size. These memory accessing modes provide a flexible memory control scheme, which in turn improves the instruction code efficiency and helps to reduce the instruction memory size.

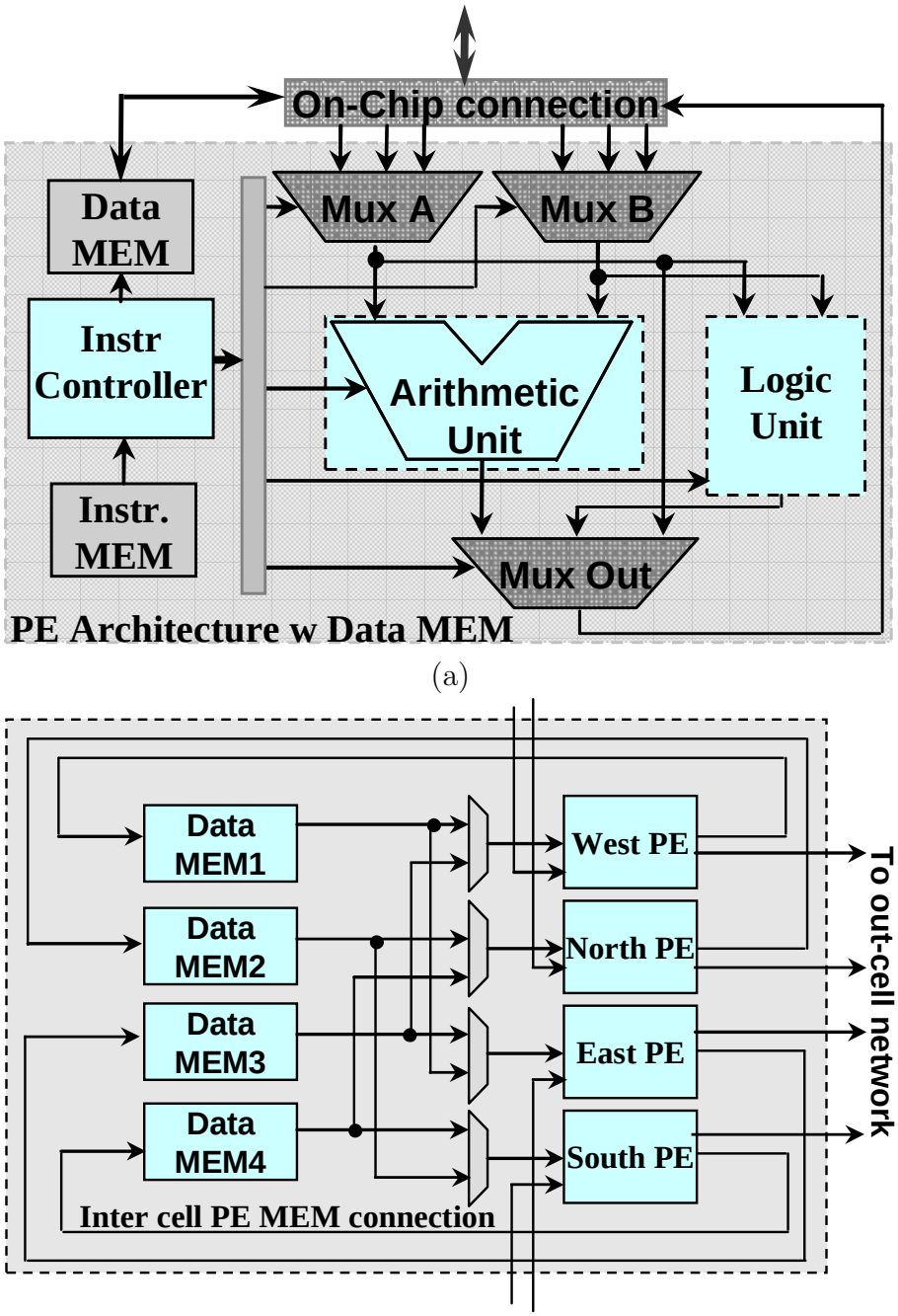


Figure 5.1: PE structure and local data memory connections in SmartCell-II. (a) PE structure with 1K data memory. (b) Inter cell PE and data memory connection.

Addressing mode	Control code	Memory address
Direct	00	$addr$
Indirect	01	$addr + addr\_ref$
Immediate	10	$current\_addr + addr\_ref$
Immediate Swap	11	$current\_addr + addr\_ref + mem\_size/2$

Table 5.1: Two access modes of data memory addressing in SmartCell-II.

5.2.2 Design of Segmented Instruction Format

As the control logic gets more complex in SmartCell-II, a 98-bit instruction code is designed with 4 separated configuration fields as shown in Fig. 5.2. The operation and datapath field controls the functional units, the dataflow and I/O scheduling at PE level. An 12-bit CMesh control field is developed for the on-chip CMesh network configurations, which include enable signal, switch fabric control, network input/output selection and PE selection. The PE selection indicates which PE(s) in the same cell is to send/receive data to/from the CMesh network. A 21-bit program control field is developed for instruction code scheduling, which provide the min and max active memory address, program counter (PC_max) and loop information. The PC_max provides the execution cycle count for current instruction code. In SmartCell-II, instruction loop is also supported to cyclically execute multiple contexts specified in the Min/Max address range. The $Loop\_number$ and $Loop\_EN$ fields indicate the iteration number and whether enabled or not. At last, a 25-bit memory control section is used to provide data memory configurations, with read/write addresses and accessing modes. The instruction loop and various data addressing modes in combination greatly improves the efficiency of the instruction code utilization and largely reduces the instruction memory size even for relatively large applications. Experience shows that 20 instruction codes are enough for 1024-point FFT mapping.

Partition of instruction code into sections leads to an intuitive insight that for software compiler design, it is possible to generate the configuration code for each field separately. These sub-fields can be then properly assembled to build the 98-bit instruction code. This might ease the configuration process and improve the code sharing capability.

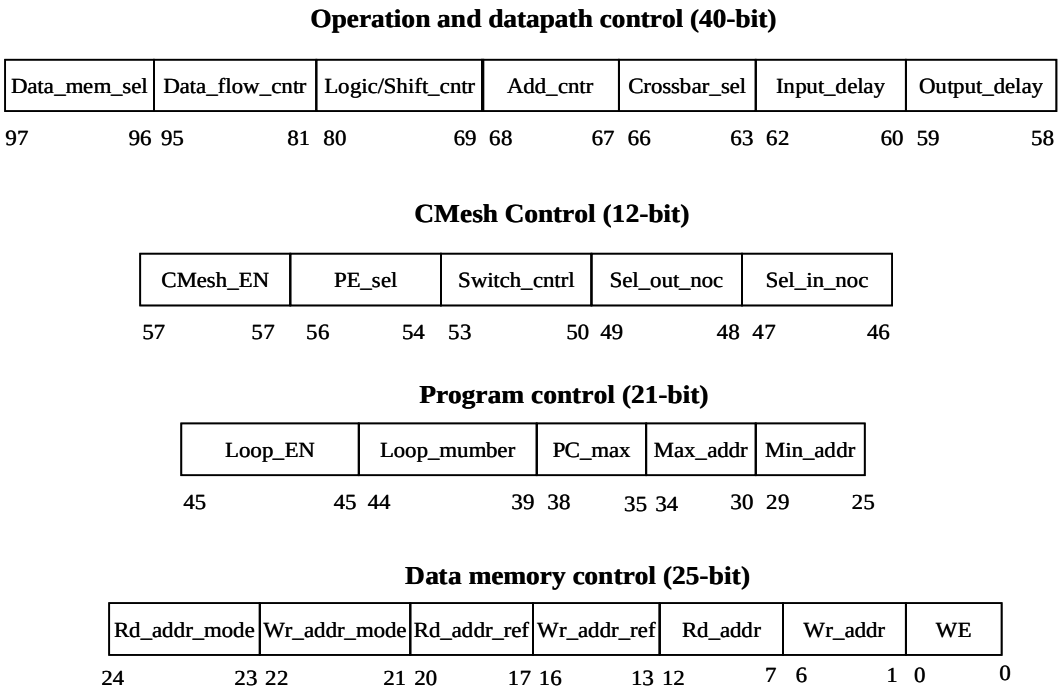


Figure 5.2: Segmented instruction format in SmartCell-II

5.2.3 Design of Two Mode Dynamic Reconfiguration

The dynamic configuration was not fully studied and evaluated in our first SmartCell architecture. In SmartCell-II, we designed a two-mode reconfiguration method in addition with a memory partitioning scheme to improve the on-line configuration performance. As discussed in Section 3.4, the on-chip instruction memories are chained from one to another through the SPI chain. The SPI structure is used for both instruction loading and updating.

In SmartCell-II, two reconfiguration modes are developed for fine-grained and coarse-grained configurations, as shown in Fig. 5.3(a) and (b). Some applications require fine control of individual PE to perform in MIMD style. The ID-based fine-grained configuration is used in this case. The new instruction code and the ID of the PE to be configured are sent into the SPI chain. PE bypasses the information to the next one until it reaches the desired PE. On the other hand, a group of PEs is configured to perform the same operation in the SIMD style for many other applications. To reduce latency, a cell broad-

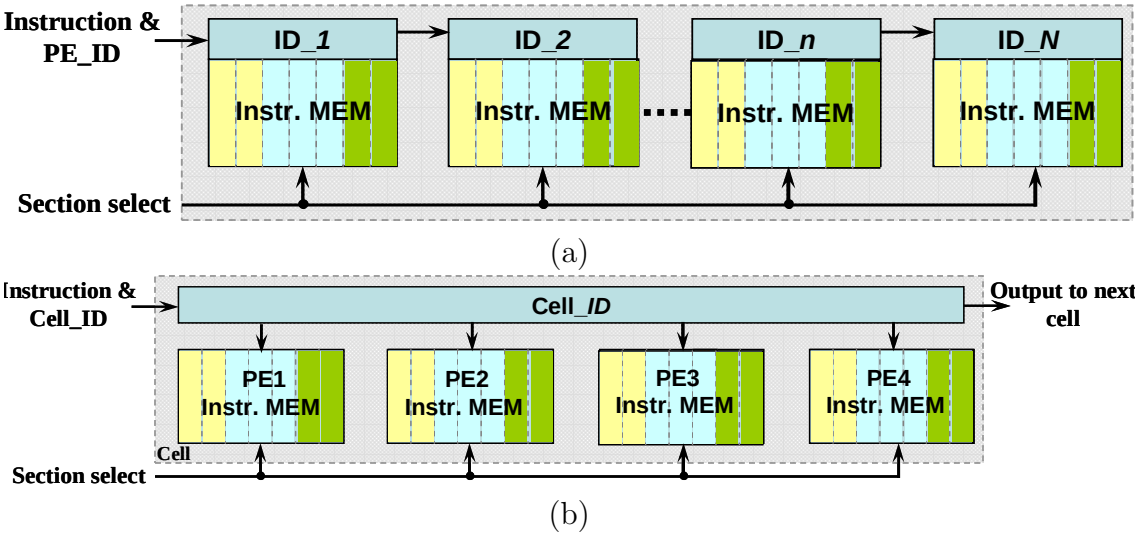
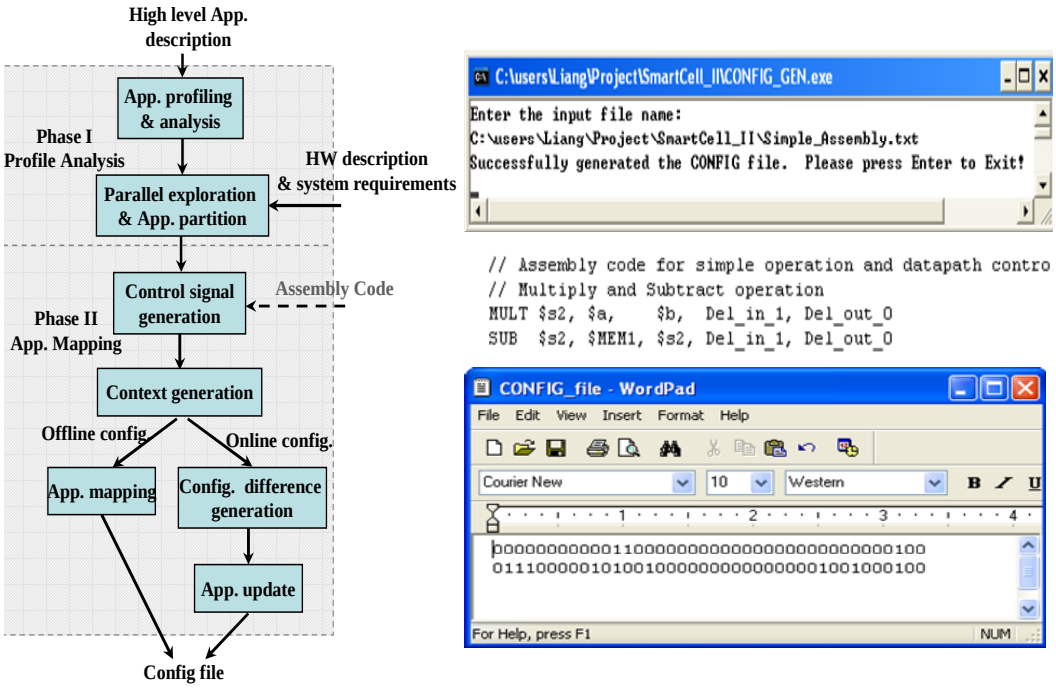


Figure 5.3: Diagram of dynamic reconfigurations in SmartCell-II. (a) ID-based fine-grained configuration. (b) Cell broadcasting coarse-grained configuration. A global select signal is developed for memory partitioning.

casting coarse-grained configuration is designed to concurrently write the reconfiguration contexts into all instruction memories in the same cell, based on the input Cell ID. In a 4 by 4 SmartCell system, 32 and 8 clock cycles are needed on average for an instruction code to reach the desired component in fine-grained and coarse-grained modes, respectively. However, the configuring propagation latency is not the same for different PE/Cell units along the SPI chain: the nearer to the input port, the faster can the configuration be done. To compromise this unbalanced configuration latency, a memory partitioning scheme is developed in our design. In this scheme, new instruction codes can be loaded into the unused context memories while the PEs are still operating in the current contexts. Once the new contexts are fully loaded, a global select signal is used to indicate the change of operation code. The configuration latency is effectively hidden by this means. Furthermore, multiple applications can be swapped within one clock cycle.

5.3 Propose of Smart_C Software Environment

Another important aspect in our research is to develop a software programming environment to assist automatic application mapping onto the SmartCell system. Enlightened by several existing compiler designs, including AppMapper [65] and SUIF [88], a software compiler, named Smart_C, is proposed to facilitate the configuration context generation in the targeted application domain. Fig. 5.4(a) represents the general application mapping flow of the Smart_C environment.



(a) (b)

Figure 5.4: Software design flow and application mapping environment for SmartCell. (a) Proposed Smart_C software Environment. (b) Example of operation and datapath context generation from assembly code.

Two phases are included in this design flow: application and architecture analysis phase (Phase I) and application mapping phase (Phase II). During Phase I, a high level application description file (preferably in C language) is input into the Smart_C environment. An application abstraction step is performed to parse the input application file and to extract the work loads from the input application. All candidate loops are broken

into linear sequences and the data dependencies among them are also analyzed during this step. On the other hand, a hardware description and system requirement file is also loaded into Smart_C to generate the hardware abstract, which specifies the computing resources and IO connections among them. At last, the parallelism/pipeline exploration and partitioning are performed to create task scheduling code based on the hardware and software abstracts. The communication datapath among active PEs are also scheduled here. The second application mapping phase transforms the scheduling code into configuring contexts that can be directly loaded into the instruction memories. At first, the control signals are determined for every computing and communication component to form the desired application datapath. Two modes are provided for offline and online configurations. In the offline configuration, the context file can be directly downloaded into the instruction memories for all active PEs. On the other hand, if the online configuration is performed, the differences between the current context and the generated one are examined. Only the PEs observing different contexts are needed to be updated.

In our current stage, the application mapping environment (Phase II) has been studied to generate the configuration contexts from an input assembly code. The four fields involved in a single instruction are generated separately based on the configuration library, which specifies the available computing operations and I/O models. Fig. 5.4(b) shows an example to generate the operation and datapath configurations from an assembly code. Given an application, the designer is responsible to properly partition the kernel operations onto the PE components and to exploit the data flows among them. After that, an application assembly code can be created to represent the computing and I/O models specified in the assembly libraries. According to these models, the context generator is able to automatically extract the control signals for the computing components from the input assembly file. Similarly, other fields can be generated based on the communication, data accessing and program scheduling requirements. These fields are then concatenated together to create the entire instruction code. The following steps remain the same as described earlier for both offline and online configurations. By this means, the configuration

overhead can be greatly reduced. The development of the system analysis phase (phase I) involves lots of experiences on system and task level profiling, workload analysis, task partitioning and redundancy optimization as usually found in complex compiler designs. Currently, the Smart_C software compiler is still in the initial stage, which points to an interesting direction in our future research.

5.4 Summary

In recognition of some system limitations from our first SmartCell prototype, this chapter presented SmartCell-II as an improved architecture design with several new features, including distributed data memory with various addressing modes, segmented instruction format and improved dynamic configuration schemes. A software compiler, named Smart_C, was also proposed to facilitate the automation of application mapping onto the SmartCell system.

Chapter 6

Matrix Multiplication and FFT on SmartCell-II

In this chapter, we present the mapping of matrix multiplication and FFT applications onto the SmartCell-II architecture. The mapping of other applications studied in Section 4.3 remains the same in SmartCell-II, since these applications do not require local data storage. A sub-block matrix multiplication scheme is adopted with both data-level parallelism and inner cell pipeline structure. For FFT application, a novel parallel algorithm is presented with optimized data flow pattern and high scalability. The performance analysis is also presented for these two applications.

6.1 Mapping of Sub-Block Matrix Multiplication onto SmartCell-II

A broad range of complex scientific and multimedia applications strongly depend on the performance of matrix-matrix multiplication. In this section, a new mapping scheme of matrix multiplication is applied on SmartCell-II, based on which the performance analysis is also provided.

Various methods have been proposed in the literature for high performance matrix multiplication designs, such as Cannon's algorithm [30], Strassen's algorithm [82], and more recently systolic algorithms using special systolic arrays. A sub-block matrix multiplication scheme is adopted in our design. In this scheme, the operand matrices are partitioned into smaller sub-matrices, each of which is then processed separately by different hardware resources in parallel. The result matrix is generated into sub-blocks of regular density matrix. This scheme can be efficiently mapped onto our SmartCell-II system to exploit both spatial and temporal parallelism to deliver higher computing performance compared with the systolic array structure mapping onto the first SmartCell prototype. At the same time, it achieves good data reusability among hardware resources, which avoids high bandwidth external memory requirement.

In our design, the operand matrices A and B are partitioned into sub blocks of 4 rows and 4 columns respectively, as shown in Fig. 6.1. They are then fed to the computational resources in column-major and row-major order, respectively, with the timing sequences from T1 to T4 denoted in the same figure. The independent 4 by 4 sub block results can be potentially calculated in parallel by different computing resources. Given the SmartCell architecture, each 4-row and column pair can be efficiently mapped onto the 4 PEs in the same cell unit.

6.1.1 Mapping Scheme

Mapping of the sub-block matrix multiplication onto a 4 by 4 SmartCell-II system is illustrated in Fig. 6.2. In this scheme, eight sub blocks of matrix A are concurrently input to the cell units and each sub block is shared between two vertically placed cells. Two sub blocks from B matrix are simultaneously broadcast to the rows of cells with one block shared by two rows. By this means, 16 sub blocks of the result matrix C can be computed in parallel.

For further performance optimization, inner cell pipeline structure is also exploited.

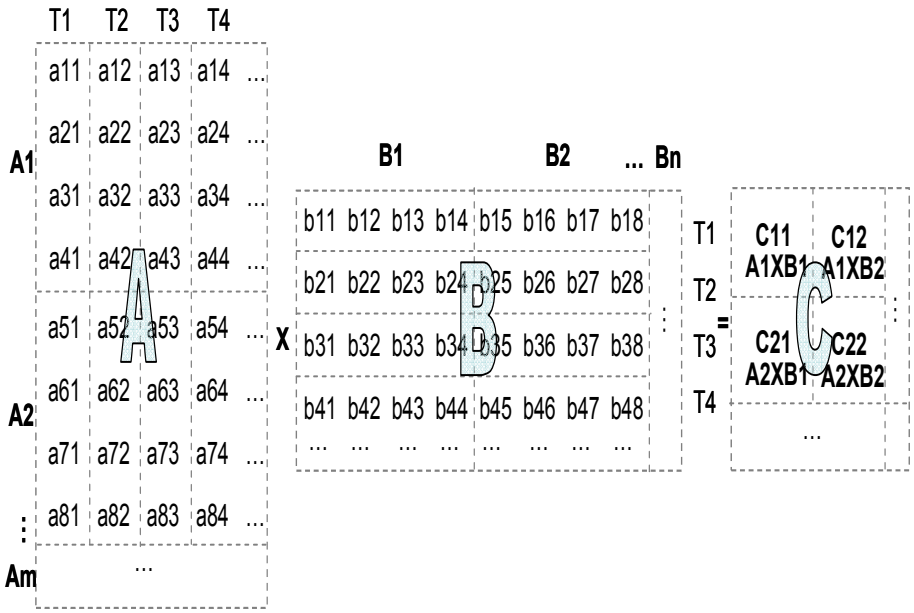


Figure 6.1: Illustration of sub-block matrix multiplication algorithm with timing information.

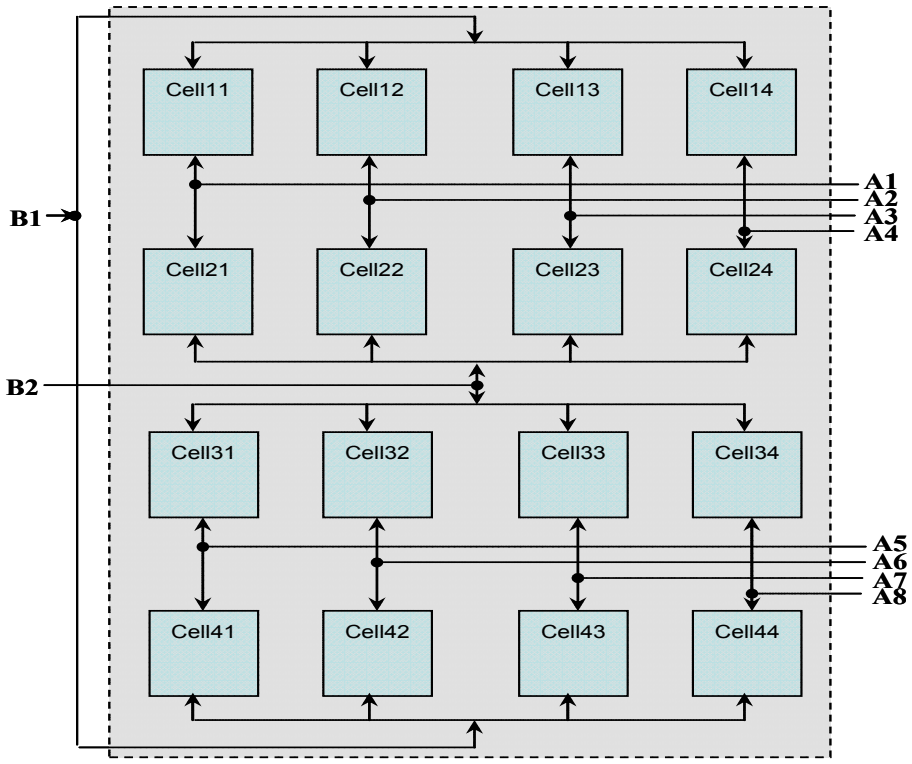


Figure 6.2: Mapping of sub-block matrix multiplication algorithm onto SmartCell-II.

The pipeline structure of one sub block result is illustrated in Fig. 6.3. Initially, element pair (a11, b11) is loaded into PE1 and generates partial result of c11. The calculated result is stored in local data memory that can be read back and accumulated during the next loop. After that, the second pair (a21, b12) is loaded into PE1 and PE2 along with previous stored input data for the computation of c21 and c12, respectively. The calculations of 4 rows in the result matrix are carried out in 4 pipes. Data used by the previous pipe is shared by the next pipe during the following time step through the crossbar unit. After 4 time steps, all pipes are filled up with computing data and are able to operate at full rate. In order to maintain full operation, this scheme only requires input of two external data in each step as highlighted in red circles in Fig. 6.3.

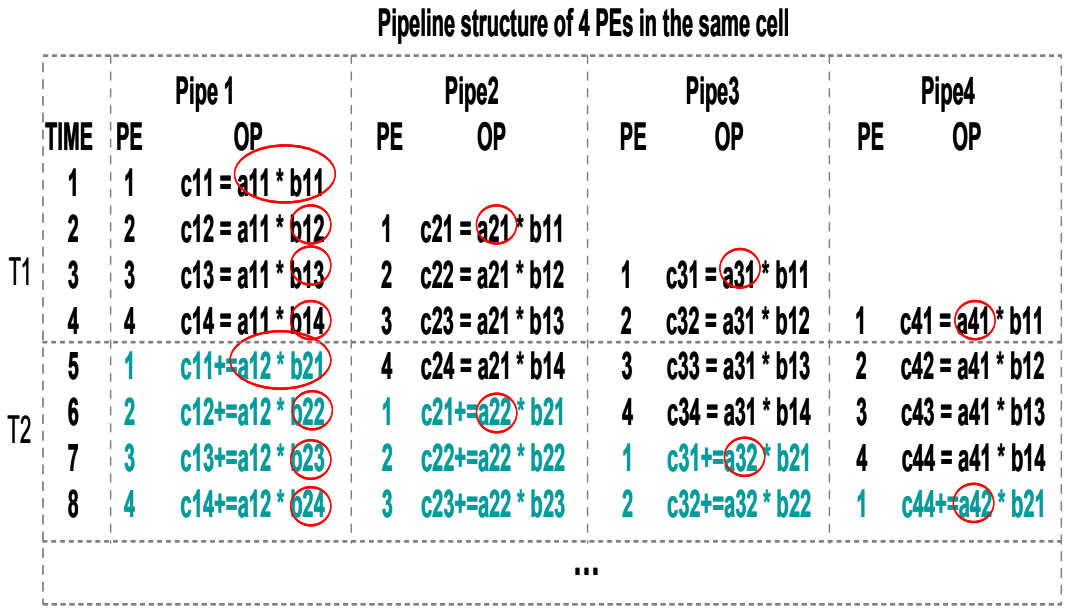


Figure 6.3: Pipelined computations for one sub-block result of matrix C. The data in red circle denotes the external inputs during each time step.

6.1.2 Performance Analysis

To evaluate its performance, a 128 by 128 square matrix multiplication is mapped onto a 4 by 4 SmartCell-II system. In general, each final element requires 128 clock cycles to finish the 128 MAC operations involved in it. Due to the fully pipelined structure, a 4 by

4 sub block result can be calculated in a single cell within 512 clock cycles. The final 128 by 128 matrix C is decomposed into 1024 independent 4 by 4 sub blocks, which can be calculated by the available 16 cells in parallel. Thus a total number of 32768 clock cycles is needed to compute a single 128 by 128 matrix multiplication, which leads to a system throughput of 12.2 KMatrices/s running at 100 MHz. This is 16 times faster than the 763 Matrices/s achieved by the systolic array mapping described in Section 4.3.

6.2 Parallel FFT Algorithm

Discrete Fourier Transform (DFT) is one of the most important digital signal processing algorithms in many communication and multimedia applications. An N-point DFT is defined in Eq. 6.1.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{6.1}$$

where $x(n)$ and $X(k)$ are the complex input and output, and the twiddle factor $W_N = e^{-2\pi i/N}$. Fast Fourier Transform (FFT) is a fast DFT algorithm that reduces the computing complexity from $O(N^2)$ to $O(Nlog_2(N))$. However, the intensive computations are still the bottleneck for large-size FFT/IFFT designs. Recently, several approaches have been proposed to compute the FFT in parallel on multi-processor architecture [21, 93, 56, 50, 66]. In this section, we propose a novel parallel FFT algorithm with evenly distributed computation tasks and optimized data transfer pattern among the processing units. It can be efficiently mapped onto SmartCell-II and be easily configured for different size of FFT applications.

6.2.1 Two-Stage Parallel FFT Algorithm

The Decimation-in-Time (DIT) Radix-2 Cooley-Tukey FFT [34] is chosen to be mapped onto the SmartCell-II architecture. Radix-2 FFT partitions the calculation of N-point DFT into $log_2(N)$ stages, each of which consists of a group of $N/2$ 2-point DFTs, also called

butterfly units (BUs). The complex number butterfly calculation is the basic operation involved in Radix-2 FFT. Comparing with the Decimation-in-Frequency (DIF) FFT, the DIT FFT achieves a more balanced computing load between the two branches in the butterfly operation, which makes it more popular in parallel FFT designs. Although Radix-4 and Radix-8 FFTs are available and require fewer computing stages, they are not adopted in our design due to higher configuration and communication complexities.

Two steps are developed in the proposed algorithm to compute the FFT transformation in parallel: local sequential execution and cross parallel execution. Before the transform begins, the N points input data are evenly distributed to the P butterfly processing units, with N/P points stored in each BU. During the local sequential execution, the first $log_2(N/2P)$ stages are computed sequentially in each BU operating on N/P locally stored data. Since each butterfly operation consumes 2 points at a time, we assume both N and P are power of 2 and satisfy $N \geq 2P$ and $P > 1$, without loss of generality. After the sequential step, a cross parallel execution is performed for the rest $log_2(2P)$ stages of N-point FFT. Cross communication among different BUs are necessary to exchange data that are more than N/P apart.

Algorithm 2 demonstrates the operations involved in the local sequential execution for each BU. Before the transform starts, the N-point signals are bit reversed and partitioned into P sub-blocks. Each N/P sub-block data are fed into one butterfly unit sequentially, denoted as $c[0]$ to $c[N/P-1]$ in the algorithm. After the data are completely loaded, the first $log_2(N/2P)$ stages of the original N-point FFT is computed locally inside each BU. The butterfly function $f(c_1, c_2, w)$ is defined in Eq. 6.2, where c_1 and c_2 are two input branches, and w is the twiddle factor. Two input data are read from two consecutive memory locations and are written back into two memory locations that are $N/2P$ apart after processing. All calculations are completely isolated with pure local data accessing. Thus no cross BU communications are needed during the sequential step. Data level parallelism is achieved by computing P sub-blocks concurrently in different BUs.

$$f(c_1, c_2, w) = c_1 + w \times c_2 \tag{6.2}$$

Algorithm 2 : Local sequential $log_2(N/2P)$ Stages

```
1: Input: c = c[0], c[1],..., c[N/P − 1]
2: Output: c = c[0], c[1],..., c[N/P − 1]
3: for i ← 0 to log2(N/2P) − 1 do
4:    for j ← 0 to N/2P − 1 do
5:       get w1, w2 based on i, j
6:       c[j] = f(c[2j], c[2j + 1], w1)
7:       c[j + N/2P] = f(c[2j], c[2j + 1], w2)
8:    end for
9: end for
```

The cross parallel step calculates the remaining $log_2(2P)$ stages of the original N-point FFT, as shown in Algorithm 3. During this step, the intermediate butterfly results need to be transferred among BUs. We use BU_ID to represent the index number of current BU ranging from 0 to $P-1$. Two cases are developed based on the number of FFT points N and the number of butterfly units P. When N is equal to $2P$, each BU only processes two points involved in the butterfly operation. No sequential step is necessary in this case. The computing and data transfer pattern are listed in Line 25 to 35 in Algorithm 3. On the other hand, when N is greater than $2P$, more than 2 points are stored and processed by each BU, which requires both sequential and parallel steps. The related cross parallel execution is listed in Line 6 to 23. The initial iteration($i == 0$) calculates the last stage of the local N/P-point FFT. After that, the data fetching pattern is changed to read from two memory locations that are $N/2P$ apart due to cross BU communications. In general, during the cross parallel execution, each BU calculates the butterfly results from its local data and then forwards the results to two BUs based on its own BU_ID.

Algorithm 3 : Cross Parallel $log_2(2P)$ Stages

1: **Input:** $c = c[0]$, $c[1]$,..., $c[N/P-1]$
2: **Output:** $c = c[0]$, $c[1]$,..., $c[N/P-1]$
3: **for** $i \leftarrow 0$ to $log_2(2P)-1$ **do**
4: **for** $j \leftarrow 0$ to $N/2P-1$ **do**
5: **if** $(N > 2P)$ **then**
6: **if** $(i == 0)$ **then**
7: $t_1 = c[2j]$
8: $t_2 = c[2j+1]$
9: **else**
10: $t_1 = c[j]$
11: $t_2 = c[j+N/2P]$
12: **end if**
13: get w_1, w_2 based on i, j and BU_ID
14: $k = BU\_ID$
15: $r_1 = f(t_1, t_2, w_1)$
16: $r_1 = f(t_1, t_2, w_2)$
17: **if** $(k \ mod \ 2 == 0)$ **then**
18: send $r_1 \rightarrow c[j]$ of $(k/2)^{th}$ BU
19: send $r_2 \rightarrow c[j]$ of $(k/2+P/2)^{th}$ BU
20: **else**
21: send $r_1 \rightarrow c[j+N/2P]$ of $((k-1)/2)^{th}$ BU
22: send $r_2 \rightarrow c[j+N/2P]$ of $((k-1)/2+P/2)^{th}$ BU
23: **end if**
24: **else**
25: get w_1, w_2 from i and BU_ID
26: $k = BU\_ID$
27: $r_1 = f(c[0], c[1], w_1)$
28: $r_1 = f(c[0], c[1], w_2)$
29: **if** $(k \ mod \ 2 == 0)$ **then**
30: send $r_1 \rightarrow c[0]$ of $(k/2)^{th}$ BU
31: send $r_2 \rightarrow c[0]$ of $(k/2+P/2)^{th}$ BU
32: **else**
33: send $r_1 \rightarrow c[1]$ of $((k-1)/2)^{th}$ BU
34: send $r_2 \rightarrow c[1]$ of $((k-1)/2+P/2)^{th}$ BU
35: **end if**
36: **end if**
37: **end for**
38: **end for**

6.2.2 Data Transfer Pattern

Besides the computational tasks, data transfer is another important consideration for high performance parallel FFT design. Efficient data transfer plays a key role in reducing com-

munication delay and overhead, which in turn improves the overall system throughput. In this section, we analyze the data transfer pattern during the cross parallel execution. No cross BU communications are involved in the sequential execution step, since all computations only involve data stored in the local data memory.

The data transfer is described in Line 17 to 23 and Line 29 to 35 in Algorithm 3. Given the total number of points N and butterfly units P, the destination BUs are only determined by the current ID k, while the destination memory locations are determined by the ID k and the iteration number j. The destination BUs and memory locations are independent on the current FFT stage i. Thus the proposed FFT algorithm achieves a fixed data transfer pattern at all FFT stages. Fig. 6.4 depicts a data flow example of 8-point FFT with 4 butterfly units. Each BU stores two input data in bit reversed order. According to the algorithm, BU0 calculates the butterfly results from its local data and sends them to memory 0 of BU0 and BU2. The same data flow is repeated during stage 2 and 3. Other BUs follow similar fixed data transfer pattern as depicted in the figure. The two-line crossings in the same color represent the butterfly operations executed by the same BU. The traditional data transfer pattern for 8-point Radix-2 FFT is drawn in Fig. 6.5(a). In this case, the computing tasks are horizontally partitioned among the BUs, as highlighted in gray boxes in Fig. 6.4. From stage 1 to stage 2, data are transferred between (BU0, BU1) and (BU2, BU3) pairs. The pattern changes to (BU0, BU2) and (BU1, BU3) pairs during the transition from stage 2 to stage 3. On the other hand, a fixed data transfer pattern is achieved for all transform stages in the proposed parallel FFT algorithm, as shown in Fig. 6.5(b). The optimized transfer pattern reduces both communication and configuration overhead, especially for large size FFTs.

6.2.3 FFT Mapping and Performance Analysis

The Radix-2 FFT butterfly calculation is formulated in Eq. 6.3. The complex number operation can be optimized into four real multiplications and six real additions. In our

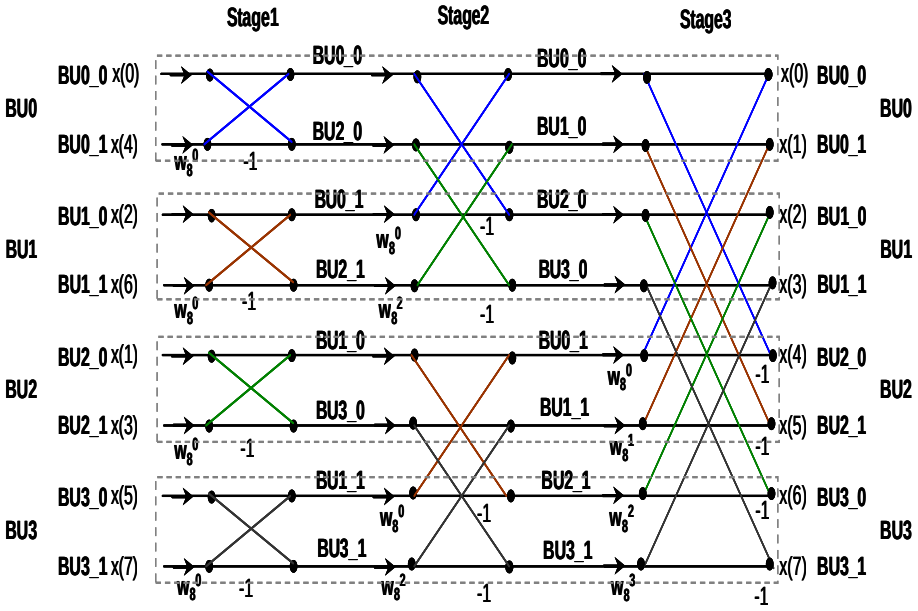
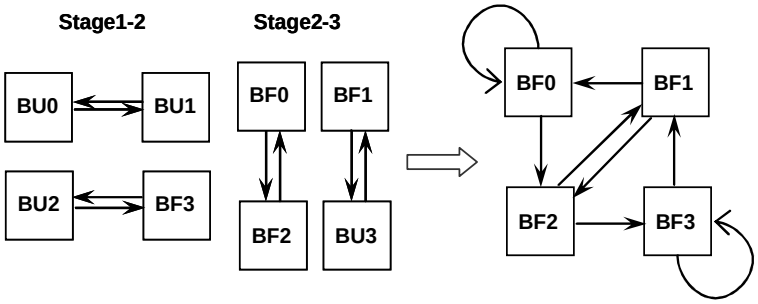


Figure 6.4: An example of Radix-2 8-point FFT butterfly structure with data flows.

design, two PEs are used to calculate the butterfly results in a sequential manner. Fig. 6.6 depicts one possible implementation with data flow scheduling. The real and imaginary parts of the output signals can be processed in parallel. The intermediate results are shared between these two PEs through the inner cell crossbar unit. In this way, two butterfly units can be mapped onto the 4 PEs in the same cell. To perform FFT on SmartCell-II architecture, the input data are partitioned into consecutive chunks, each of which is then loaded into the data memories of two PEs performing the same butterfly operation. The sequential execution starts after the input data are fully loaded. Since the data processing is isolated inside the local PEs, no cross cell communications are involved during this step. After the local sequential step is finished, the FFT butterfly needs to transfer data between PEs in multiple cells. Due to the fixed communication pattern, the intermediate results can be efficiently exchanged through either nearest neighbor connection or CMesh network within 1 hop distance. No extra communication and configuration overhead are involved.



(a) (b)

Figure 6.5: Data transfer pattern for 8-point FFT. (a) Traditional communication pattern. (b) Optimized fixed data flow in the proposed FFT alrogithm.

$$
\begin{aligned}
R_{A'} &= R_A + R_B R_W - I_B I_W \\
I_{A'} &= I_A + I_B R_W + R_B I_W \\
R_{B'} &= R_A - R_B R_W + I_B I_W \\
I_{B'} &= I_A - I_B R_W - R_B I_W
\end{aligned}
\tag{6.3}
$$

Scalability is an important performance criterium for FFT designs. For the proposed parallel FFT algorithm, SmartCell-II is able to implement FFT of any sizes as long as the input data can be fully stored into the on-chip data memories. Larger size FFTs are able to be implemented with the help of additional external memories and control logic. By changing several loop control parameters, SmartCell-II can be easily reconfigured to compute a different size FFT. To analyze its performance, a 1024-pt FFT is mapped onto a 4 by 4 SmartCell with 1K data memory attached to each PE. In this case, 32 butterfly units are available to compute the FFT in parallel. The first 4 stages of the 1024 FFT are carried out locally in each cell with 6 cycles for a single butterfly operation. During the cross parallel execution, the butterfly operation still takes 6 cycles to finish since all communication is within one hop distance in the CMesh network. Simulation results show that a block of 1024-pt FFT can be finished in 992 clock cycles, which leads to a system throughput of 101 KBlocks/s operating at 100 MHz.

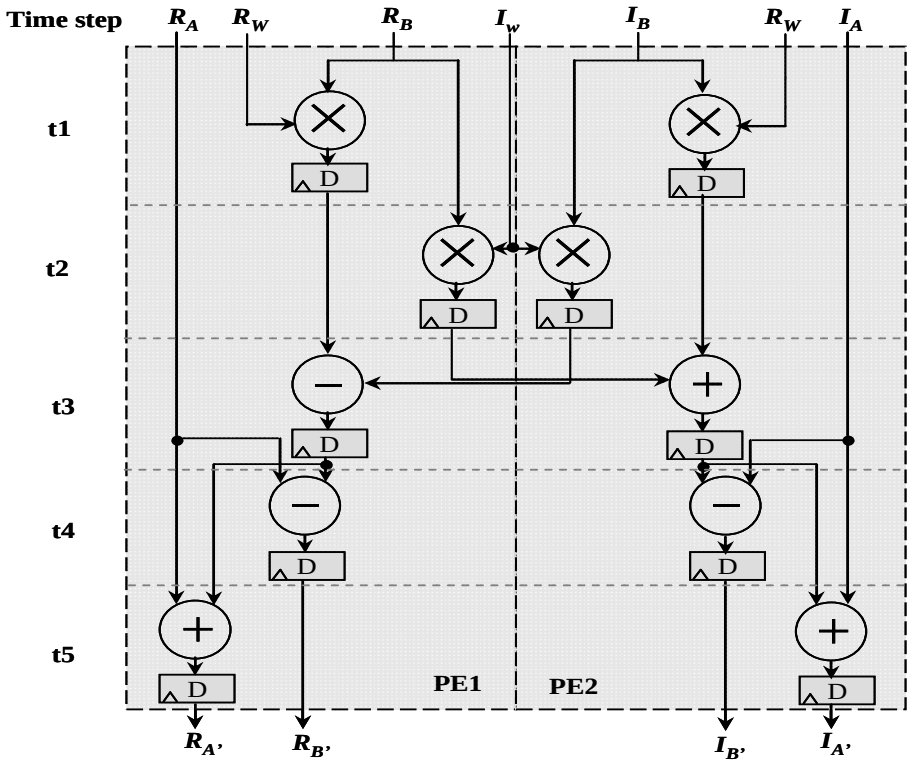


Figure 6.6: Mapping of butterfly operation onto two PEs in one cell.

6.3 Summary

This chapter presented the mapping of two computing intensive applications onto SmartCell-II system, including matrix multiplication and FFT. The sub-block matrix multiplication algorithm was adopted to exploit both temporal and spatial parallelism on SmartCell. The new matrix multiplication mapping achieved a 16x throughput gain, compared with the first SmartCell prototype results. For FFT application, a two-stage parallel algorithm was proposed to provide balanced workload among processing units with fixed data flow pattern among different processing stages. The mapping of proposed FFT algorithm onto SmartCell-II was also discussed in details.

Chapter 7

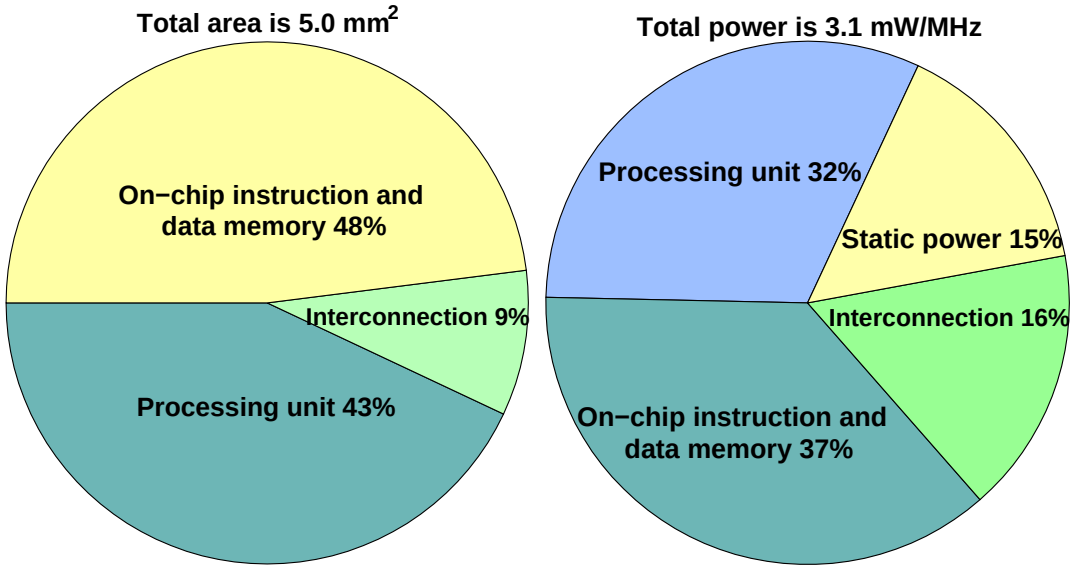
SmartCell-II Experimental Results and Evaluations

A 4 by 4 SmartCell-II system was developed and synthesized in standard cell ASICs, with similar design and verification methodologies described in Section 4.1 and 4.2. The area and timing performance was provided based on the synthesis reports. The proposed parallel FFT algorithm was manually mapped onto the prototype system for performance evaluations. Up to 1024-pt FFT can be directly implemented in our current design. We also compared the system throughput with some other FFT implementations, including FPGA, DSP, NoC [21] and MorphoSys [50] platforms. The reason to choose NoC and MorphoSys FFTs is that both of them present parallel FFT implementations with the same amount of processing elements as used in the SmartCell system. At last, the energy efficiency was compared among SmartCell, Xilinx Virtex II Pro FPGA [89] and TI's C6x DSPs [47] based on these FFT benchmarks.

7.1 Synthesis Results

SmartCell-II was developed and synthesized with standard CAD tools. A functional RTL model was firstly designed in hardware description language (HDL) and was then synthesized in Synopsys DesignCompiler to generate the CMOS standard cell ASICs using TSMC 90 nm technology. The area and timing results were generated by DesignCompiler using worst case conditions.

According to the synthesis results, the prototype SmartCell-II occupies about 5.0 mm^2, which is about 2.0 million gates. The system area, separated into PE, on-chip memory and interconnections, is shown in Fig. 7.1(a). About half of the total area is consumed by the on-chip data and instruction memories. The processing units and the hierarchical interconnection networks roughly consist of 43% and 9% in the total area, respectively. Experiments showed that the FFT benchmarks were able to fully operate at up to 295 MHz. In terms of dynamic reconfiguration, SmartCell-II can be reconfigured to a different size FFT in 90 clock cycles, which is within 1 μs running at 100 MHz.



(a) (b)

Figure 7.1: Area and average power consumption of the SmartCell-II prototype. (a) Area breakdown (b) Average power consumption breakdown at 100 MHz.

The power consumption of SmartCell-II for the evaluated FFT benchmarks is estimated in Synopsys PrimePower based on the netlist annotation from gate level simulations. Fig. 7.1(b) shows the processing units consume about 32% of total power, while the on-chip memories consume about 37% of the total power. Due to technology shrink, the static power increased from 6% to 15% compared with our first implementation described in Section 4.5. On average, SmartCell-II consumes 3.1 mW/MHz for the evaluated FFT benchmarks.

Table 7.1 compares the synthesis results between the two SmartCell prototype systems. In SmartCell-II, the gate count has increased from 1.6 million to 2.0 million mainly caused by the on-chip data memory increases. For the same reason, more power is consumed by SmartCell-II. The maximum frequency achieved in SmartCell-II is about 2.4 times higher than the first implementation because of design optimizations and process shrinking. The dynamic configuration of SmartCell-II is also 10 times faster than before benefitting from the fine and coarse-grained configuration schemes.

	SmartCell	**SmartCell-II**
Process	130 nm	90 nm
Dimension	4 by 4	4 by 4 with 64K data memory
Gate Count	1.6 Million	2.0 Million
Power	1.6 mW/MHz	3.1 mW/MHz
Max Freq.	123 MHz	295 MHz
Dyn. Config @ 100MHz	$\sim$10 μs	$\sim$1 μs

Table 7.1: Comparison of two SmartCell prototype systems.

7.2 Comparison with FPGA

In this section, we compare the system throughput and energy efficiency results between SmartCell-II and FPGAs. The Xilinx's Virtex II Pro XC2VP20 FPGA [89] was selected as the benchmark platform. The FFT of different sizes was generated from CoreGen in Xilinx's ISE 10.1 CAD tool [11]. Pipelined FFT architecture was adopted in the FPGA

implementations with $log_2(N)$ butterfly stages chained in a pipeline structure. Synthesis results showed that a maximum frequency of 256 MHz can be achieved by FPGA for the evaluated FFT benchmarks.

Fig. 7.2 compares the per-block FFT processing time between FFT and SmartCell-II. SmartCell-II is able to compute a 64-pt FFT in 38 cycles, while 64 cycles are needed in FPGAs. With both running at maximum frequencies, SmartCell-II achieves a maximum throughput gain of 1.9 compared with FPGA. On average, SmartCell-II is about 1.5 times faster than FPGA implementations for the evaluated benchmarks. Moreover, due to its unbalanced memory loads and intensive control requirements, the pipelined FFT is not suited for reconfigurable architectures especially when the number of points needs to be changed on the fly.

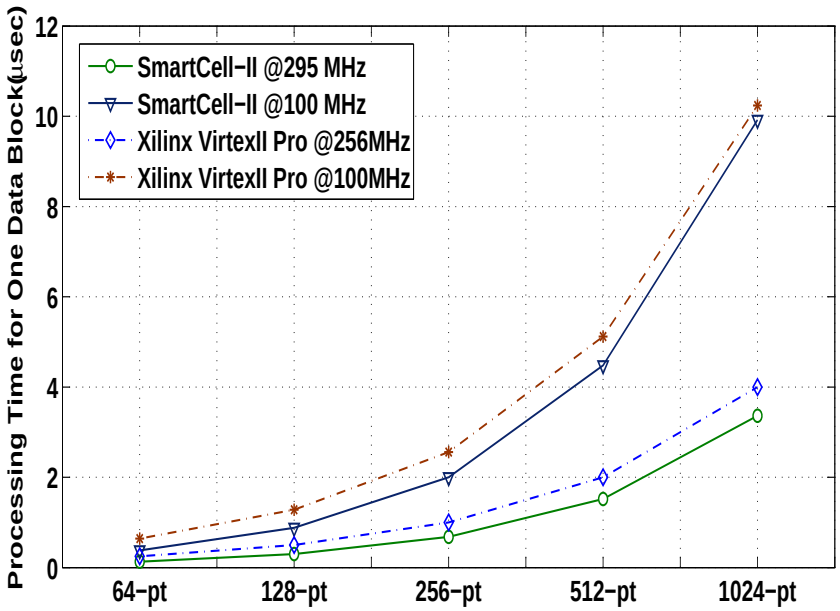


Figure 7.2: Processing time comparison between SmartCell-II and FPGA for the evaluated FFT benchmarks.

Fig. 7.3 compares the FFT energy consumption for one data block between SmartCell-II and FPGA operating at 100 MHz. The Xilinx XPower Analyzer was used to evaluate the FPGA power consumption based on the switching annotation from gate level simulations. Only the core power consumption was recorded in FPGA designs for fair comparison.

In 64-pt FFT, SmartCell-II consumes 77.3% less energy than FPGA, since limited data memories are used in SmartCell-II for this case. On average, SmartCell-II is about 3.6 times more energy efficient than the fine-grained FPGA. Results show that SmartCell-II achieves an average energy efficiency of about 20.6 GOPS/W for all benchmarks under test.

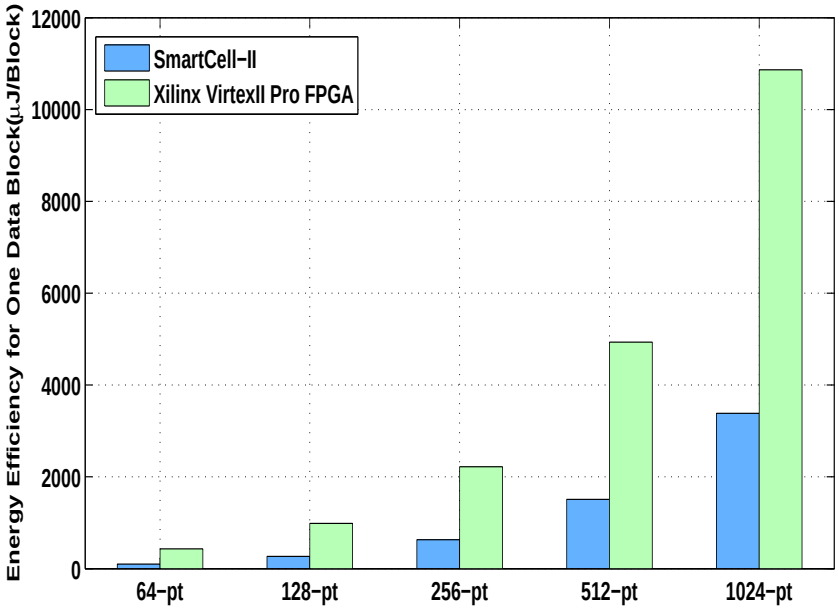


Figure 7.3: Energy consumption comparison between SmartCell-II and FPGA.

7.3 Comparison with DSP

The system throughput and energy efficiency for the FFT benchmarks were also compared between SmartCell-II and general purpose DSPs, including TI's TMS320C6203^{TM} and TMS320C6713^{TM} [47]. The reason to choose these two DSPs to compare with was that they are targeted at high performance signal processing applications through the advanced VLIW architecture and the Radix-2 FFT benchmark results are provided by the vendor [14, 15]. The 8-way VLIW structure includes 8 data processing units that can be configured to perform 8 MAC operations at the same time. Fig. 7.4 compares the FFT processing time for one data block between SmartCell-II and DSPs. We assume the DSP is

operating at the highest frequency of 300 MHz specified in the data sheet. In terms of cycle counts, TMS320C6713 needs 20522 cycles to compute 1024-pt FFT, while SmartCell-II only requires 992 cycles. When the maximum frequency is used, SmartCell-II is about 20.8 times faster than the DSP based implementations. According to datasheet, the typical core power consumption for TMS320C6203 and TMS320C6713 is about 5.3 mW/MHz and 4.3 mW/MHz, respectively. Fig. 7.5 compares the per-block energy consumption between SmartCell-II and DSPs. The results showed that, on average, SmartCell-II is about 36.8 and 28.9 times more energy efficient than TMS320C6203 and TMS320C6713 DSPs, respectively.

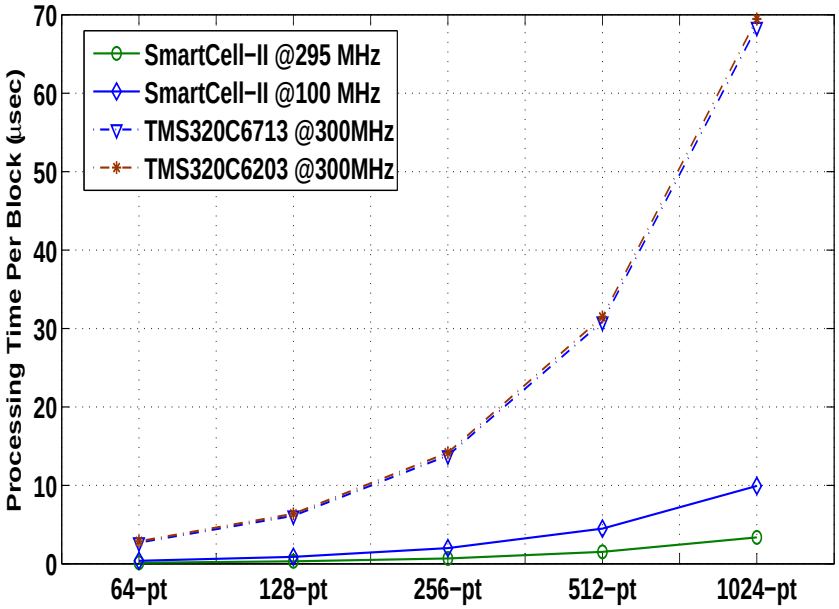


Figure 7.4: Processing time comparison between SmartCell-II and TI DSPs for the evaluated FFT benchmarks.

7.4 Comparison with other Parallel FFT Platforms

At last, we compared the system throughput of SmartCell-II with other parallel FFT implementations, including NoC [21] and MorphoSys [50] platforms. Due to different operating frequencies, the number of clock cycles required to finish one data block was compared.

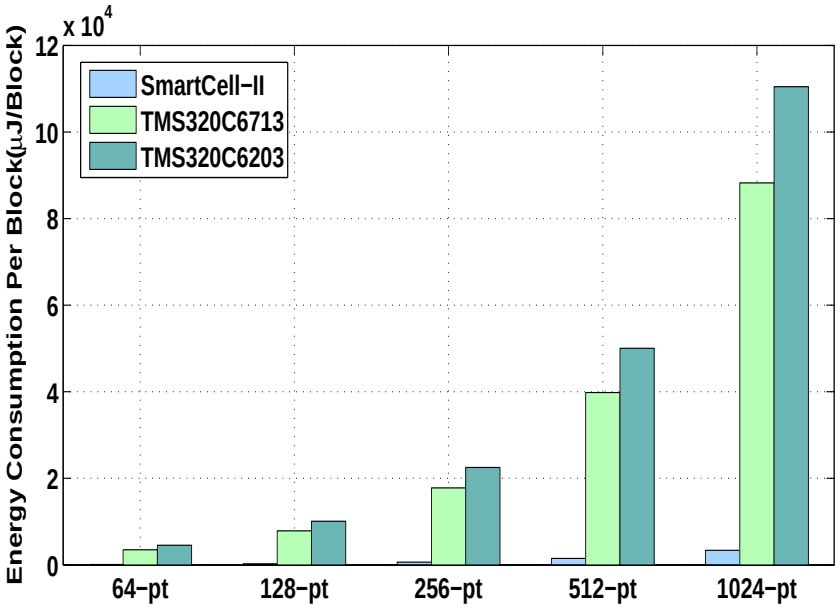


Figure 7.5: Energy consumption comparison between SmartCell-II and TI DSPs.

Table 7.2 lists the system throughput among different FFT platforms. SmartCell-II outperforms both NoC and MorphoSys platforms in all benchmark FFTs regarding to system throughput. SmartCell-II achieves a maximum 24.1 times throughput gain than the NoC implementation in 256-pt FFT. On average, SmartCell-II is about 2.7x faster comparing with the coarse-grained MorphoSys implementation. These performance gains are mainly benefited from reduced communication and configuration overhead by the proposed parallel FFT algorithm.

FFT Size	SmartCell-II	MorphoSys [50]	NoC [21]
64	38	111	-
128	88	225	-
256	200	520	4827
512	448	1222	5702
1024	992	2613	7726

Table 7.2: Cycle counts comparison among different parallel FFT platforms.

7.5 Summary

In this chapter, we presented the prototype results of the SmartCell-II system with 64 PEs implemented in standard cell ASICs with TSMC 90 *nm* technology. The proposed FFT algorithm with different sizes were mapped onto the prototype SmartCell-II device. For the evaluated FFT benchmarks, SmartCell-II dissipates about 310 mW power operating at 100 MHz achieving an energy efficiency of 20.6 GOPS/W. Comparing with other FFT implementations, SmartCell-II is about 14.9 and 2.7 times faster than the parallel FFT implementations in NoC and MorphoSys, respectively. SmartCell-II is also about 3.6 and 28.9 times more energy efficient than the FPGA and DSP based implementations. The results again demonstrate that SmartCell is a promising reconfigurable and energy efficient computing platform for data streaming applications.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This dissertation presents the SmartCell architecture design, application mapping, performance analysis and evaluation of a coarse-grained reconfigurable architecture for stream processing applications.

SmartCell integrates a large number of homogenous cell units onto the same chip in a 2D mesh structure. In each core, four processing units are placed at the four edges, attached with their own instruction and data memories. Benefiting from this unique system structure, a hierarchical configurable network has been developed in SmartCell, which includes three level of interconnections: fully connected crossbar inside a cell, nearest neighbor connection among adjacent cells and clustered mesh network for non-adjacent cell units. The cell broadcasting and ID-based configurations were also developed for dynamic reconfiguration to address various control requirements. SmartCell are flexible to exploit deep pipeline and large amount of parallelism with various operation modes. In combination of these features, SmartCell is able to achieve high energy efficiency while maintaining high computational performance, which is well suited for the computing intensive data streaming applications with stringent power budget.

In our research, a set of benchmark applications has been successfully designed and mapped onto the SmartCell system, representing a wide range of real-time applications from signal processing, multimedia application to scientific computing and data encryption. To achieve best performance, SmartCell was designed to operate under various modes in these benchmarks, including pipelined structure, SIMD mode, 1D or 2D systolic array structures, which demonstrates the flexibility provided by SmartCell through the resource reorganization and hierarchical on-chip networks. A novel two-step parallel FFT algorithm has also been proposed in this dissertation that distributes the transform task onto multi-core systems for parallel computing. The proposed algorithm achieves balanced workload and fixed data flow throughout different FFT stages. Analysis showed that the proposed algorithm is able to improve the scalability and greatly reduce the communication and configuration overhead. This algorithm has been successfully mapped onto the SmartCell-II system.

A prototype SmartCell system with 64 PEs was initially developed in standard cell ASICs with TSMC 0.13 μm technology. The chip consists of 1.6 million gates with an average power consumption of 1.6 mW/MHz for the applications under test. SmartCell achieves a peak performance of 45.7 GOPS/W. Furthermore, it is about 4.1x more energy efficient than the fine-grained FPGA and is about 6.4x less efficient than the fixed function ASICs. When compared with other CGRAs, SmartCell achieves 4x and 2x throughput gains and is about 8% and 69% more energy efficient than Montium and RaPiD CGRAs, respectively. These results showed that SmartCell is able to provide a promising solution in the stream processing domain to achieve the high performance and energy efficiency requirements.

Despite the performance advantages, some limitations were also observed from the experiences of the first SmartCell implementation. For example, eliminating of on-chip data memory makes it difficult to map large size application onto SmartCell. During dynamic configuration, unbalanced delays were observed for different PEs along the SPI chain. To address these issues, we developed SmartCell-II, the second generation of SmartCell,

with distributed data memory, segmented instruction formats and various configuration schemes for dynamic reconfiguration. Four data addressing modes were developed for flexible data memory accessing. New instruction loop logics were also designed to improve the efficiency of the instruction code utilization, which in turn reduces the instruction memory requirements.

A 4 by 4 SmartCell-II prototype system was implemented in standard cell ASICs with TSMC 90 nm technology. The proposed FFT algorithm with different sizes were mapped onto the prototype SmartCell-II device. The synthesis results showed that SmartCell-II dissipates about 3.1 mW/MHz power and can be operated at up to 295 MHz. It is about 5.0 mm^2 with roughly 2.0 million gates. Compared with other FFT implementations, SmartCell-II is about 14.9 and 2.7 times faster than the parallel FFTs implemented by NoC and MorphoSys, respectively. SmartCell-II is also about 3.6 and 28.9 times more energy efficient than FPGA and DSP based implementations.

8.2 Future Work

There are quite a few interesting research topics led by this work that are worthwhile for future investigations.

8.2.1 Design of a Complete SmartCell System

This dissertation presented the architecture design of SmartCell core module and evaluated its performance based on synthesis results. One important area of future work would be a complete system design and integration. For the SmartCell core processors, the synthesized front end netlist file can be imported into the back end design CAD tools, such as Cadence Encounter [7] or Synopsys Astro [8], for automatic logic optimization (both timing and power), floorplanning, placing, clock tree insertion, routing, chip I/O packaging, and etc. Several power management schemes can be applied during the back end design to further improve the power and energy efficiency. At system level, dynamic power man-

agement (DPM) can be developed to selectively turn on and off system components based on workload requirement, such as voltage scaling, multiple voltage supplies and software power management [25, 67]. On the other hand, several logic level techniques can be adopted to reduce power consumption, such as memory block structuring, pre-computing [43, 70]. The results from the back end design can be then sent to foundry company for taping out.

Besides the physical design and implementation of the core processor module, another important area of future work would be to integrate the SmartCell core with other controller and memory modules to build a complete system. Fig. 8.1(a) depicts one possible solution. An ARM or MIPS processor can be used to handle the control and configuration of SmartCell. The control processor can also be used to execute some sequential application tasks that are not suited for parallel computing. An external memory block will be integrated into the system to provide off-chip data storage. The memory can be connected to the SmartCell core through a Direct Memory Accessing (DMA) controller to build high-bandwidth memory interface. Given the nature of the targeted applications, a high speed I/O design is essential to provide high data communication bandwidth. Fig. 8.1(b) proposes an I/O structure design, which tries to shed light on the high speed I/O designs for stream processing. This I/O structure includes high speed low power PCI-E interface [52], two channel DDR2 memory interface and other on-chip configuration ports. Limited parallel I/Os will also be included to selectively route the critical internal signals to the chip I/O pads for real-time system monitoring.

8.2.2 Software Support for SmartCell

The development of the software programming environment is another interesting direction for our future work. In Section 5.3, Smart_C software compiler is proposed to assist the automation of application mapping onto the SmartCell system. Two phases are involved in the software design flow, with application and architecture analysis phase to

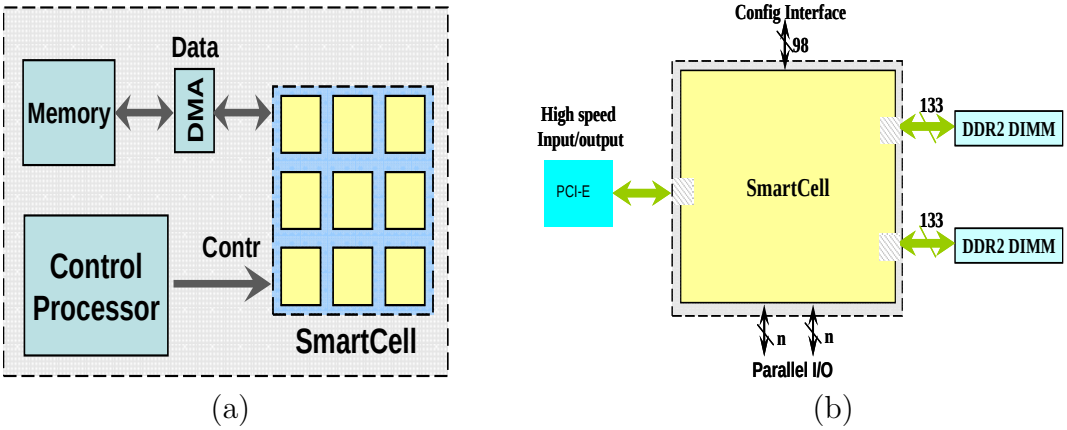


Figure 8.1: Propose of an integration structure for SmartCell system with high speed I/O designs for future research directions. (a) Integration of SmartCell core, microcontroller and data memory into the same system. (b) High speed I/O design for off-chip data transmission and system configurations.

parse the workload onto different hardware resources and a second application mapping phase to generate the configuration contexts that can be directly loaded into the SmartCell instruction memories. Due to the reconfigurable nature, hardware description library needs to be developed to describe different resource primitives, probably including the computing configuration, communication configuration, data memory configuration and program control configuration. A compiler algorithm needs to be developed that takes a hardware configuration file and a software construct file (preferably in C) as inputs and generates a run-time scheduling file to be mapped onto the predefined resource libraries. Benefitting from the regular tile structure and uniformed control logic, SmartCell can be configured to accommodate different system requirements of high performance or ultra low power consumption. The compiler needs to be robust enough to take advantage of this hardware flexibility. One solution would be that the compiler reads in a system constraint file, specifying the system requirements, available hardware resources and targeted frequency, based on which the configuration contexts are generated to satisfy these system requirements. Hardware virtualization also needs to be handled by the compiler to separate large computational tasks into smaller ones that can fit on the hardware resources and be processed individually. The optimization of task partitioning and scheduling needs

to be addressed to exploit both spatial and temporal parallelism offered by SmartCell. Other issues, such as loop breaking, redundancy optimization and task partitioning, also need to be addressed in the future Smart_C compiler design.

8.2.3 Scalability

The last but not least area of future work involves the investigation of the scalability of SmartCell. The 2D tiled architecture is able to provide good scalability. The distributed on-chip memory is also much more flexible to be scaled up or down to fit different storage requirements compared with shared memory scheme with centralized control structure. But some other issues are needed to be considered. As the system scales up, the clock distribution becomes more and more complex in synchronized architecture, which becomes an obstacle for performance improvement and energy efficiency. To address this issue, some asynchronous architectures [90, 49] has been proposed to separate the processing blocks such that each block is locally synchronized by an independent clock domain. This scheme eliminates the global clock distribution and has the potential to achieve better scalability and performance efficiency. This is an interesting direction for future SmartCell research.

Currently, the static routing is designed in SmartCell to deterministically route data packages from one component to another to build an efficient interconnection with small latency and area cost. But as the system scales up, the static routing may become more and more challenging for the compiler to generate the routing controls for each individual component. The dynamic routing may provide an alternative solution to dynamically generate the data path based on local traffic information. By this means, only the source and destination locations need to be specified to initialize a data transfer, which greatly releases the configuration stress and improves system flexibility. However, dynamic routing introduces non-negligible overhead in terms of circuitry area and communication delays that need to be fully studied in the future work.

Bibliography

[1] CPU power consumption: http://en.wikipedia.org/wiki/CPU_power_dissipation.

[2] http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4.

[3] http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5.

[4] http://www.altera.com/products/devices/stratix2/st2-index.jsp.

[5] NVIDIA Inc. http://www.nvidia.com/page/geforce_8800.html.

[6] AMD Inc. http://www.amd.com/us/fusion/Pages/index.aspx.

[7] http://www.cadence.com/products/di/soc_encounter/pages/default.aspx.

[8] http://solvnet.synopsys.com/dow_retrieve/C-2009.03/ni/psyn.html#Astro.

[9] "Arrix family product brief," Mathstar Inc. http://www.mathstar.com/.

[10] "ATI Stream developer forum," AMD Inc. http://www.amd.com/streamdevforum.

[11] "ISE Design Suite 10.1 guide," Xilinx Inc. http://www.xilinx.com/support/software_manuals.htm.

[12] "KC256 technical overview," Rapport Inc. http://www.rapportincorporated.com/.

[13] "Synopsys Inc. DsignWare Datasheet, version A2007.12DWDoc0803."

[14] "TMS320C62x DSPs C62x core benchmarks from texas instruments," Texas Instruments Inc. http://www.ti.com.

[15] "TMS320C67x floating point DSPs C67x core benchmarks," Texas Instruments Inc. http://www.ti.com.

[16] "AMD Athlon 64 X2 Dual-Core Processor Product Data Sheet," 2005.

[17] "NVIDIA Inc. NVIDIA CUDA programming guide 2.0," 2008.

[18] S. Agarwala, T. Anderson, A. Hill, M. D. Ales, R. Damodaran, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, A. Rajagopal, A. Chachad, M. Agarwala, J. Apostol, M. Krishnan, D. Bui, Q. An, N. S. Nagaraj, T. Wolf, and T. T. Elappuparackal, "A 600-MHz VLIW DSP," *IEEE Journal of Solid-State Circuits*, pp. 1532–1544, 2002.

[19] S. Agarwala and etc., "A 65nm c64x+ multi-core DSP platform for communication infrastructure," *In procedding of IEEE International Solid-State Circuits Conference*, 2007.

[20] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, "Evaluating the imagine stream architecture," *In procedding of International Symposium on Computer Architecture*, 2004.

[21] J. Bahn, J. Yang, and N. Bagherzadeh, "Parallel FFT Algorithms on Network-on-Chips," *In Proceedings of IEEE Fifth International Conference on Information Technology*, pp. 1087–1093, 2008.

[22] J. Balfour and W. Dally, "Design tradeoffs for tiled CMP on-chip networks," *In Proceedings of the 20th annual international conference on Supercomputing*, pp. 187–198, 2006.

[23] M. Bayoumi and N. Ling, "Specification and verification of systolic arrays," *World Scientific Publishing Singapore*, 1999.

[24] J. Becker and M. Vorbach, "Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC)," *In Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 107–112, 2003.

[25] L. Benini, A. Bogliolo, and G. D. micheli, "A survey of design techniques for system-level dynamic power management," *in Proceedings of IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 299–316, 2000.

[26] V. Bhaskaran and K. Konstantinides, "Image and Video Compression Standards algorithms and architecture - Second Edition," *Kluwer Academic Publishers*, 1999.

[27] T. Bjerregarrd and S. Mahadevan, "A survey fo research and practices of Network-on-Chip," *ACM Computing Surveys*, 2006.

[28] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architecture exploration of the ADRES coarse-grained reconfigurable array," *Springer Reconfigurable Computing: Architectures, Tools and Applications*, pp. 1–13, 2007.

[29] F. Bouwens, M. Berekovic, B. Sutter, and G. Gaydadjiev, "Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array," *Springer: Lecture Notes in Computer Science*, pp. 61–61, 2008.

[30] L. Cannon, "A cellular computer to implement the kalman filter algorithm," *Technical report, Ph.D. Thesis, Montana State University*, 1969.

[31] E. Caspi, M. Chu, R. Huang, J. Yeh, Y. Markovskiy, J. Wawrzynek, and A. DeHon, "Stream computations organized for reconfigurable execution (SCORE): Introduction and tutorial," *Web resourse: Extended version of Stream Computations Organized for Reconfigurable Execution (SCORE): Extended Abstract, in FPL 2000*, 2000.

[32] D. Chinnery and K. Keutzer, "Closing the gap between ASIC and custom: An ASIC perspective," *In Proceedings of 37th Design Automation Conference*, pp. 637–641, 2000.

[33] G. Chiu, "The odd-even trun model for adaptive routing," *IEEE Transactions on Parallel and Distributed Sysmtems*, pp. 729–738, 2000.

[34] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput*, pp. 297–301, 1965.

[35] D. Cronquist, C. fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," *In Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, pp. 23–40, 1999.

[36] W. J. Dally and A. Chang, "The role of custom designs in ASIC chips," *In Proceedings of 37th Design Automation Conference*, pp. 643–647, 2000.

[37] A. DeHon, "Reconfigurable Achitectures for General Purpose Computing," *MIT AI Lab Report no. AR 1586*, 1996.

[38] A. DeHon, Y. Markovsky, E. Caspi, M. Chu, R. Huang, S. Perissakis, L. Pozzi, J. Yeh, and J. Wawrzynek, "Stream computations organized for reconfigurable execution," *Elsevier Microprocessors and Microsystems*, vol. 30, no. 6, pp. 334–354, 2006.

[39] J. Draper, J. Sondeen, S. Mediratta, and I. Kim, "Implementation of a 32-bit RISC processor for the data-intensive architecture processing-in-memory chip," *In Proceedings of the IEEE Low Power Electronics and Design*, pp. 161–166, 2005.

[40] C. Fisher, K. Rennie, G. Xing, S. Berg, K. Bolding, J. Naegle, D. Parshall, D. Portnov, A. Sulejmanpasic, and C. Ebeling, "Emulator for exploring RaPiD configurable computing architectures," *In Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pp. 17–26, 2001.

[41] C. Glass and L. Ni, "The turn model for adaptive routing," *In proceedings of the 19th annual international symposium on Computer architecture*, pp. 278–287, 1992.

[42] I. Gopal, "Prevention of store-and-forward deadlock in computer networks," *IEEE Transaction on Communications*, pp. 1258–1264, 2007.

[43] S. Hanson, B. Zhai, D. Blaauw, and D. Sylvester, "Energy-Optimal Circuit Design," *in Proceedings of IEEE International Symposium on System-on-Chip*, pp. 1–4, 2007.

[44] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," *In Proceedings of IEEE Conference and Exhibition on Design, Automation and Test in Europe*, pp. 642–649, 2001.

[45] P. Heysters, G. Smit, and E. Molenkamp, "Energy-efficiency of the Montium reconfigurable tile processor," *In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 38–44, 2004.

[46] C. Hsieh and T. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 169–175, 1999.

[47] http://focus.ti.com/dsp/docs/dsphome.tsp?sectionId=46.

[48] Intel, "Paragon XP/S Product overiew."

[49] X. Jia and R. Vemuri, "Studying a GALS FPGA architecture using a parameterized automatic design flow," *in Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pp. 688–693, 2006.

[50] A. Kamalizad, C. Pan, and N. Bagherzadeh, "Fast parallel FFT on a reconfigurable computation platform," *In Proceedings of IEEE 15th Symposium on Computer Architecture and High Performance Computing*, pp. 254–259, 2003.

[51] S. M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits Analysis and Design, the Third Edition.* Published by McGraw-Hill, 2002.

[52] A. Kazmi, "Minimizing PCI Express Power Consumption," 2007, http://www.pcisig.com.

[53] S. W. Keckler, D. Burger, C. R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. S. Hrishikesh, N. Ranganathan, and P. Shivakumar, "A wire-delay scalable microprocessor architecture for high performance systems," *In Proceedings of IEEE International Solid-State Circuits Conference*, pp. 168–169, 2003.

[54] B. Khailany, W. Dally, U. Kapasi, P. Mattson, J. Namkoong, J. Owens, B. Towles, A. Chang, and S. Rixner, "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, 2001.

[55] B. Khailany, T. Williams, J. Lin, E. Long, M. Rygh, D. Tovey, and W. Dally, "A programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing," *IEEE Journal of Solid-State Circuits*, pp. 202–213, 2008.

[56] D. Kim, M. Kim, and G. Sobelman, "Parallel FFT computation with a CDMA-based network-on-chip," *In Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 1138–1141, 2005.

[57] Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," *In Proceedings of the IEEE Design, Automation and Test in Europe*, pp. 12–17, 2005.

[58] A. KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja, "Adapting the SPEC 2000 benchmark suite for simulation-based computer architecture research," *Workload characterization of emerging computer applications*, pp. 83–100, 2001.

[59] C. Kozyrakis, "Scalable vector media-processors for embedded systems," *PhD thesis, Unitersity of California at Berkeley*, 2002.

[60] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integerated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[61] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and challenges," *in Foundations and Trends in Electronic Design Automation*, pp. 135–253, 2008.

[62] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1712–1724, 2005.

[63] C. Liang and X. Huang, "A Fault-Tolerant Fully Adaptive Routing Algorithm for Collaborative Computing in Mesh Networks," *in Proceedings of SPIE*, 2007.

[64] ——, "SmartCell: A power-efficient reconfigurable architecture for data streaming applications," *In Proceedings of IEEE Workshop on Signal Processing Systems (SiPS'08)*, pp. 257–262, 2008.

[65] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An Architecture and Compiler for Scalable On-Chip Communication," *in IEEE Transactions on VLSI Systems*, pp. 711–726, 2004.

[66] Y. Lin and C. Lee, "Design of an FFT/IFFT Processor for MIMO OFDM Systems," *In Proceedings of IEEE Transactions on Circuits and Systems I*, pp. 807–815, 2007.

[67] J. Lorch and A. Smith, "Software strategies for portable computer energy management," *in Proceedings of IEEE Personal Commun.*, pp. 60–73, 1998.

[68] T. Marshall, L. Stansfield, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," *In Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field Programmable Gate Arrays*, pp. 135–143, 1999.

[69] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 157–166, 1996.

[70] J. Monteiro, "Techniques for power management at the logic level," *in Proceedings of IEEE International Conference on Electronics, Circuits and Systems*, pp. 181–184, 1998.

[71] S. Naffziger, T. Grutkowski, and B. Stackhouse, "The implementation of a 2-core multithreaded Itanium family processor," *In Proceedings of IEEE International Solid-State Circuits Conference*, pp. 182–183, 2005.

[72] A. Peleg, S. Wilkie, and U. Weiser, "Intel mmx for multimedia pcs," *Communications of the ACM*, pp. 24–38, 1997.

[73] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa, "The design and implementation of a first-generation CELL processor," *in Proceedings of IEEE International Solid-State Circuits Conference*, pp. 184–185, 2005.

[74] S. Rixner, "Stream Processor Architecture," *Kluwer Academic Publishers*.

[75] R. Russell, "The Cray-1 computer system," *Communications of the ACM*, 1978.

[76] K. Sankaralingam, R. Nagarajan, H. Liu, J. Huh, C. Kim, D. Burger, S. Keckler, and C. Moore, "Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture," *In International Symposium on Computer Architecture*, pp. 422–433, 2003.

[77] M. Saravana, S. Govindan, D. Burger, and S. Keckler, "TRIPS: A distributed explicit data graph execution microprocessor," *HotChips*, 2007.

[78] H. Schmit, "Incremental reconfiguration for pipelined applications," *In Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 47–55, 1997.

[79] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," *In Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 63–66, 2002.

[80] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[81] L. Smit, G. Rauwerda, A. Molderink, P. Wolkotte, and G. Smit, "Implementation of a 2-D 8x8 IDCT on the reconfigurable Montium core," *In Proceedings of the 2007*

International Conference on Field Programmable Logic and Applications (FPL'07), pp. 562–566, 2007.

[82] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math. 13*, pp. 354–356, 1969.

[83] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The RAW microprocessor: a computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[84] M. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim, "Evaluation of the RAW microprocessor: an exposed-wire-delay architecture for ilp and streams," *In Proceedings of the IEEE 31st Annual International Symposium on Computer Architecture*, pp. 2–13, 2004.

[85] T. Tuan and B. Lai, "Leakage power analysis of a 90 nm FPGA," *in Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 57–60, 2003.

[86] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Lyer, A. Singh, T. Jacb, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS," *in Proceedings of IEEE International Solid-State Circuits Conference*, pp. 98–99, 2007.

[87] J. von Neumann, "First draft of a report on the EDVAC," *Reprinted (in part) in Randell, Brian. 1982. Origins of Digital Computers: Selected Papers, Springer-Verlag, Berlin Heidelberg, June 1945.*, 1945.

[88] R. Wilson, R. French, C. Wilson, S. Amarasinghe, J. Anderson, S. Tjing, S. Liao, C. W. Tseng, M. Hall, M. Lam, and J. Hennessy, "SUIF: An infrastructure for re-

search on parallelizing and optimizing compilers," *ACM SIGPLAN Notices*, pp. 31–37, 1994.

[89] www.xilinx.com/support/mysupport.htm/Virtex II

[90] Z. Yu, "High performance and energy efficient multi-core systems for dsp applications," *PhD thesis, Unitersity of California at Davis*, 2007.

[91] Z. Yu, M. J. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas, "AsAP: an Asynchronous Array of Simple Processors," *IEEE Journal of Solid-State Circuits*, pp. 695–705, 2008.

[92] J. Zawodny and P. Kogge, "Cache-In-Memory," *Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp. 3–11, 2001.

[93] Q. Zhang, A. Kokkeler, and G. Smit, "An Efficient FFT for OFDM Based Cognitive Radio on a Reconfigurable Architecture," *In Proceedings of IEEE International Conference on Communications*, pp. 6522–6526, 2007.