ARCHITECTURE AND ANALYSIS FOR NEXT GENERATION MOBILE SIGNAL PROCESSING

by

Mark Woh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2011

Doctoral Committee:
Professor Trevor N. Mudge, Chair
Professor David Blaauw
Professor William Martin
Associate Professor Scott A. Mahlke
Professor Chaitali Chakrabarti

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance and support of many people. First, and foremost, I would like to give thanks to my advisor, Professor Trevor Mudge. Throughout my Ph.D., he has given me support and encouragement through thick and thin. His guidance has helped me overcome many hurdles that i have encountered. Without him, this dissertation would not be possible. I would also like to thank Professors Scott Mahlke and Chaitali Chakrabarti. Without their tireless effort and help, my research would not be possible. Both have put in countless hours to develop ideas and help complete my research.

I would like to thank ARM Ltd. and especially Krisztian Flautner for not only funding my research but also providing me with the valuable insight that has gotten me this far. Without their trust and confidence in my work, I would not have gotten this far.

I would also like to thank my thesis committee members, Professors David Blaauw and William Martin. They have given their time and provided me with valuable insights which have helped me improve my thesis.

The research presented in this dissertation also would not be possible without the

support and encouragement of many of my colleagues. In particular, Dr. Yuan Lin, Dr. Sangwon Seo and Dr. Hyunseok Lee have assisted me in virtually every aspect of my graduate school life: running experiments, refining ideas, writing papers, and even picking me up when i was stranded in Auburn Hills.

I would also like to thank all of my colleagues in EECS. Many of whom I've discussed ideas with for countless hours. Their support not only as a colleague but also as a friend were crucial to my graduate school experience. I'd like to personally thank Amir Hormati, Shuguang Feng, Ronald Dreslinski, Geoffrey Blake, Ganesh Dasika, Shantanu Gupta, Amin Ansari, Mojtaba Mehrara, Steven Pelley, David Meisner, Hyunchul Park, Yongjun Park, Kevin Fan, and Hyounkyu Cho.

Finally, I would like to thank my family and long time friends for their support, encouragement, and advice. Without them I would not have enough strength to complete this long journey.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES viii

LIST OF TABLES xiii

ABSTRACT xiv

CHAPTERS

1 Introduction 1
- 1.1 Organization 6

2 Overview of Processor Designs for Mobile Software Defined Radio . . 7
- 2.1 SDR Overview 10
- 2.2 Workload Profiling and Characterization 11
  - 2.2.1 The W-CDMA Physical Layer Processing 11
  - 2.2.2 W-CDMA Workload Profiling 15
- 2.3 Architecture Design Trade-offs 18
  - 2.3.1 8-bit and 16-bit Fixed-point Operations 19
  - 2.3.2 Vector-based Arithmetic Computations 19
  - 2.3.3 Control Plane versus Data Plane 20
  - 2.3.4 Scratchpad Memory versus Cache 20
  - 2.3.5 Algorithm-Specific ASIC Accelerators 21
- 2.4 Baseband Processor Architectures 22
- 2.5 SODA Processor 23
- 2.6 Other SIMD-based Architectures 27
  - 2.6.1 EVP 27
  - 2.6.2 DXP 28
  - 2.6.3 TigerSHARC 28
  - 2.6.4 MuSIC-1 29
  - 2.6.5 Sandblaster 30
  - 2.6.6 SIMT 30
- 2.7 Reconfigurable Architectures 31

2.7.1 ADRES . . . 31
2.7.2 Montium . . . 32
2.7.3 Adapt2400 ACM . . . 32
2.7.4 XiRisc . . . 33
2.8 Conclusion . . . 34

3 From SODA to Scotch: The Evolution of a Wireless Baseband Processor 35
3.1 Architecture Overview . . . 38
3.1.1 SODA Architectural Overview . . . 39
3.1.2 Ardbeg Architecture . . . 41
3.2 Architectural Evolution: SODA to Ardbeg . . . 44
3.2.1 Optimized Wide SIMD Design . . . 44
3.2.2 LIW SIMD Execution . . . 49
3.2.3 Application Specific Hardware Acceleration . . . 51
3.3 Results and Analysis . . . 57
3.3.1 Wireless Protocol Results . . . 58
3.3.2 Wireless Algorithm Analysis . . . 60
3.4 Conclusion . . . 64

4 The Next Generation Challenge for Software Defined Radio . . . 66
4.1 4G Physical Layer . . . 68
4.1.1 OFDMA . . . 69
4.1.2 MIMO . . . 70
4.1.3 Channel Encoder/Decoder . . . 73
4.2 Computational Analysis . . . 75
4.2.1 Baseline Architecture . . . 75
4.2.2 Workload Profile . . . 77
4.2.3 Computational Patterns . . . 78
4.3 Architectural Implications . . . 80
4.3.1 Multi-Processor . . . 81
4.3.2 Wider SIMD . . . 81
4.3.3 Special Purpose Functional Units . . . 82
4.3.4 Memory System . . . 83
4.4 Summary . . . 84

5 Algorithmic Analysis and Architectural Implications of the Next Generation Software Defined Radio . . . 85
5.1 SDR Processors . . . 87
5.2 4G Wireless Kernels . . . 89
5.2.1 Fast Fourier Transforms (FFT and IFFT) . . . 90
5.2.2 Space Time Block Codes (STBC) . . . 93
5.2.3 Vertical Bell Laboratories Layered Space-time . . . 94
5.2.4 Low Density Parity-Check Codes (LDPC) . . . 97
5.3 Methodology . . . 99
5.4 SIMD Width Analysis . . . 99

5.5 Register Access Analysis . . . 104
5.6 Summary . . . 107

6 Multimedia Algorithms . . . 109
6.1 H.264 Algorithm Analysis . . . 110
6.1.1 Intra Prediction . . . 111
6.1.2 Deblocking Filter . . . 112
6.1.3 Motion Compensation . . . 113
6.1.4 Motion Estimation . . . 114
6.2 H.264 Architectural Implications . . . 116
6.2.1 Multiple SIMD Widths . . . 116
6.2.2 Diagonal Memory Organization . . . 117
6.2.3 Bypass and Temporary Buffer Support . . . 119
6.2.4 Fused Operation . . . 119
6.2.5 Programmable Crossbar . . . 120

7 Low Power SIMD Architectures using a Scalable SRAM Based Swizzle Interconnect . . . 122
7.1 Related Works . . . 125
7.1.1 Crossbars . . . 126
7.1.2 Benes Network . . . 127
7.1.3 Banyan Network . . . 128
7.2 XRAM Fundamentals . . . 129
7.2.1 XRAM configurations . . . 133
7.2.2 Summary of XRAM . . . 135
7.3 Case Studies . . . 137
7.3.1 FFT Engine . . . 138
7.3.2 Color-space Conversion Hardware . . . 140
7.4 Usage Models for XRAM . . . 143
7.4.1 Multi-core Interconnect . . . 143
7.4.2 Memory Interconnect . . . 144
7.4.3 Reliability . . . 144
7.5 Conclusion . . . 146

8 AnySP: Anytime Anywhere Anyway Signal Processing . . . 148
8.1 Mobile Signal Processing . . . 149
8.1.1 Overview of Benchmarks . . . 149
8.1.2 Algorithm Analysis . . . 152
8.2 AnySP Architecture . . . 160
8.2.1 AnySP PE Design . . . 160
8.3 Results and Analysis . . . 170
8.3.1 Methodology . . . 170
8.3.2 Algorithm-Level Results . . . 171
8.3.3 System-Level Results . . . 176
8.4 Related Work . . . 177

8.5 Summary 179

9 Conclusion 181
9.1 Summary 181
9.2 Future Work 182

BIBLIOGRAPHY 185

LIST OF FIGURES

Figure

1.1 Performance verses power requirements for various mobile computing applications. 2
2.1 Categories of Wireless Networks 8
2.2 Throughput and power requirements of typical 3G wireless protocols. The results are calculated for 16-bit fixed point operations.) 9
2.3 Architecture of a 3G cellular phone.) 11
2.4 Physical layer operation of W-CDMA wireless protocol. Each block includes the algorithm's name, vector or scalar computation, vector width, and the data precision. The algorithms are grouped into four categories, shown in shaded boxes: filtering, modulation, channel estimation, and error correction.) 12
2.5 Workload analysis result of W-CDMA physical layer processing. "Vector comp" indicates whether the algorithm contains vector-based arithmetic operations. "Vector width" lists the native computation vector width. "Bit width" lists the data precision width. "Comp Mcycle/sec" lists the peak workload of running the algorithm on a general purpose processor.) 15
2.6 W-CDMA Instruction Type Breakdown.) 16
2.7 SODA Architecture. The system consists of 4 data processing elements (PEs), 1 control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a 32-lane 16-bit SIMD pipeline, a 16-bit scalar pipeline, two local scratchpad memories, an AGU for calculating memory addresses, and a DMA unit for inter-processor data transfer. 23
2.8 8-wide SIMD Shuffle Network(SSN) 24
3.1 Plots of normalized energy, delay, and energy-delay product versus area plots for different Ardbeg SIMD width configurations running 3G wireless algorithms. The results are normalized to the 8-wide SIMD design. 38

3.2 SODA and Ardbeg architectural diagrams, and a summary of the key architectural features of the two designs. 40
3.3 SIMD shuffle network for the SODA PE and the Ardbeg PE. For illustration clarity, these examples show 16-wide shuffle networks. The SODA PE has a 32-wide 16-bit 1-stage iterative shuffle network, and the Ardbeg PE has a 128-lane 8-bit 7-stage Banyan shuffle network. . 43
3.4 Normalized energy and energy-delay product for key SDR algorithms running on Ardbeg for different shuffle network topologies. 44
3.5 Ardbeg VLIW support. The results are shown for software pipelined Ardbeg assembly code. Ardbeg has 7 different function units, as listed in sub-figure a. These seven function units share 3 SIMD register file read and 2 write ports. At most two SIMD operations can be issued per cycle, and not all combinations of SIMD operations are supported. Different LIW configurations are evaluated in terms of delay and energy-delay product, as shown in sub-figures c and d. . . 48
3.6 Ardbeg's pair-wise butterfly SIMD operation implemented using a fused permute and ALU operation. The figure shows pairs of a 2-element butterfly operation. Ardbeg supports pairs of 1-,2-,4-,8-,and 16-element butterfly of 8- and 16-bits. This butterfly operation uses the inverse perfect shuffle pattern because the input to each SIMD ALU lane must come from the same SIMD lane. 54
3.7 SSN shuffling patterns used for matrix transpose. 55
3.8 DSP algorithms that are used in W-CDMA, 802.11a and DVB, DVB-H wireless protocols. 57
3.9 Throughput and power achieved for SODA and Ardbeg for W-CDMA, 802.11a and DVB-T/H. ASIC 802.11a, Pentium M, Sandblaster, and ADI TigerSharc results are also included for comparison purposes. Results are shown for processors implemented in 90nm, unless stated otherwise. 59
3.10 Ardbeg speedup over SODA for the key DSP algorithms used in our wireless protocol benchmarks. The speedup is broken down into the different architectural optimizations. These include optimized SIMD ALU, wider 1-cycle SIMD shuffle network, reduced SIMD memory latencies through LIW execution, and compiler optimizations with software pipelining. 61
4.1 The physical layer for a 4G terminal 68
4.2 The data movement of an 8 point FFT and the computations in a 2 point FFT 69
4.3 Transmission code matrix and computation patterns of the Alamouti 2x2 STBC 71
4.4 LDPC graphical representation and decoding operations 74
4.5 SODA Architecture for SDR 76

4.6 Technology scaling from 180nm to 22nm with respect to Frequency, Power, Vdd on SODA for 4G 79
5.1 SODA Architecture for SDR. The system consists of 4 data processing elements (PEs), 1 control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a 32–wide 16–bit SIMD pipeline, a 16–bit scalar pipeline, two local scratchpad memories, an Address–Generation–Unit(AGU) for calculating memory addresses, and a Direct–Memory–Access (DMA) unit for inter-processor data transfer. . . . 88
5.2 Block Diagram Overview of a 4G System. 89
5.3 For the 8-point FFT, the butterfly pattern is the cross between two different elements in the vector. The major inner loop of FFT performs the operation of the butterfly across a SIMD. The nodes represent SODA instructions and the edges represent data dependencies. The instruction pattern represents the actual butterfly operation that is performed in each lane. 90
5.4 STBC general operation of Alamouti scheme. 91
5.5 The inner loop of STBC shown calculates the values x[1] and x[2] of Alamouti Scheme. The common instruction pattern as shown is the majority of the inner loop code. 92
5.6 General decoding process of V-BLAST 93
5.7 These two instruction patterns are reused many times within V-BLAST. They are common in many of the matrix operations. 95
5.8 LDPC graphical representation and decoding operations 96
5.9 The inner loop operations of LDPC. These three operations represent the majority of the LDPC workload. They correspond closely to steps 2-4 of the LDPC decoding operation. 97
5.10 Instruction breakdown of each kernel with respect to SIMD width. . . 101
5.11 Normalized speedup of each kernel with respect to SIMD width. . . . 102
5.12 Total energy consumption of each kernel with respect to SIMD width. 103
5.13 Power breakdown for the kernels on SODA 104
5.14 Modified Register File. Data can either be written to the main register file, the small register or bypassed completely from both allowing data to be forwarded directly to the next functional unit. 106
5.15 Total energy consumption of each kernel with reduced register file access. SODA is the baseline processor where each instruction always writes back to the register file then is read when needed. RRA-SODA is the modified SODA architecture that can write the values to a smaller register file. 106
6.1 Block Diagram of H264 110
6.2 Mapping a 16x16 luma macroblock intra-prediction process on the proposed architecture. Example of the Diagonal Down Right intra-prediction for a 4x4 sub block (grey block) is presented with fused operations. 111

6.3 Mapping a deblocking filter process when BS (Block Strength)=4. . . 113
6.4 Example of interpolation of motion compensation (half-pel). 114
6.5 Mapping a motion estimation process for a 4x4 block on the proposed architecture; The search area is 8x8. 115
6.6 Diagonal memory organization and shuffle network, which allows the horizontal and vertical memory access without conflict. The 64x64 shuffle network realigns 64 16-bit data. 118
6.7 Subgraphs for the inner loops for two H.264 kernels; The bypass path is not shown for simplicity. 118
6.8 Permutation Patterns for H.264 Intra-prediction Modes 120
7.1 Permutations are 1-to-1 mappings of input to output ports. Swizzles are the generic form of duplication and permutation combined where the output does not need to contain all the inputs. Broadcast is a special form of swizzle where one of the values is broadcasted to all the outputs. Multicasting is another special form of swizzle where multiple input ports are duplicated in a regular pattern to the outputs. 124
7.2 Different Commonly Used Network Toplogies 125
7.3 Comparison between Crossbar, Benes, Banyan and XRAM networks with respect to Scalability, Speed, Energy, and Flexibility where N is the number of ports in an NxN network 129
7.4 XRAM is a low power high performance matrix style crossbar that reuses output buses for control programming and stores multiple swizzle configurations at the cross points using SRAM cells. The XRAM controller sends the control line to pick which configuration SRAM cell to use. The encoder and decoder are the transition encoding/decoding logic used to minimize the switching power when the input and output bits do not change values. 131
7.5 Different Placements of XRAM within the Processor Architecture. For the memory interconnect benefits occur when M $\geq$ 2N. 134
7.6 Comparison between the XRAM, Benes and MUX crossbar network with respect to Area, Delay and Energy per bit (total bits = number ports * 16-bit). 136
7.7 FFT Engine 138
7.8 FFT results 139
7.9 Examples of different YUV modes. In each YUV mode the Y plane is always written out to memory then the U plane followed by the Y plane. In YUV 4:2:0 semi-planar, the U and V data are interleaved together. The typical size of the Y plane is 16x16 pixels. 140
7.10 Color Space Conversion Hardware 142
8.1 Select subgraphs for the inner loops of MSP algorithms. 153

8.2 Register file access for the inner loops of MSP algorithms. Bypass write and read are the RF read and write accesses that do *not* need to use the main RF. Register accesses are the read and write accesses that *do* need to use the main RF. 155
8.3 Instruction pair frequencies for different MSP algorithms. 157
8.4 A set of swizzle operations that are required for a subset of MSP algorithms. 159
8.5 AnySP PE. It consists of SIMD and scalar datapaths. The SIMD datapath consists of 8-groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32 and 64. Each of the 8-wide SIMD units are composed of groups of Flexible Functional Units (FFUs). The local memory consists of 16 memory banks; each bank is an 8-wide SIMD containing 256 16-bit entries, totalling 32KB of storage. 161
8.6 Flexible Functional Unit 164
8.7 The swizzle operation in computer graphics. The rows of the selection matrix correspond to the input vector and the columns correspond to the output vector selected. Each row can only have a single 1, but columns can have multiple 1s which allow for selective multi-casting of the data. 166
8.8 Comparison of power between the SRAM-based swizzle network and MUX-based crossbar. The values are normalized to the MUX-based crossbar for different crossbar widths with 16-bit data inputs. These results correspond to a switching activity of 0.50. 167
8.9 Demonstration of the output adder tree for matrix multiplication . . 169
8.10 AnySP speedup over SODA for the key algorithms used in 4G and H.264 benchmarks. The speedup is broken down into the different architectural enhancements - wider SIMD width, single-cycle SRAM-based crossbar, fused-operations and buffer support 171
8.11 Normalized Energy-Delay product for each kernel algorithm 171
8.12 Mapping a luma 16x16 MB intra-prediction process on AnySP; Example of the Diagonal Down Right intra prediction for a 4x4 sub block (grey block) is presented with each cycle's operations listed. 173
8.13 PE Area and Power Summary for AnySP running 100Mbps high mobility 4G wireless and H.264 4CIF video at 30fps 174
8.14 Performance verses power requirements for various mobile computing applications. 176

LIST OF TABLES

Table

4.1 Cycle Count of Major 4G Kernels on SODA 77
4.2 Computational Pattern of 4G algorithms 78
5.1 Data level parallelism analysis for major 4G kernels. 99
5.2 Percent of total register file access that are produced and consumed within a subgraph. These register file accesses can use the small register file 104
6.1 Shuffle patterns for six intra prediction modes for 4x4 luma 112
6.2 Kernel operations, SIMD workload, required SIMD width, and the amount of thread level parallelism (TLP) for H.264 encoder/decoder algorithms 116
6.3 Instruction pair frequency for H.264 kernel algorithms 119
8.1 Data level parallelism analysis for MSP algorithms. Overhead workload is the amount of instructions needed to aid the SIMD operations like data shuffle and SIMD load/store. 150

ABSTRACT

ARCHITECTURE AND ANALYSIS FOR NEXT GENERATION MOBILE SIGNAL PROCESSING

by

Mark Woh

Chair: Trevor N. Mudge

Mobile devices have proliferated at a spectacular rate, with more than 3.3 billion active cell phones in the world. With sales totaling hundreds of billions every year, the mobile phone has arguably become the dominant computing platform, replacing the personal computer. Soon, improvements to todays smart phones, such as high-bandwidth internet access, high-definition video processing, and human-centric interfaces that integrate voice recognition and video-conferencing will be commonplace. Cost effective and power efficient support for these applications will be required.

Looking forward to the next generation of mobile computing, computation requirements will increase by one to three orders of magnitude due to higher data

rates, increased complexity algorithms, and greater computation diversity but the power requirements will be just as stringent to ensure reasonable battery lifetimes. The design of the next generation of mobile platforms must address three critical challenges: efficiency, programmability, and adaptivity. The computational efficiency of existing solutions is inadequate and straightforward scaling by increasing the number of cores or the amount of data-level parallelism will not suffice. Programmability provides the opportunity for a single platform to support multiple applications and even multiple standards within each application domain. Programmability also provides: faster time to market as hardware and software development can proceed in parallel; the ability to fix bugs and add features after manufacturing; and, higher chip volumes as a single platform can support a family of mobile devices. Lastly, hardware adaptivity is necessary to maintain efficiency as the computational characteristics of the applications change. Current solutions are tailored specifically for wireless signal processing algorithms, but lose their efficiency when other application domains like high definition video are processed.

This thesis addresses these challenges by presenting analysis of next generation mobile signal processing applications and proposing an advanced signal processing architecture to deal with the stringent requirements. An application-centric design approach is taken to design our architecture. First, a next generation wireless protocol and high definition video is analyzed and algorithmic characterizations discussed. From these characterizations, key architectural implications are presented, which form the basis for the advanced signal processor architecture, *AnySP*.

CHAPTER 1

Introduction

In the coming years, the deployment of untethered computers will continue to increase rapidly. The world has more than 3 billion active cell phones [3], each a sophisticated multiprocessor built around application specific integrated circuits (ASICs). With mobile sales and usage totaling hundreds of billions of dollars every year, the cell phone has arguably become the dominant computing platform, a candidate for replacing the personal computer. We expect to see both the types and numbers of mobile digital devices to increase in the near future. New devices will improve on the mobile phone by incorporating advanced functionality, such as faster internet access, high quality video and human-centric interfaces that integrate voice recognition and video-conferencing. We can already see elements of advanced functionality in mobile devices from the major manufacturers, and with fourth generation (4G) wireless broadband and mobile high definition video on the horizon, cost effective and power efficient support for these applications will be required in the very near future.

The next generation of mobile computing will have higher data rates, increased

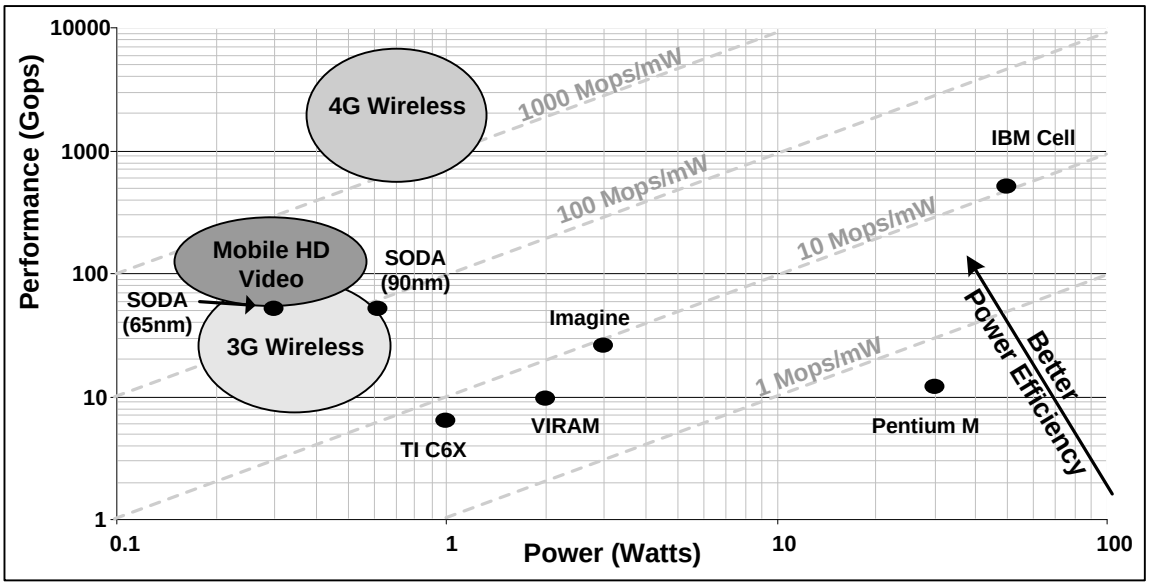


Figure 1.1: Performance verses power requirements for various mobile computing applications.

complexity algorithms, and greater computation diversity but the power requirements will be just as stringent. Mobile devices perform signal processing as one of their primary computational activities due to their heavy usage of wireless communication and their rendering of audio and video signals. Fourth generation wireless technology (4G) has been proposed to increase the bandwidth to maximum data rates of 100 Mbps for high mobility and 1 Gbps for low mobility. This translates to an increase in the computational requirements of 10-1000x over previous third generation wireless technologies (3G) with a power envelope that can only increase by 2-5x [90]. Other signal processing tasks, such as high-definition video, are also 10-100x more compute intensive than current mobile video. Scaling of existing approaches will not suffice, instead, the inherent computational efficiency, programmability, and adaptability of the hardware must change.

Figure 1.1 presents the demands of the 3G and 4G protocols in terms of the peak processing throughput and power budget. Conventional processors cannot meet the power-throughput requirements of these protocols. 3G protocols, such as W-CDMA, require approximately 100 Mops/mW. Desktop processors, such as the Pentium M, operate below 1 Mop/mW, while digital signal processors, such as the TI C6x, operate around 10 Mops/mW. High performance systems, like the IBM Cell [69], can provide excellent throughput, but its power consumption makes it infeasible for mobile devices [69]. Research solutions, such as VIRAM [43] and Imagine [4], can achieve the performance requirements for 3G, but exceed the power budgets of mobile terminals. SODA improved upon these solutions and was able to meet both the power and throughput requirements for 3G wireless [54]. Companies such as Phillips [88], Infineon [73], ARM [91], and Sandbridge [27] have also proposed domain-specific systems that meet the requirements for 3G wireless.

Current solutions for high performance mobile devices has been to wrap multiple digital signal processing (DSP) cores around ASICs and fixed function accelerators. As new applications are developed, these mobile platforms are redesigned with additional ASICs. As more applications and features are added to the phone, this becomes increasingly expensive for the consumer and manufacture in terms of development and material costs. Furthermore, current designs which are inflexible become obsolete. An example of a new application that many companies invested resources in is WiMAX. In order to stay ahead of the market, many companies like Intel and Motorola invested large amounts of resources into the development of a WiMAX platform. As

technology improved, the WiMAX standard became replaced with a new 4G wireless standard (LTE) even before WiMAX was deployed. Because the building blocks for these systems were designed specifically for WiMAX, these companies were left with inflexible designs that may not be able to adapt to future technologies. To solve these problems, many companies are looking for ways to design programmable mobile platforms that can support multiple standards and applications. These platforms will have to meet the stringent demands which each application requires.

Mobile computing platforms are not limited to performing only wireless signal processing. High-definition video is also an important application that these platforms will need to support. Figure 1.1 shows that the performance requirements of video exceed that of 3G wireless, but are not as high as 4G wireless. Thus the power per Gops requirement is further constrained. Moreover, the data access complexity in video is much higher than wireless. Wireless signal processing algorithms typically operate on single dimension vectors, whereas video algorithms operate on two or three dimensional blocks of data. Thus, video applications push designs to have more flexible higher bandwidth memory systems. High definition video is just one example of a growing class of applications with diverse computing and memory requirements that will have to be supported by the next generation of mobile devices.

The design of the next generation of mobile platforms must address three critical issues: *efficiency*, *programmability*, and *adaptivity*. The existing computational *efficiency* of 3G solutions is inadequate and must be increased by at least an order of magnitude for 4G. As a result, straightforward scaling of 3G solutions by

increasing the number of cores or the amount of data-level parallelism is not enough. *Programmability* provides the opportunity for a single platform to support multiple applications and even multiple standards within each application domain. It also provides faster time to market and higher chip volumes, thereby reducing manufacturing cost. Lastly, hardware *adaptivity* is necessary to maintain efficiency as the core computational characteristics of the applications change. 3G solutions rely heavily on the widespread amounts of vector parallelism in wireless signal processing algorithms, but lose most of their efficiency when vector parallelism is unavailable or constrained as in other application domains like high-definition video.

This thesis presents a set of design proposals for realizing an advanced signal processing architecture that targets the next generation mobile computing. In order to design a solution that can meet the lofty requirements, this thesis takes an application-centric approach – evaluate and optimize all aspects of the design based on the characteristics of the applications. We must first understand the workings of the different applications and their algorithms. With the insights from the applications, a mobile signal processing architecture is designed and optimized for these applications. This thesis makes the following contributions:

- Analysis and design of a commercial software defined radio architecture, Ardbeg, for current generation wireless protocols.
- Algorithmic characterization of two mobile signal processing applications: a fourth generation wireless protocol and high definition mobile video.
- Architectural insights derived from the characterizations of mobile signal processing applications.
- An advanced signal processing architecture, AnySP, that targets multiple applications domains, specifically 4G wireless communication and high-definition video decoding within the power budget of a mobile device.

1.1 Organization

The remainder of this dissertation is organized as follows. Chapter 2 introduces software defined radio and discusses previous architectures that have been proposed to solve the current generation signal processing challenge. Chapter 3 provides details about the architectural design of the Ardbeg processor which targets current generation wireless protocols. Chapter 4 provides analysis on the software characteristics of next generation wireless protocols. Chapter 5 uses these characterizations of the next generation wireless protocols and proposes key architectural insights to help guide us in development of a low power architecture. Chapter 6 provides analysis of a high definition mobile video application, H.264. Chapter 7, a low power scalable SRAM based swizzle interconnect is analyzed. And finally, in Chapter 8, an advanced signal processing architecture, *AnySP*, is proposed that targets the next generation mobile computing.

CHAPTER 2

Overview of Processor Designs for Mobile Software Defined Radio

Wireless communication has become one of the dominating applications in today's world. Mobile communication devices are the largest consumer electronic group in terms of volume. In 2007, there was an estimated 3.3 billion mobile telephone subscriptions. This number is roughly half of the world's population. Applications like web browsing, video streaming, email and video conferencing have all become key applications for mobile devices. As technology becomes more advanced, users will require more functionality from their mobile devices and more bandwidth to support them. Furthermore, in recent years, we have seen the emergence of an increasing number of wireless protocols that are applicable to different types of networks. Figure 2.1 lists some of these wireless protocols and their application domains, ranging from the home and office WiFi network to the city-wide cellular networks. With the proliferation of wireless mobile communications, the next generation mobile devices

will enable users to connect to information ubiquitously from every corner of the world.

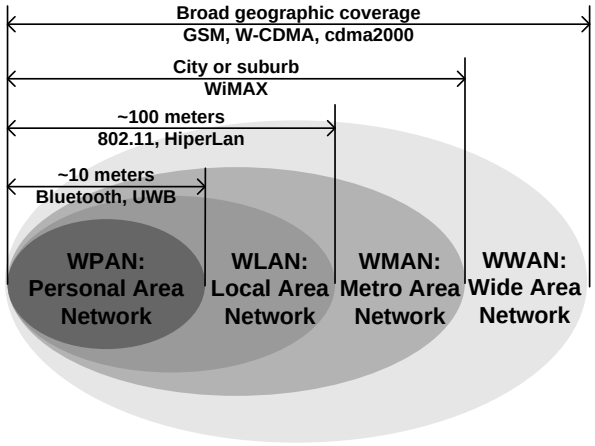


Figure 2.1: Categories of Wireless Networks

One of the key challenges in realizing ubiquitous communication is the seamless integration and utilization of the multiple existing and future wireless communication networks. In many current wireless communication solutions, the physical layer of the protocols are implemented with non-programmable ASIC processors. This means that the communication device would consist of multiple processors, one for each wireless protocol. Such a solution is not scalable and is clearly infeasible. Software Defined Radio (SDR) promises to deliver a cost effective and flexible solution by implementing a wide variety of wireless protocols in software, and running them on the same hardware platform. A software solution offers many potential advantages, including but not limited to the following:

- A programmable SDR processor would allow multimode operation, running different protocols depending on the available wireless network, GSM in Europe, CDMA in the USA and some parts of Asia, and 802.11 in coffee shops. This is possible with less hardware than custom implementations.

- A protocol implementation's time to market would be shorter because it would reuse the hardware. The hardware integration and software development tasks would progress in parallel.

- Prototyping and bug fixes would be possible for next-generation protocols on existing silicon through software changes. The use of a programmable solution would support the continuing evolution of specifications; after the chipset's manufacture, developers could deploy algorithmic improvements by changing the software without redesign.

- Chip volumes would be higher because the same chip would support multiple protocols without requiring hardware changes.

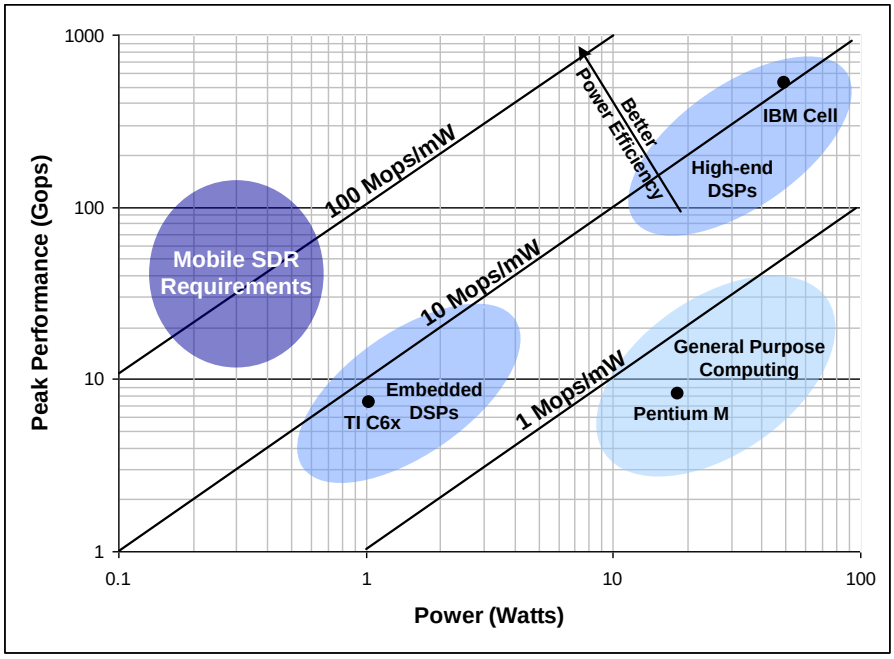


Figure 2.2: Throughput and power requirements of typical 3G wireless protocols. The results are calculated for 16-bit fixed point operations.)

Designing a SDR processor for mobile communication devices must address two keys challenges — meeting the computational requirements of wireless protocols while operating under the power budget of a mobile device. The operation throughput requirements of current third-generation (3G) wireless protocols are already an order of magnitude higher than the capabilities of modern DSP processors. This gap is likely to grow in the future. Figure 2.2 shows the computation and power demands of a typical 3G wireless protocol. Although most DSP processors operate at an efficiency of approximately 10 million operations per second (Mops) per milliwatt (mW), the typical wireless protocol requires 100 Mops/mW.

This chapter presents the challenges and trade-offs in designing architectures for baseband processing in mobile communication devices. It gives an overview of baseband processing in SDR, followed by workload and performance analysis of a representative protocol. Next it describes the architectural features of a low power baseband architecture, SODA, followed by brief descriptions of other representative processor prototypes.

2.1 SDR Overview

SDR promises to solve the problems of supporting multiple wireless protocols and address future challenges. The SDR Forum, which is a consortium of service operators, designers and system integrators, defines SDR as:

> *"A collection of hardware and software technologies that enable reconfigurable system architectures for wireless networks and user terminals. SDR provides an efficient and comparatively inexpensive solution to the problem of building multi-mode, multi-band, multi-functional wireless devices that can be enhanced using software upgrades. As such, SDR can really be considered an enabling technology that is applicable across a wide range of areas within the wireless industry."*

Figure 2.3 shows the architecture for a typical third generation cellular phone. The architecture includes four major blocks: analog front-end, digital baseband, protocol processor, and application processor. The physical layer of wireless protocols includes both the analog front-end and the digital baseband. The analog front-end is usually

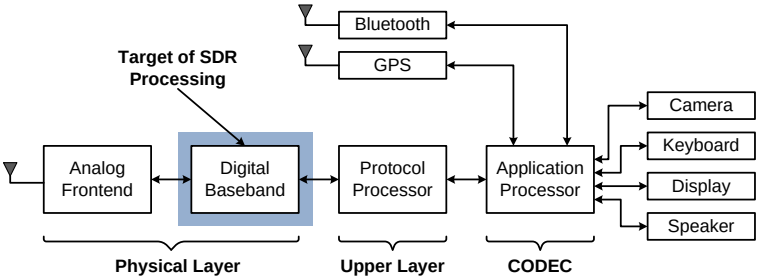


Figure 2.3: Architecture of a 3G cellular phone.)

implemented with analog ASICs. The digital baseband block performs the remaining physical layer operations and is also typically implemented with ASICs. The upper layers are implemented by the protocol processor and application processor which are usually SoCs and consist of general purpose embedded and DSP processors.

The objective of SDR is to replace the baseband ASICs with a programmable hardware platform, and to implement the baseband processing in software. Designing programmable analog front-ends is quite a challenge and is beyond the scope of this chapter. Here we focus on design of programmable digital baseband processing engines for SDR.

2.2 Workload Profiling and Characterization

2.2.1 The W-CDMA Physical Layer Processing

We select the wide-band code division multiple access (W-CDMA) protocol as a representative wireless workload case study for designing the SDR processor. This section provides a brief summary of its algorithms and characteristics. A more de-

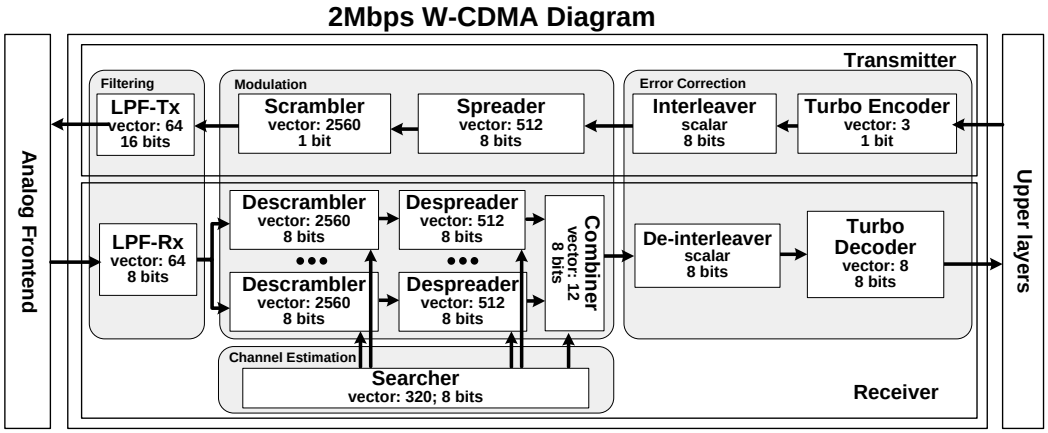


Figure 2.4: Physical layer operation of W-CDMA wireless protocol. Each block includes the algorithm's name, vector or scalar computation, vector width, and the data precision. The algorithms are grouped into four categories, shown in shaded boxes: filtering, modulation, channel estimation, and error correction.)

tailed analysis can be found in [49].

The W-CDMA system is one of the dominant 3G wireless communication networks where the goal is multimedia service including video telephony on a wireless link [36]. It improves over prior cellular protocols by increasing the data rate from 64 Kbps to 2 Mbps. The protocol stack of the W-CDMA system consists of several layers. At the bottom of the stack is the physical layer which is responsible for overcoming errors induced by an unreliable wireless link. The next layer is the medium access control (MAC) layer which resolves contention in shared radio resources. The upper layer protocols including MAC are implemented on a general purpose processor due to their relatively low computation requirements. In this section we focus on the computation model of the W-CDMA physical layer.

Figure 2.4 shows a high level block diagram of the digital processing in the W-

CDMA physical layer. It shows that the physical layer contains a set of disparate DSP kernels that work together as one system. There are four major components: filtering, modulation, channel estimation, and error correction.

Filtering. Filtering algorithms are used to suppress signals transmitted outside of the allowed frequency band so that interference with other frequency bands are minimized. The FIR filter operations in WCDMA can be very easily parallelized.

Modulation.

Modulation algorithms map source information onto the signal waveforms of the transmitter, and the receiver demodulates the signal waveforms back into source information. Two sets of codes are used for modulation: channelization codes and scrambling codes. Channelization codes are used so that the same radio medium can be used to transmit multiple different signal streams. Scrambling codes are used to extract the signal of a specific terminal among many transmitting terminals. On the receiver side, despreader is used to decode the channelization codes and descrambler is used to decode scrambling codes. Demodulation requires the transmitter and receiver to be perfectly synchronized. However, radio transmission suffers from multi-path fading effect, where multiple delayed versions of the same signal stream are received due to environment interference. A searcher is used to find the synchronization point of each of the delayed signal streams and each of these delayed signals is decoded with its own despreader and descrambler. The decoded output of the despreader/descrambler pairs are then combined together as the demodulated output.

Channel Estimation. Channel estimation algorithms calculate the channel con-

ditions to synchronize the two communicating terminals to ensure lock-step communication between the sender and the receiver. W-CDMA uses a searcher as its channel estimation algorithm. Searcher is called once per W-CDMA frame. There are two types of searchers — full searcher and quick searcher. In a group of 8 frames, the full searcher is used for the first frame, and the quick searcher is used for the remaining 7 frames. Both types of searchers consist of four steps: correlation, filtering out high frequency noise, detecting peaks, and global peak detection.

Error Correction Algorithms. Error correction algorithms are used to combat noisy channel conditions. The sender encodes the original data sequence with a coding scheme that inserts systematic redundancies into the output, which is decoded by the receiver to find the most likely original data sequence. Two types of error correction algorithms are used – convolutional coding and Turbo coding. Turbo coding is used for the 2Mbps data channel, while convolutional coding is used for all the other channels. For decoding, the Viterbi decoder is used for convolutional codes and the Turbo decoder is used for Turbo codes. Turbo decoding is usually the most computationally intensive algorithm in baseband processing. The corresponding decoder consists of two component decoders that are typically of type SOVA (Software Output Viterbi Algorithm) or based on MAP (Maximum A Posteriori) and connected together by interleavers. The interleaving pattern is specified by the W-CDMA standard, but the choice of component decoder is left to the designers.

Algorithms	Configurations	Vector Comp.	Vector Length	Bit Width	Comp. Mcycles/sec
W-CDMA (2Mbps)					
Scrambler	Defined in W-CDMA standard	yes	2560	1,1	240
Descrambler*	12 fingers, 3 base stations	yes	2560	1,8	2,600
Spreader	Spreading factor = 4	yes	512	8	300
Despreader*	12 fingers, 3 base stations	yes	512	8	3,600
PN Code (Rx)	3 base stations	no	1	8	30
PN Code (Tx)	Defined in W-CDMA standard	no	1	8	10
Combiner*	2Mbps data rate	partial	12	8	100
FIR (Tx)	4 filters x 65 coeff x 3.84Msps	yes	64	1,16	7,900
FIR (Rx)	2 filters x 65 coeff x 7.68Msps	yes	64	8,8	3,900
Searcher*	3 base stations, 320 windows	no	320	1,8	26,500
Interleaver	1 frame	no	1	8	10
Deinterleaver	1 frame	partial	1	8	10
Turbo Enc.	K=4	yes	3	1,1	100
Turbo Dec.*	K=4, 5 iterations	yes	8	8,8	17,500
*These algorithms have dynamically changing workloads that are dependent on channel conditions					

Figure 2.5: Workload analysis result of W-CDMA physical layer processing. "Vector comp" indicates whether the algorithm contains vector-based arithmetic operations. "Vector width" lists the native computation vector width. "Bit width" lists the data precision width. "Comp Mcycle/sec" lists the peak workload of running the algorithm on a general purpose processor.)

2.2.2 W-CDMA Workload Profiling

Figure 2.5 shows the result of workload profiling for W-CDMA in active mode with a throughput of 2Mbps [49]. The first column lists the W-CDMA algorithms. The second column lists the corresponding configurations for each of the algorithms. The third and fourth column lists the vector computation information for the algorithms. The fifth column lists the data precision width. The last column shows the peak workload of the algorithms. It is the minimum performance needed to sustain 2Mbps throughput, under the worst wireless channel conditions. To analyze it, the W-CDMA

model was compiled with an Alpha gcc compiler, and executed on M5 architectural simulator [12]. The peak workload of each algorithm was calculated by dividing the maximum number of processing cycles by its execution time.

The analysis shows that there are a set of key DSP algorithms that are responsible for the majority of the computation. These algorithms include the FIR filter, searcher, Turbo decoder, descrambler and despreader. The workloads of Viterbi and Turbo decoder require further verification because their processing times are not fixed. They are based on estimates that are derived from the protocol standard specifications for the different transmission channels that use these decoders.

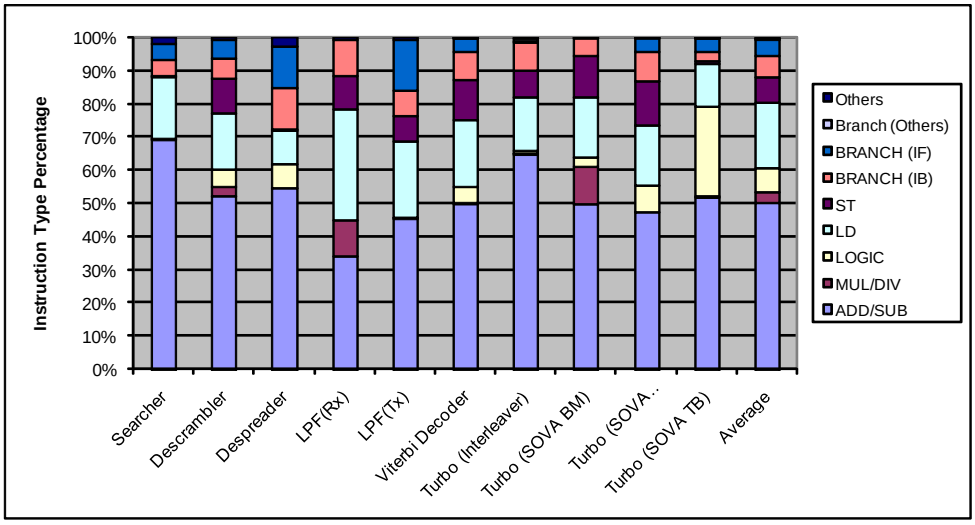


Figure 2.6: W-CDMA Instruction Type Breakdown.)

Instruction Type Breakdown. Figure 2.6 shows the instruction type breakdown for the W-CDMA physical layer and their weighted average for the case when the terminal is in the active state and has peak workload [49]. Instructions are grouped into seven categories: add/sub; multiply/divide; logic; load; store; branches;

and miscellaneous instructions. All computations are done using fixed point arithmetic. The first thing to notice is the high percentage of add/subtract instructions. They account for almost 50% of all instructions, with the searcher at about 70%. This is because the searcher's core operation is the inner product of two vectors, and the multiplication here can be simplified into a conditional complement which can again be implemented by a subtraction. Other hot spots, like the Turbo decoder, also have a large number of addition operations. The second thing to notice is the lack of multiplications/divisions ($< 3\%$ on average). This is because the multiplications of major algorithms are simplified into logical or arithmetic operations as discussed earlier. The multiplication of the combiner and Turbo encoder is not as significant because their workload is very small. One exception is multiplications in the LPF-Rx. Figure 2.6 also shows that the number of load/store operations are significant. This is because most algorithms consist of loading two operands and storing the operation result. Results also show that the portion of branch operations is about 10% on average. Most frequent branch patterns are loops with a fixed number of iterations corresponding to a vector size, and a conditional operation on vector variables. There are a few while or do-while loops, and most loops are 1 to 2 levels deep.

Parallelism in the Protocol. To meet the real-time W-CDMA performance requirement in software, the inherent algorithmic parallelism must be exploited. Columns 3 and 4 of Figure 2.5 show the potential parallelism that can be exploited either through Data Level Parallelism (DLP) or Thread Level Parallelism (TLP). Most of the algorithms operate on one dimensional vectors. Therefore, we define par-

allelism as the maximum vector width for these algorithms' arithmetic operations, which is shown in column 4.

From this result, we can see that the searcher, filter, scrambler, and descrambler all contain very wide vector operations. They also have a fair amount of TLP. For instance, the searcher operation can be executed as 5120 concurrent threads. For the case of scrambler and descrambler, TLP can be obtained by bisecting a wide vector into smaller ones. Although sliced vectors are not perfectly uncorrelated, we can execute the smaller vector operations with negligible dependency. Another example is the Turbo decoder which when implemented in a straight-forward manner, contains limited vector and TLP. This can be mitigated by use of sliding window techniques where multiple blocks of data can be processed at the same time.

Stream Computation. Another important characteristic of the W-CDMA protocol is that it is a streaming computation system. Here multiple DSP kernel algorithms are connected together in feed-forward pipelines and data streams through these pipelines sequentially. There is very little data temporal locality, which means that data does not get reused. Thus cache structures provide little additional benefits, in terms of power and performance, over software controlled scratchpad memories.

2.3 Architecture Design Trade-offs

This section describes some of the key challenges and trade-offs in designing processor architectures for supporting mobile SDR.

2.3.1 8-bit and 16-bit Fixed-point Operations

In W-CDMA wireless protocol, the majority of the algorithms operate on 1- to 8-bit data, with some algorithms operating on 16-bit data. In 802.11a wireless protocol, the majority of the DSP algorithms operate on 16-bit fixed point data, with a few algorithms operating on 8-bit data. A few DSP algorithms, such as FFT for the next generation 4G wireless protocols, may require 24-bit or 32-bit fixed-point precision. There are no wireless protocols that require floating-point support.

2.3.2 Vector-based Arithmetic Computations

Almost all computations in a DSP application involve vector and matrix arithmetic operations. This is true for wireless protocols as well. However, it is important to note that wireless protocols' computations are mostly on one-dimensional vectors.

DSP computations on vectors and matrices can generally be broken into two key components: data computation and data rearrangement. Data computation includes addition, multiplication, and other arithmetic and logic operations. Data rearrangement reorders the elements in vectors and matrices. In most DSP computations, it is not the data computation, but the data rearrangement that is the real performance bottleneck. Data computation limitations can always be addressed by adding more hardware computation units (such as adders and multipliers). However, there is no single common hardware solution that can accelerate all rearrangement operations for any arbitrary sized vectors and matrices. For example, FFT requires a butterfly operation, where pairs of data values in a vector are swapped. In comparison, an

error decoder may require an interleaver that transposes a matrix by blocks. The optimal hardware accelerators for these two patterns are quite different. An efficient design must exploit the specific data rearrangement patterns of the kernel algorithms. Since most of the vector arrangement patterns do not change during run-time, the interconnects between the ALUs and memories can be tailored to these algorithms.

2.3.3 Control Plane versus Data Plane

Complete software implementations of wireless protocols usually require two separate steps: 1) implementation of the DSP algorithms; and 2) implementation of the protocol system. The DSP algorithms are computational intensive kernels that have relatively simple data-independent control structures. The protocol system has relatively light computation load, but complicated data-dependent control behavior. Therefore, most commercial SDR solutions include a two-tiered architecture: a set of data processors that are responsible for heavy duty data processing; and a control processor that handles the system operations, and manages the data processors through remote-procedure-calls and DMA operations.

2.3.4 Scratchpad Memory versus Cache

In addition to data computation, wireless protocols must also handle the data communication between algorithms. These inter-algorithm communications are usually implemented by data streaming buffers. While the processors operate on the current data, the next batch of data can be streamed between the memories and

register files. There is very little data locality between the algorithms. This suggests that a scratchpad memory is better suited for SDR than a cache-based memory. Scratchpad memories are easier to implement for the hardware designers, and they consume less power. However, they do not have the hardware support for managing data consistencies, which means that the memory management must be performed explicitly by the software. This adds extra burden and complexities for the programmers and compilers. In addition, executing memory management instructions prevents processors from doing DSP computations, which may significantly degrade the overall performance. Therefore, many scratchpad memory systems provide DMA (Direct Memory Access) units. DMAs can move large blocks of data between the memories without interrupting the processors. However, these devices must also be explicitly managed by software.

2.3.5 Algorithm-Specific ASIC Accelerators

SDR is a solution where the entire wireless protocol is implemented in software. However for a SDR solution to be competitive with existing ASIC solutions, both in terms of performance and power, ASIC accelerators are sometimes used for error correction and filtering. This is because these two groups of algorithms are often the most computationally intensive algorithms in a wireless protocol, and different wireless protocols often use very similar algorithms for filtering and error correction. Previous studies have shown that there is an approximately 4-5x energy efficiency between an ASIC and a programmable solution for these two groups of algorithms.

Consequently, there have been multiple efforts both in academia and industry on developing combined Viterbi and Turbo ASIC accelerators. Texas Instrument already sells a Viterbi accelerator, as a part of a SOC package, that can support several configurations for multiple wireless protocols.

2.4 Baseband Processor Architectures

Many baseband processing architectures have been proposed in the last few years. They can be broadly categorized into SIMD-based architectures and reconfigurable architectures [73]. SIMD-based architectures usually consist of one or few high-performance DSP processors that are typically connected together through a shared bus, and managed through a general purpose control processor. Some SIMD-based architectures also have a shared global memory connected to the bus. SODA [54], which is described in detail, fall under the SIMD-based architecture category. Reconfigurable architectures, on the other hand, are usually made up of many simpler processing elements (PEs). Depending on the particular design, these PEs range from fine-grain ALU units to coarse-grain ASICs. The PEs are usually connected through a reconfigurable fabric. Compared with SIMD-based design, reconfigurable architectures are more flexible at the cost of higher power. In this chapter, we first describe SODA followed by descriptions of other SIMD and reconfigurable SDR architectures.

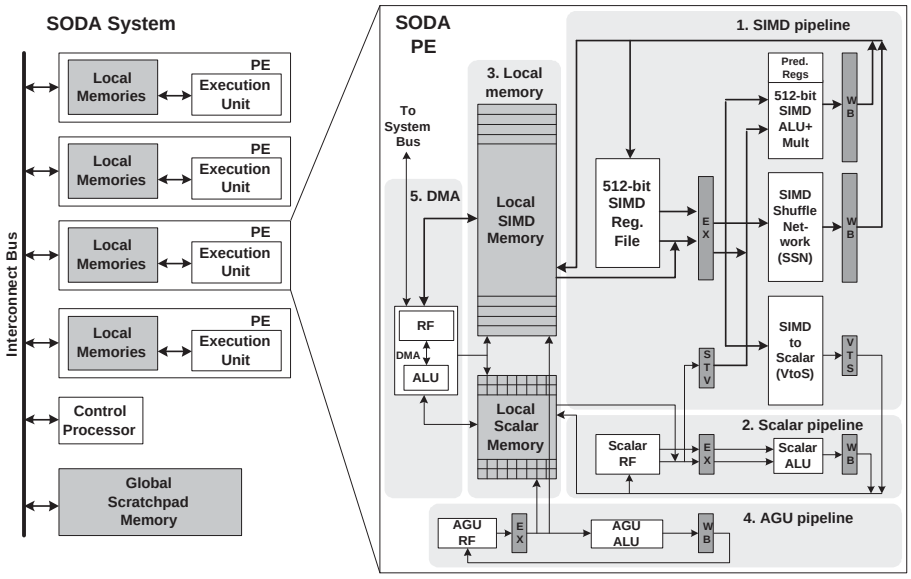


Figure 2.7: SODA Architecture. The system consists of 4 data processing elements (PEs), 1 control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a 32-lane 16-bit SIMD pipeline, a 16-bit scalar pipeline, two local scratchpad memories, an AGU for calculating memory addresses, and a DMA unit for inter-processor data transfer.

2.5 SODA Processor

SODA (Signal Processing On Demand) processor architecture is an academic research prototype for mobile SDR [54]. It is a SIMD-based DSP architecture designed to meet both the performance and power requirements for two representative protocols, WCDMA and IEEE 802.11a. The SODA multiprocessor architecture is shown in Figure 2.7. It consists of multiple processing elements (PEs), a scalar control processor, and global scratchpad memory, all connected through a shared bus. Each SODA PE consists of 5 major components: 1) an SIMD (Single Instruction, Multiple Data) pipeline for supporting vector operations; 2) a scalar pipeline for sequential operations; 3) two local scratchpad memories for the SIMD pipeline and the scalar

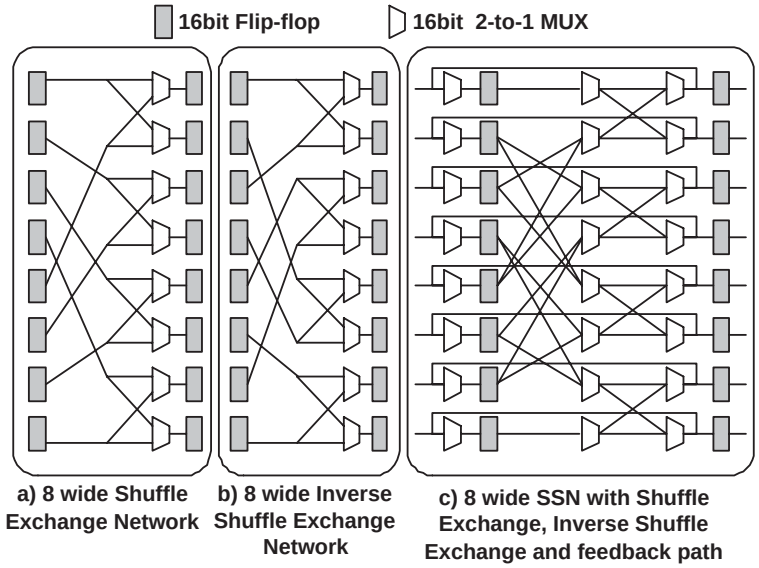


Figure 2.8: 8-wide SIMD Shuffle Network(SSN)

pipeline; 4) an Address Generation Unit (AGU) pipeline for providing the addresses for local memory access; and 5) a programmable DMA unit to transfer data between memories and interface with the outside system. The SIMD pipeline, scalar pipeline and the AGU pipeline execute in VLIW-styled lock-step, controlled with one program counter (PC). The DMA unit has its own PC, its main purpose is to perform memory transfers and data rearrangement. It is also the only unit that can initiate memory access with the global scratchpad memory.

The SIMD pipeline consists of a 32-lane 16-bit datapath, with 32 arithmetic units working in lock-step. It is designed to handle computationally intensive DSP algorithms. Each datapath includes a 2 read-port, 1 write-port 16 entry register file, and one 16-bit ALU with multiplier. The multiplier takes two execution cycles when running at the targeted 400MHZ. Intra-processor data movements are supported through

the SSN (SIMD Shuffle Network), as shown in Figure 2.8. The SSN consists of a shuffle exchange (SE) network, an inverse shuffle exchange (ISE) network, and a feedback path. By including both the SE and ISE networks, the number of iterations can be reduced. In addition to the SSN network, a straight-through connection is also provided for data that does not need to be permutated.

The SIMD pipeline can take one of its source operands from the scalar pipeline. This feature is useful in implementing trellis computations. It is done through the STV (Scalar-To-Vector) registers, shown in the SIMD pipeline portion of Figure 2.7. The STV contains 4 16-bit registers, which only the scalar pipeline can write, and only the SIMD pipeline can read. The SIMD pipeline can read 1, 2, or all 4 STV register values and replicate them into 32-element SIMD vectors. SIMD-to-Scalar operations transfer values from the SIMD pipeline into the scalar pipeline. This is done through the VTS (Vector-To-Scalar) registers, shown in Figure 2.7. There are several SIMD reduction operations that are supported in SODA, including vector summation, finding the minimum and the maximum.

The DMA controller is responsible for transferring data between memories. It is the only component in the processor that can access the SIMD, scalar and global memories. The DMA is also implemented as a slave device controlled by the scalar pipeline. However, it can also execute its own instructions on its internal register file and ALU, similar to the scalar pipeline. This gives the DMA the ability to access the memory in a wide variety of application-specific patterns without assistance from the master processor.

Arithmetic Data Precision. SODA PE only provides support for 8- and 16-bit fixed-point operations. This is because for both WCDMA and 802.11a wireless protocols, the majority of the DSP algorithms operate on 8-bit or 16-bit fixed point data. Each lane in the SODA PE supports 16-bit fixed-point arithmetic operations. The AGU registers are 12-bit, but only support 8-bit addition and subtraction. This is because AGU is used for software management of data buffers, in which 8 bits are sufficient. The higher 4 bits are used to address different PEs, as well as different memory buffers within PEs.

Vector Permutation Operations. With SODA's SSN network (shown in Figure 2.8), any 32-wide vector permutation can be performed with a maximum of 9 cycles. Combining with predicated move operations, the SSN network can support any vector length permutation. However, for every 32-wide vector permutation, one additional instruction must be used to setup the permutation network. This is because each MUX within the shuffle network requires its own control bit. Each iteration through the SSN requires 64 bits. For a maximum of 9 iterations, this requires 576 bits of control information. The SSN is not very efficient if there are many permutation patterns that are used frequently. However, the majority of the algorithms in SDR only use one or a few shuffle patterns, which makes the network setup overhead not significant.

Performance. For W-CDMA and 802.11a, the SODA architecture achieves large speed ups over a general purpose Alpha processor. For example, W-CDMA's searcher algorithm requires 26.5Gops on the general purpose processor; however the algorithm

requires only 200Mops on SODA. The performance improvements are mainly due to SODA's wide SIMD execution.

The RTL Verilog model of SODA was synthesized in TSMC 180nm technology. The results show for a clock frequency of 400 MHz, SODA consumes 2.95W for W-CDMA 2Mbps system and 3.2W for 802.11a 24Mbps system. However, with technology scaling, the power numbers are expected to reduce to acceptable levels such as 450mW for 90nm technology and 250mW for 65nm technology.

2.6 Other SIMD-based Architectures

In this section, we describe several other SIMD-based architectures proposed in industry and in academia. Like SODA, most of these architectures consist of one or more high-performance SIMD DSP processors which are connected through a shared bus. These architectures usually use software-managed scratchpad data memories to meet the real-time constraints. Many of them support VLIW execution by allowing concurrent memory and SIMD arithmetic operations. Some solutions also choose to incorporate accelerators for error correction algorithms, such as those based on convolutional and Turbo codes.

2.6.1 EVP

The Embedded Vector Processor, EVP [87] from NXP consists of a 16-lane 16-bit SIMD datapath, one scalar datapath, programmable memory, and VLIW controller. It allows five vector operations, four scalar operations, three address operations and

one loop control to be executed at once. The SIMD datapath consists of vector memory, 16 vector registers, load/store unit, ALU, MAC/shift unit, intravector unit, and code generation unit. The 16-bit datapath supports 8-bit and 32-bit data to allow word-level parallelism. It also supports data types such as complex numbers. In the SIMD datapath, the shuffle unit rearrange the elements of a single vector according to any pattern; the intravector unit supports summation, maximum, and split operations; and the code generation unit supports different CDMA-codes and CRC checks.

2.6.2 DXP

The Deep eXecution Processor (DXP) [42] from ICERA is a 2-LIW 4-lane SIMD architecture. Its key features are deeply pipelined execution units and a programmable configuration map which holds pseudo-static configurations and constants for the execution units. In the SIMD execution datapath, SIMD ALUs are chained to exploit the characteristics of streaming data. The chained operation saves register file access power at the cost of a less flexible SIMD datapath. Icera's processors do not use any hardware accelerators.

2.6.3 TigerSHARC

The TigerSHARC [25] implementation, ADSP-TS001, from Analog Devices adopts several mechanisms that are found in general-purpose processor such as a register-based load-store architecture with a static super-scalar dispatch mechanism, a highly

parallel short-vector-oriented memory architecture, support for multiple data types including 32-bit single-precision floating point and 8-bit/16-bit fixed point, parallel arithmetic instructions for two floating-point multiply-accumulate (MAC) operation or eight 16-bit MACs, 128-entry four-way set associative branch target buffer, and 128 architecturally visible, fully interlocked registers in four orthogonal register files. The TigerSHARC architecture provides concurrent SIMD arithmetic operations by having two 4-lane SIMD computation blocks controlled with two instructions. This VLIW/superscalar architecture fetches four instructions and issues one to four instructions per clock cycle. The 128 32-bit registers are memory mapped and divided into four separate register files of size 32x32 bit. The multiple data type supports subword parallelism in addition to inherent SIMD data parallelism.

2.6.4 MuSIC-1

The MuSIC-1[13] processor from Infineon consists of four SIMD core clusters, a general-purpose processor, shared memory and dedicated programmable processors for FIR filter and Turbo/Viterbi decoder. Each SIMD core contains four processing elements (PEs) and supports special instructions and LIW features for arithmetic operations, local memory accesses, and inter-PE communications. The general-purpose processor runs control programs to provide the PE controller with instruction addresses. The code and data are stored in an external memory and the on-chip memory consists of multiple banks to support simultaneous accesses.

2.6.5 Sandblaster

The Sandblaster [27] architecture from Sandbridge Technology is an example of a multi-threaded SIMD vector architecture. It consists of a RISC-based integer execution unit and multiple SIMD vector units. In addition, multiple copies of I-cache and data memory are available for each thread. Each instruction has four fields: load/store, ALU, integer multiplier, and vector multiplier. Therefore, the architecture can issue up to four simultaneous instructions where one may be a data parallel vector operation. This architecture also uses Token Triggered Threading, which consumes much less power than simultaneous multithreading because it issues instructions in round-robin fashion. The Sandblaster architecture supports up to eight threads.

2.6.6 SIMT

The Single Instruction stream Multiple Task architecture SIMT [63] from Linkoping University consists of Complex MAC (CMAC) SIMD units, Complex ALU (CALU) SIMD units, multiple memory banks, on-chip network, accelerators, and a controller unit. The CMAC unit consists of four complex MAC lanes, each of which uses 14x14 bit complex multipliers and has eight 2x40 bit accumulator registers. The CALU unit is similar to the CMAC unit except with simplified complex multiplier supporting only 0,+/-1,+/-i multiplications. The controller efficiently manages the two SIMD units and the matching memory bank system so that several threads can be run simultaneously. There are multiple memory banks and each bank contains its own address generation unit to minimize memory access conflicts. The programmable

reconfigurable crossbar switch is used as the on-chip network.

2.7 Reconfigurable Architectures

Reconfigurable architectures usually consist of many small PEs, which are connected through a interconnection fabric. These architectures can be categorized as either homogeneous or heterogeneous based on the type of PE. In addition, these PEs range from fine-grain LUTs to coarse-grain ALU units and even ASICs.

2.7.1 ADRES

An Architecture for Dynamically Reconfigurable Embedded System ADRES [59] from IMEC is an example of a coarse-grain reconfigurable tile architecture, which tightly couples a VLIW processor and coarse-grain reconfigurable matrix. This tightly coupled system has advantages such as shared resources, reduced communication costs, improved performance, and simplified programming model. The VLIW processor and the reconfigurable matrix share Functional Units (FUs) and Register Files (RFs). For the reconfigurable matrix part, there are many reconfigurable cells (RCs) which comprise FUs, RFs, and configuration RAM. These RCs are connected to nearest neighbor RCs and RCs within the same row or column in the tile. Therefore, kernels with a high level of DLP are assigned to the ADRES tiles whereas sequential codes are run on the VLIW processor. In ADRES architecture, the data communication is performed through the shared RF and memory, which is more compiler-friendly than the message-passing method. In addition, the ADRES relies on modulo schedul-

ing and traditional VLIW compiler support to exploit both DLP and ILP to maximize PE utilization.

2.7.2 Montium

The Montium [75] architecture from Delft University is a coarse-grained reconfigurable processor targeting 16-bit algorithms. Montium consists of two parts: 1) Communication and Configuration Unit (CCU) and 2) Montium Tile Processor (TP). The CCU configures the Montium TP and parts of the CCU itself for either "block-based communication" mode or "streaming communication" mode. The TP consists of five Montium ALUs and ten local memories, which are vertically segmented into five processing part array (PPA). A relatively simple sequencer controls the entire PPA and selects configurable PPA instructions that are stored in the decoders. Montium ALU consists of four functional units in level 1 followed by multiplier and adder in level 2. Neighboring ALUs can also communicate directly on level 2 in the tile processor. Finally, each local SRAM is 16-bit wide and accompanies each address generation unit (AGU), and the memory can be used for both integer and fixed point lookup tables.

2.7.3 Adapt2400 ACM

Adapt2400 ACM (Adaptive Computing Machine) [70] from QuickSilver consists of the two basic components: Nodes and Matrix Interconnection Network (MIN). Nodes are the computing resources in the ACM architecture that perform actual

work. Each node has its own controller, memory, and computational resources so that it independently execute algorithms that are downloaded in the form of binary files, called SilverWare. A node is capable of implementing a first come, first serve, non preemptive multitasking with the support of a hardware task manager. On the other hand, the MIN ties the heterogeneous nodes together carrying data, SilverWare, and control information between nodes as well as the external system. The network is hierarchically structured, and data within the MIN is transported in single 32-bit word packets to any other node or external interface. This heterogeneous coarse-grain reconfigurable architecture cooperates with InSpire SDK Tool Set to provide integrated scalable hardware/software platform.

2.7.4 XiRisc

XiRisc [15] from XiSystem is an example of a fine-grain reconfigurable architecture. This VLIW RISC processor features two concurrent execution datapaths and a set of DSP-like functional units that are shared between two datapaths. The concurrent execution path represented by a Pipelined Configurable Gate Array (PiCoGA) provides a run-time extension of the processor ISA for application-specific functions. The PiCoGA is a programmable pipelined datapath composed of an array or rows, which can function as customized pipeline stages. Each row is composed of 16 Reconfigurable Logic Cells (RLCs) containing two 4-input 2-output LUTs, four registers, and dedicated logic for a fast carry chain. Each RLC is connected to the others through a programmable interconnection matrix with 2-bit granularity switches. XiRisc ex-

ploits the synergy of the different execution units, ranging from a 32-bit dedicated MAC unit to bit-level processing blocks on PiGoGA, thereby increasing the execution efficiency and saving energy.

2.8 Conclusion

Wireless communications will soon be ubiquitous. Users will be able to access high speed internet with their mobile devices in every corner of the globe. To support this, mobile communication devices will have to be able to connect to a multitude of different wireless networks. SDR promises to deliver a cost-effective communication solution by supporting different wireless protocols in software. In this chapter, we presented low power digital baseband processors for SDR. We described the design rationales behind SODA. SODA is a multi-processor wide-SIMD architectures that exploit the data-level parallelism, the thread-level parallelism and the streaming nature of the inter-kernel computations. We also described the salient features of other existing baseband architectures that are either SIMD-based or reconfigurable tile-based. The lessons learned from these SDR architectures will help us in the development of future low power architectures for mobile devices that support the next generation wireless technologies.

CHAPTER 3

From SODA to Scotch: The Evolution of a Wireless Baseband Processor

In recent years, we have seen an increase in the number of wireless protocols that are applicable to different types of communication networks. Traditionally, the physical layer of these wireless protocols is implemented with fixed function ASICs. Software Defined Radio (SDR) promises to deliver a cost effective and flexible solution by implementing a wide variety of wireless protocols in software. Such solutions have many potential advantages: 1) Multiple protocols can be supported simultaneously on the same hardware, allowing users to automatically adapt to the available wireless networks; 2) Lower engineering and verification efforts are required for software solutions over hardware solutions; 3) Higher chip volumes because the same chip can be used for multiple protocols, which lowers the cost; and 4) Better support for future protocol changes. With the tremendous benefits of SDR, it is likely that many mobile communication devices are going to be supported by SDR technolo-

gies in the foreseeable future. Recently, Samsung was the first to announce a mobile phone that supports TD-SCDMA/HSDPA/GSM/GPRS/EDGE standards using a SDR baseband processor [64].

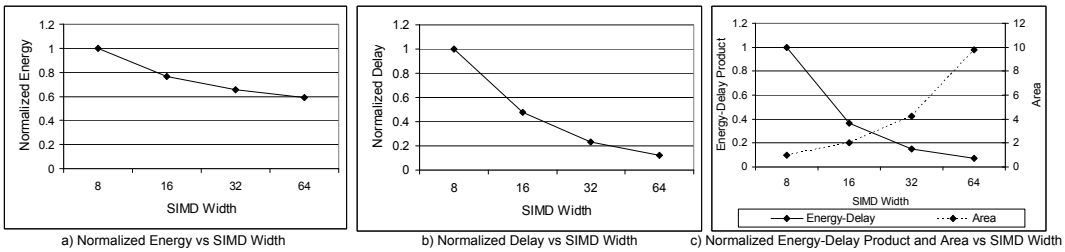
Wireless Protocol Workloads. The computational requirements of current generation wireless protocols are orders of magnitude higher than the capabilities of modern general-purpose processors. A wireless protocol processor must sustain this high computation throughput while meeting the strict power budget of an embedded mobile terminal. This is the reason why many wireless protocols to date are implemented with custom hardware. The challenge of SDR is to meet these performance and power requirements while maintaining the flexibility of a programmable processor. Previous work on workload characterization of 3G wireless and other wireless baseband processing protocols showed that there exists large amount of data-level parallelism (DLP), with the majority of the operations being long vector arithmetic computations [49].

SODA Processor Architecture. The SODA multi-core architecture was proposed for supporting 3G wireless baseband processing [54]. SODA consists of an ARM control processor, four data processing elements (PEs), and a shared global scratchpad memory. Designed for long vector arithmetic operations, each SODA PE includes a wide 512-bit SIMD unit that is capable of operating on 32 16-bit elements concurrently. In addition, each PE also has a scalar datapath, local scratchpad memories, address generation unit (AGU), and direct memory access (DMA) support.

Ardbeg Processor Architecture. A commercial prototype, Ardbeg, based on SODA has been developed by ARM Ltd. Ardbeg shares many features with SODA. It is a multi-core architecture, with one control processor and multiple data PEs. Each data PE contains a 512-bit wide SIMD datapath. Ardbeg adds algorithm-specific hardware and optimizes the architecture specifically for wireless applications. In contrast, SODA was designed to test the feasibility of a fully programmable wireless baseband solution and purposely avoided algorithm-specific designs. While SODA was focused on supporting 3G wireless protocols, Ardbeg is also designed to scale for future protocols. Overall, Ardbeg achieves between 1.5-7x speedup over SODA while operating at a lower clock frequency.

The evolution of SODA to Ardbeg was a process with many design choices. The major design choices can be grouped into the following three categories:

- **Optimized Wide SIMD Design.** SODA was originally designed in 180nm technology. In 90nm technology, the SIMD datapath choices need to be re-examined. We re-evaluated the SIMD width and found that SODA's original 32-lane 512-bit SIMD datapath is still the best SIMD design point in 90nm. On the other hand, the SIMD shuffle network redesigned to support faster vector permutation operations. Compared with SODA's two cycle multiplier, 90nm technology enables a single cycle multiplier, which provides significant speedup for several key SDR algorithms.
- **LIW Support for Wide SIMD.** For W-CDMA and 802.11a, the SODA SIMD ALU unit is utilized around 30% of the total execution cycles. LIW execution on the SODA SIMD pipeline was considered a poor choice due to the low utilization of the SIMD units and was abandoned due to the concern about the extra power and area costs of adding more SIMD register file ports. We revisited this concern when designing Ardbeg in order to improve the computational efficiency. The result was Ardbeg issuing two SIMD operations each cycle. Not all combinations of SIMD instructions are allowed. Ardbeg implements a restricted LIW designed to support the most common parallel execution patterns found in SDR algorithms with minimal hardware overhead. Our analysis shows that having this restricted LIW support would provide better performance and power efficiency over single-issue SIMD datapath, but also that having larger issue widths does not provide any additional performance benefit over a simple two-issue LIW.



a) Normalized Energy vs SIMD Width b) Normalized Delay vs SIMD Width c) Normalized Energy-Delay Product and Area vs SIMD Width

Figure 3.1: Plots of normalized energy, delay, and energy-delay product versus area plots for different Ardbeg SIMD width configurations running 3G wireless algorithms. The results are normalized to the 8-wide SIMD design.

- **Algorithm Specific Hardware Acceleration.** A set of algorithm specific hardware is also added to the Ardbeg architecture. These include an ASIC accelerator for Turbo decoder, block floating point support, and fused permute and arithmetic operations. This set of algorithm specific hardware was chosen to achieve higher computational efficiency while maintaining enough flexibility to support multiple protocols.

The rest of the chapter is organized as follows. Section 3.1 gives a brief description of the overall architectures of SODA and Ardbeg. Section 3.2 presents the architectural evolution from SODA to Ardbeg. We provide experimental results and analysis to explain the rationale behind the major Ardbeg architectural design decisions. Section 3.3 presents the performance results of the two architectures for various wireless protocols.

3.1 Architecture Overview

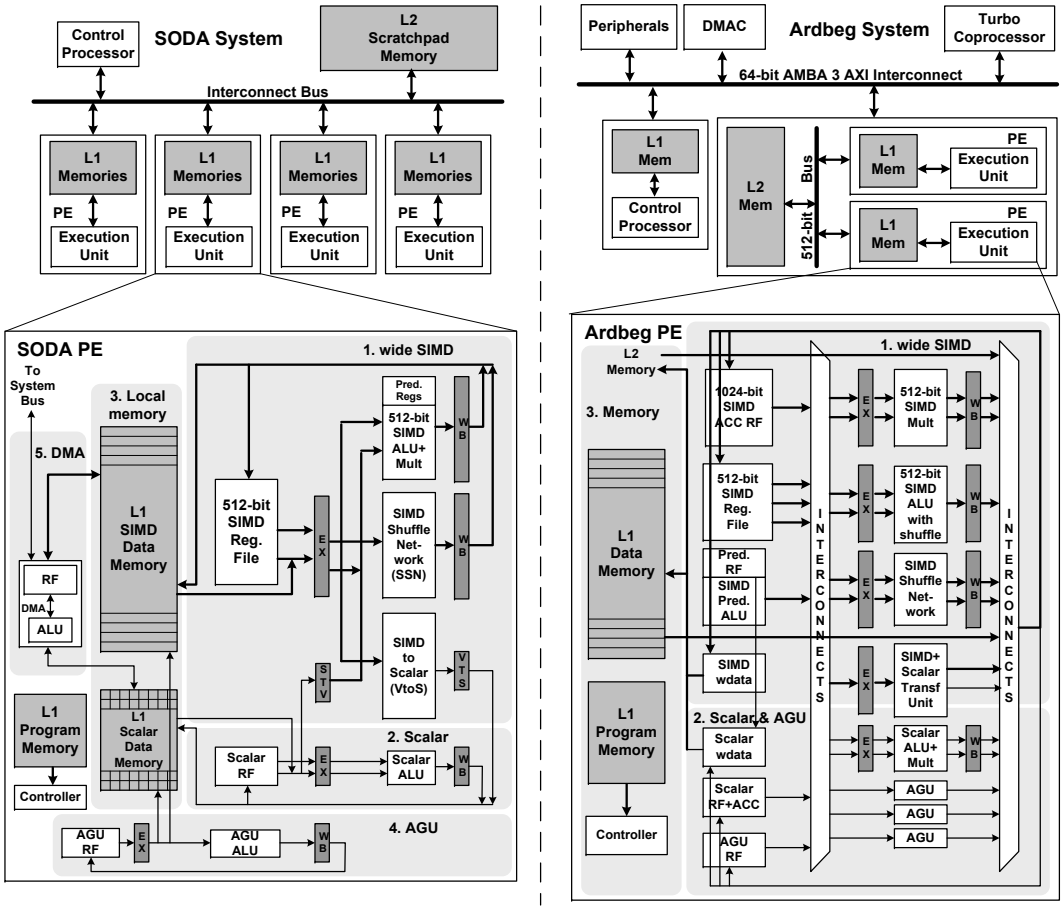
Because the majority of the SDR computations are based on vector arithmetic, previous work on SODA has demonstrated that having a wide SIMD datapath can achieve significant speedup while maintaining low power consumption [54]. With a 32-lane SIMD datapath, SODA was able to achieve an average of 47x speedup

for W-CDMA DSP algorithms over a general purpose Alpha processor. However, as an initial research prototype, many architectural optimizations were overlooked. Ardbeg has improved upon the base SODA architecture, as will be illustrated in the subsequent sections. This section provides an overview of the SODA and Ardbeg architectures and summarizes the differences.

3.1.1 SODA Architectural Overview

The SODA multicore system is shown on the left in Figure 3.2. It consists of four data PEs, a scalar control processor, and a global L2 scratchpad memory, all connected through a shared bus. Each SODA PE consists of five major components: 1) an SIMD datapath for supporting vector operations; 2) a scalar datapath for sequential operations; 3) two local L1 scratchpad memories for the SIMD pipeline and the scalar pipeline; 4) an AGU pipeline for providing the addresses for local memory access; and 5) a programmable DMA unit to transfer data between memories. The SIMD, scalar, and AGU datapaths execute in lock-step, controlled with one program counter.

The SIMD datapath consists of a 32-lane, 16-bit datapath, with 32 arithmetic units working in lock-step. It is designed to handle computationally intensive DSP algorithms. Each datapath includes a 2 read-port, 1 write-port 16 entry register file, and one 16-bit ALU with multiplier. Synthesized in 180nm technology, the multiplier takes two execution cycles when running at the targeted 400 MHz. Intra-processor data movements are supported through the SSN (SIMD Shuffle Network). The SSN



Comparison summary of the architectural features of SODA and Ardbeg

	SODA	Ardbeg
PE Architecture		
Organization	SIMD + scalar + AGU	SIMD + scalar + AGU
Execution Model	SIMD/Scalar LIW	SIMD/Scalar and SIMD/SIMD LIW
PE Frequency	400MHz (180nm)	350MHz (90nm)
SIMD Architecture		
SIMD Datapath	single issue	ALU + memory + SSN
SIMD Width	512 bits	512 bits
Data Precision	16-bit FXP	8/16/32-bit FXP
Block Floating Point	no	yes
SIMD Predication	yes	yes
SIMD Mult Latency	2 cycles	1 cycle
SIMD Shuffle Network	32-lane 1-stage iterative perfect shuffle	128-lane 7-stage Banyan network
Reduction Network	reduction tree	pair-wise operation/reduction tree
SIMD Reg File	2 read/1 write ports, 16 entries	3 read/2 write ports, 15 entries
L1 Memory	8KB	32KB~128KB
L2 Memory	64KB	256KB~1MB
Others		
Coprocessor	no	Turbo coprocessor
Compiler Opti.	no	software pipelining

Figure 3.2: SODA and Ardbeg architectural diagrams, and a summary of the key architectural features of the two designs.

consists of a shuffle exchange (SE) network, an inverse shuffle exchange (ISE) network, and a feedback path. Various SIMD permutation patterns require multiple iterations of the SSN network. SIMD-to-scalar (VTS) and scalar-to-SIMD (STV) units are used to transfer data between the SIMD and scalar datapath.

3.1.2 Ardbeg Architecture

The Ardbeg system architecture is shown on the right in Figure 3.2. Similar to the SODA architecture, it consists of multiple PEs, an ARM general purpose controller, and a global scratchpad memory. The overall architecture of the Ardbeg PE is also very similar to the SODA PE, with a 512-bit SIMD pipeline, scalar and AGU pipelines, and local memory. Ardbeg was designed using the OptimoDE framework [19]. The framework allowed the creation of custom VLIW-style architectures and evaluating many architectural design trade-offs quickly. These trade-offs will be discussed in the next section. The instruction set for Ardbeg was derived from the ARM NEON extensions [8]. The bottom portion of Figure 3.2 also provides a side-by-side comparison between the two architectures.

The Ardbeg system has two PEs, each running at 350 MHz in 90nm technology. In addition, it includes an accelerator dedicated to Turbo decoding. In comparison, in the SODA system, Turbo decoding is allocated to one of the four PEs. Both the Ardbeg and SODA PEs have three major functional blocks: SIMD, scalar, and AGU.

The SODA and Ardbeg PEs both support 512-bit SIMD operations. The SODA PE only supports 16-bit fixed point operations, whereas the Ardbeg PE also supports

8-, 32-bit fixed point, as well as 16-bit block floating point operations. Support for 8-bit helped lower the power for many of the W-CDMA kernels that only needed 8-bit precision. Legacy wireless protocols like 802.11b have many kernels that operate on 8-bit data and do not require the 16-bit precision that SODA supported. Support for 32-bit was added in order to accommodate future algorithms which may require higher precision.

One of the key differences between Ardbeg and SODA is that the Ardbeg PE supports LIW execution on its SIMD pipeline, allowing different SIMD units to execute in parallel. In the SODA PE, only one SIMD operation can be issued per cycle. Also, SODA's SIMD permutation network is a single stage, multi-cycle perfect shuffle network, whereas Ardbeg's SIMD permutation network is a modification of a 7-stage, single-cycle Banyan network. Detailed analysis of the Banyan network can be found in [29]. The shuffle instructions that are used in Ardbeg are an extended set of the ARM Neon permutation instructions. In terms of the number of registers, the Ardbeg PE has additional SIMD and scalar accumulators to hold the output of the multiplier. Ardbeg has a 1-cycle multiplier, whereas SODA's multiplier requires 2 cycles.

The memory hierarchy in Ardbeg is similar to the Cell processor [35] in that each PE has a local scratchpad memory and PEs share a global memory, which are all explicitly managed. The DMA can transfer data between each of the PE's local memories and also to and from the global memory. A write buffer to memory is also added to Ardbeg. Both Ardbeg's local and global memories are larger than SODA's memories. In addition, instead of the separate scalar and SIMD memories in SODA,

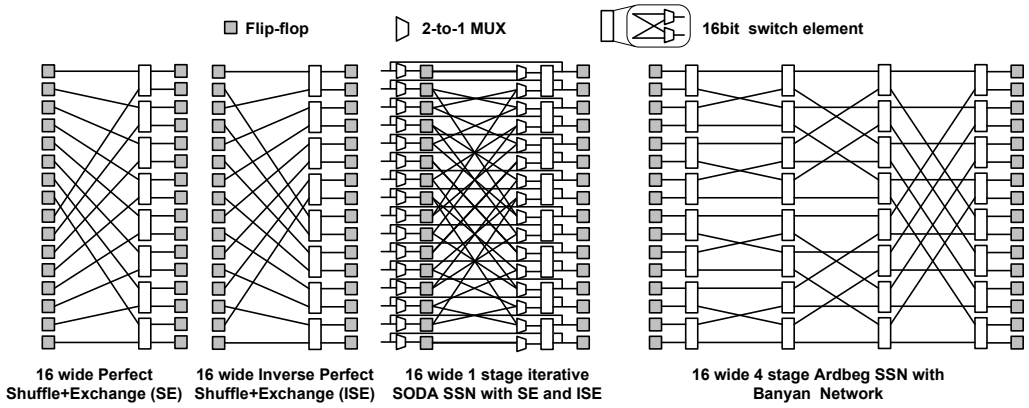
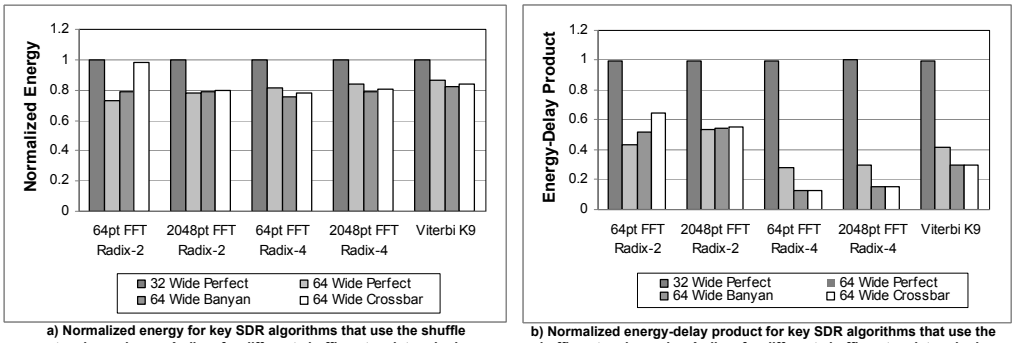


Figure 3.3: SIMD shuffle network for the SODA PE and the Ardbeg PE. For illustration clarity, these examples show 16-wide shuffle networks. The SODA PE has a 32-wide 16-bit 1-stage iterative shuffle network, and the Ardbeg PE has a 128-lane 8-bit 7-stage Banyan shuffle network.

Ardbeg has one unified local scratchpad memory. Because many DSP algorithms don't have much scalar code, it is more efficient to share the memory space between the SIMD and scalar datapath.

For system mapping in Ardbeg, the application is represented as a task graph and a set of filters (like StreamIt [86]). The compiler performs coarse-grain software pipelining to assign tasks to PEs and inserts DMA transfers to transfer data between PEs. Streaming dataflow is explicit, so data follows the task assignment and no special data partitioning is required. Oversubscription of the PE's local memory is handled by spilling sections to the global memory. More details about system mapping and scheduling can found in [46].



a) Normalized energy for key SDR algorithms that use the shuffle network running on Ardbeg for different shuffle network topologies

b) Normalized energy-delay product for key SDR algorithms that use the shuffle network running Ardbeg for different shuffle network topologies

Figure 3.4: Normalized energy and energy-delay product for key SDR algorithms running on Ardbeg for different shuffle network topologies.

3.2 Architectural Evolution: SODA to Ardbeg

3.2.1 Optimized Wide SIMD Design

Since the majority of the SDR algorithms operate on wide vectors, SODA used a wide SIMD datapath, namely a 512-bit 32-lane SIMD datapath. Ardbeg has also adopted the 512-bit SIMD datapath, and extended it to support 64-lane 8-bit and 16-lane 32-bit SIMD arithmetics. The SIMD shuffle network (SSN) is redesigned to provide better performance at lower power. With a target frequency of 350 MHz, implementing Ardbeg in 90nm also allows for a single-cycle SIMD multiplication unit. The rest of this section explains our rationale for these architectural design decisions. For each of the studies, we synthesized in 90nm the different sizes and configurations of the functional units and calculated the number of cycles and energy to run the kernels.

SIMD Width Analysis. The SODA architecture was designed using a 180nm process technology. A 32-lane configuration was found to be the most energy efficient

SIMD configuration. One of the first Ardbeg design considerations is to determine if SODA's proposed 32-lane SIMD is still the best configuration in 90nm. In this study, we examine SIMD configurations ranging from 8-lane to 64-lane. Figures 3.1a and 3.1b show the normalized energy and delay for different SIMD width Ardbeg processors synthesized for 350 MHz in 90nm for various key SDR algorithms like FFT, FIR, W-CDMA Searcher, and Viterbi. All values are normalized to the 8-wide SIMD configuration.

The figures show that as SIMD width increases, both delay and energy consumption decreases. The delay result is expected as wider SIMD configurations can perform more arithmetic operations per cycle. While power consumption of a wider SIMD is greater, because wider SIMD takes fewer cycles to perform the same number of arithmetic operations and the control overhead per instruction is amortized across the SIMD, the overall energy consumption is lower for wide SIMD. Figure 3.1c shows the energy-delay product and the area of these SIMD configurations. A 32-lane SIMD configuration has better energy and performance results compared to the 8-lane and 16-lane SIMD configurations. A 64-lane SIMD configuration has slightly better results than the 32-lane SIMD configuration. If energy and delay are the only determining factors, then implementing Ardbeg with a 64-lane SIMD configuration is probably the best design choice. However, in a commercial product, area is also a major design factor. As SIMD width increases, area increases at a higher rate than the decrease in either energy or delay. Taking area into account, Ardbeg chose to keep SODA's 32-lane SIMD datapath configuration.

SIMD Permutation Support. It is common for DSP algorithms to rearrange vector elements before computation. One of the central challenges in designing a wide SIMD architecture is the vector permutation support. A partially connected SIMD shuffle network (SSN) was employed in SODA as shown in Figure 3.3. It is a 32-lane single stage iterative shuffle network consisting of a perfect shuffle and exchange (SE) pattern, an inverse perfect shuffle and exchange (ISE), and a feedback path. Multi-stage networks were considered, but in 180nm technology the delay for the multi-stage network was more than one clock cycle running at 400 MHz. In addition, there were concerns that the area for a multi-stage network may be too large. Therefore, a multi-cycle iterative shuffle network was chosen for SODA. In designing Ardbeg's shuffle network in 90nm, several SIMD configurations and network topologies were revisited. We first examined the performance and energy trade-offs of a wider SSN. Figure 3.4a provides the normalized energy of key SDR algorithms for 32-lane and 64-lane SODA SSNs. The SIMD datapath is still 32-lane for both SSN configurations. The 64-lane SSN operates on two 32-lane SIMD vectors by reading from two SIMD register file ports. Filter algorithms are excluded from this study because their implementations do not use the SSN. Compared to the 32-lane network, a 64-lane network consumes approximately 20% less energy across all benchmarks, despite the fact that the 64-lane network consumes more power than the 32-lane network. This is because these DSP algorithms operate on long vectors, where the vector width is greater than the SIMD width. Because many long vector permutations require extra instructions to store intermediate permutation results, the number of instructions required to perform

long vector permutations does not always scale linearly with the width of the SSN. A smaller SSN requires more instructions than a larger SSN, which results in more frequent SIMD register file accesses and other execution overhead.

We then examined the performance and energy trade-offs of different network topologies. In addition to the iterative SE/ISE network, we also examined a 64-lane Banyan network and full crossbar. The SE/ISE and the Banyan networks are shown in Figure 3.3. The Banyan network is a flattened 7-stage network that can perform 64-lane 16-bit vector permutations in a single cycle. Energy and energy-delay products of these three networks are shown in Figure 3.4. For radix-2 FFT, a 64-lane iterative SE/ISE network is slightly better than a 64-lane Banyan network, because there exists an implementation of this algorithm that is optimized specifically for the SE/ISE network. However, if an algorithm requires more complex permutation patterns, such as the radix-4 FFT and Viterbi algorithms, the single-cycle Banyan network has shorter delays than the multi-cycle iterative shuffle network. Though the difference in energy consumption between the iterative SE/ISE network and 64-lane Banyan is not very large, Figure 3.4b shows that the single-cycle Banyan network has better energy-delay product than the iterative SE/ISE network. Overall, the Banyan network performs as well as the full crossbar, and with $\sim$17x area savings compared to the crossbar. Therefore, Ardbeg's SSN is implemented with the Banyan network. In addition to supporting 16-bit permutations, Ardbeg's Banyan network can also support 32-lane 32-bit and 128-lane 8-bit vector permutations.

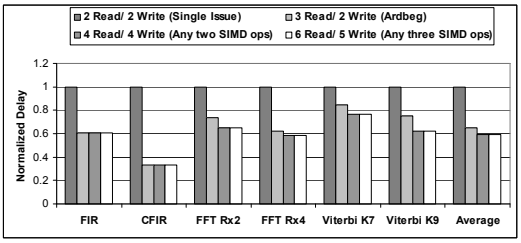
Reduced Latency Functional Units. In SODA, the 180nm process technology

Ardbeg Function Units	# of SIMD RF Ports Required
Memory Load/Store	1 read / 1 write
SIMD Arithmetic	2 read / 1 write
SIMD Multiply	2 read / No write (ACC RF)
SIMD Shuffle	2 read / 2 write
SIMD+Scalar Transfer Unit	1 read / 1 write
ACC-to-SIMD Move	1 read / 2 write
SIMD Comparison	2 read / 1 write (Pred. RF)

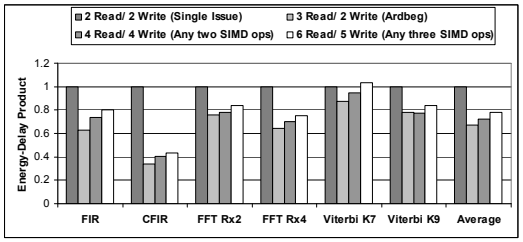
a) This table lists the function units in Ardbeg, and the number of SIMD register file ports required for each unit. At most two SIMD operations can be issued every cycle.

	Mem.	Arith.	Mult.	Shuffle	Trans.	Move	Comp.
Mem.	NA	--	--	--	--	--	--
Arith.	High	NA	--	--	--	--	--
Mult.	High	Mid	NA	--	--	--	--
Shuffle	Low	High	Mid	NA	--	--	--
Trans.	High	Mid	High	Mid	NA	--	--
Move	Low	Low	High	Low	Low	NA	--
Comp.	Low	Low	Low	Low	Low	Low	NA

b) Shaded box means Ardbeg can issue instructions on these two function units in the same cycle. "High/Mid/Low" represent the relative usage frequency for each pair of function units within wireless protocols.



c) Normalized delay for various key SDR kernels running on Ardbeg with different VLIW configurations.



d) Normalized energy-delay product for various key SDR kernels running on Ardbeg with different VLIW configurations

Figure 3.5: Ardbeg VLIW support. The results are shown for software pipelined Ardbeg assembly code. Ardbeg has 7 different function units, as listed in sub-figure a. These seven function units share 3 SIMD register file read and 2 write ports. At most two SIMD operations can be issued per cycle, and not all combinations of SIMD operations are supported. Different LIW configurations are evaluated in terms of delay and energy-delay product, as shown in sub-figures c and d.

put a constraint on the latency of the functional units. Because SODA's target frequency was set to 400 MHz, the multiplier had to be designed with a 2-cycle latency. For Ardbeg, the target frequency is set at 350 MHz due to the control latency for controlling the LIW pipeline. With 90nm process technology, Ardbeg implements power efficient multipliers with single cycle latency. Because many DSP algorithms require a large number of multiplication operations, the single-cycle multiplication results in up to 2x performance improvement (see Section 3.3).

3.2.2 LIW SIMD Execution

For W-CDMA and 802.11a, the SODA SIMD ALU unit is utilized around 30% of the total time. The poor utilization is mainly due to the fact that SODA's SIMD datapath is shared with the memory access unit and the SSN. Functional unit under-utilization not only increases register file accesses but also execution time. LIW execution on the SIMD pipeline was considered for the SODA architecture to reduce these problems, but was abandoned due to the concern about the extra power and area costs of adding more SIMD register file ports. In SODA, the SIMD register file was the largest power consumer, accounting for approximately 30% of the total power. When designing Ardbeg, we re-evaluated LIW execution to decrease execution time and to reduce register file power.

To determine the effectiveness of LIW, we analyzed different kernels within the set of wireless protocols and found how often functional units could be used in parallel. There are 7 SIMD function units in Ardbeg's SIMD datapath as listed in Figure 3.5a, along with their register port requirements. The values listed in Figure 3.5b represent the frequency that the functional units could execute instructions in parallel. We can see that there are few instruction combinations that occur in high frequency in the algorithms. This suggests that we could implement a LIW and minimize the number of register file ports to save power while increasing throughput.

We have studied the performance and energy efficiency trade-offs for supporting various LIW configurations in Ardbeg. We examined configurations with a different number of SIMD register file read and write ports: single issue with 2 read and 2

write ports, restricted 2-issue LIW support with 3 read and 2 write ports, full 2-issue LIW support with 4 read and 4 write ports, and full 3-issue LIW support with 6 read and 5 write ports. The performance and energy efficiency results of the synthesized implementations are shown in Figures 3.5c and 3.5d. The performance is normalized to the cycle count for a single issue Ardbeg. We found that LIW support is beneficial for many key SDR algorithms. This indicates that there is still instruction-level parallelism (ILP) within SIMDized Ardbeg assembly code. However, we also find that a 2-issue LIW configuration is enough to capture the majority of the ILP, as a 2-issue configuration results in a similar speedup as a 3-issue configuration. This is because a significant portion of the parallelism is already exploited through SIMD execution. Also, many SIMD operations cannot execute in parallel simply because of data dependencies.

LIW execution is supported in Ardbeg, but with restrictions on the combinations of instructions that can be issued in a cycle. This results in slower speedup than a full 2-issue LIW, but provides better energy-delay product due to a lesser number of SIMD register file ports. The set of valid Ardbeg LIW instruction combinations are shown in Figure 3.5b as shaded boxes. Among these LIW combinations, overlapping memory accesses with SIMD computation is the most beneficial because most DSP algorithms are streaming. The SIMD arithmetic/multiplication and SIMD-scalar transfer combination is the most beneficial for filter-based algorithms. And, the SIMD multiply and move combination is the most beneficial for FFT-based algorithms. The responsibility is left to the compiler to produce valid instruction schedules that can

utilize this capability. Overall, Ardbeg's SIMD datapath can achieve an average of 60% SIMD ALU utilization with restricted LIW execution.

3.2.3 Application Specific Hardware Acceleration

Designing an application specific processor for SDR is a balancing act between programmability and performance. A processor must be flexible enough to support a multitude of wireless protocols. However, too much flexibility results in an inefficient architecture that is unable to meet the stringent performance and power requirements. SODA was designed to meet the throughput requirements of 3G wireless protocols, such as W-CDMA and 802.11a. In addition to these 3G protocols, Ardbeg was designed with future wireless protocols in mind. Hardware accelerators were added in Ardbeg to increase computational and energy efficiency.

3.2.3.1 Turbo Coprocessor

Turbo decoding is one of the error correction algorithms used in the W-CDMA wireless protocol for the 2 Mbps data communication channel. It is the most computationally intensive algorithm in W-CDMA. In addition, it is the most difficult algorithm to vectorize. Unlike the wide vector arithmetics of other SDR algorithms, Turbo decoder operates on narrow 8-wide vectors. Parallelization techniques can be applied to utilize the 32-lane SIMD datapath by processing four 8-wide vectors concurrently [55]. However, this requires concurrent memory accesses for the 4 vectors. Because the SODA and Ardbeg PEs only have one memory port, serialized memory

accesses become the bottleneck of the algorithm. Software pipelining cannot help, because the main loop in the decoder has data dependencies between consecutive loop iterations. The combination of these factors makes Turbo decoder the slowest algorithm on the SODA and Ardbeg PEs. The SODA and Ardbeg PEs can sustain 50-400 Mbps of data throughput for various FIR and FFT algorithms, but only 2 Mbps for Turbo decoder. The SODA PE was targeted at 400 MHz because of the computational requirements of the Turbo decoder. Offloading the Turbo decoder to a coprocessor allows the Ardbeg PE to lower the target frequency to 350 MHz.

Because of the high computational requirements, one SODA PE is dedicated solely for Turbo decoding, accounting for roughly 25% of the total power consumption. In a 90nm implementation, a SODA PE would be able to maintain 2 Mbps while consuming an estimated power of 111mW. In contrast, in 130nm, an ASIC Turbo decoder is able to support 13.44 Mbps while consuming 262 mW [80]. In 90nm technology, this roughly translates to 21 mW for sustaining 2 Mbps throughput. Therefore, in the case of Turbo decoder, the cost of programmability is approximately 5x in terms of power consumption. Furthermore, since 2 Mbps is the maximum throughput for a SODA PE running at 400 MHz, higher decoding throughput, as required by future protocols, would require either higher frequencies or multiple PEs. Both these considerations led Ardbeg to offload Turbo decoding on a coprocessor. Other DSP systems aimed at wireless communications, such as the Phillips' EVP [88], have also taken a similar approach.

3.2.3.2 Application Specific Instruction Set Extensions

Many wireless protocols can share the same error correction ASIC accelerator, but the approach of using more ASIC accelerators is not viable due to the inherent differences in the protocols. However, while the algorithms are different, they share many commonalities within their basic computational blocks. This allows us to increase computational efficiency by adding re-usable algorithm-specific instructions.

Block Floating Point Support. Large point FFTs are used in many wireless protocols. Even though the input and output data are 16-bit numbers, the intermediate results often require higher precision. Block floating point (BFP) provides near floating point precision without its high power and area costs. In floating point, each number has its own mantissa and the exponent. In BFP, each number has its own mantissa, but the exponent is shared between a block of numbers. BFP is commonly used in ASIC design, but very few programmable processors have provided direct hardware support. A key operation in BFP is finding the maximum value among a block of numbers. Most DSP processors support this operation in software. However, for the 32-lane Ardbeg SIMD datapath, this is inefficient, as all lane values must be compared. In Ardbeg, BFP is supported through special hardware that finds the maximum value in a 32-lane 16-bit vector. Each instruction that supports BFP has special flags which, when enabled, automatically perform value tracking and store the result in a special register. BFP support allows the Ardbeg PE to operate in the 16-bit SIMD datapath mode for FFT computations, instead of the 32-bit SIMD datapath mode that would have been required to satisfy precision requirements. Though

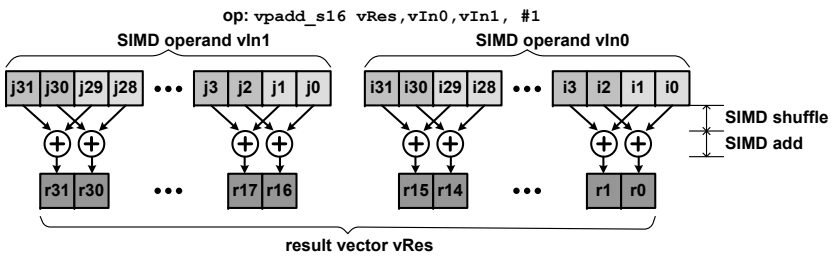


Figure 3.6: Ardbeg's pair-wise butterfly SIMD operation implemented using a fused permute and ALU operation. The figure shows pairs of a 2-element butterfly operation. Ardbeg supports pairs of 1-,2-,4-,8-,and 16-element butterfly of 8- and 16-bits. This butterfly operation uses the inverse perfect shuffle pattern because the input to each SIMD ALU lane must come from the same SIMD lane.

FFT is where BFP is currently used, any algorithm that requires higher precision can utilize the BFP instruction extensions.

Fused Permute-and-ALU Operations. It is common in DSP algorithms to permute the vectors before performing arithmetic operations. An example is the butterfly operation in FFT, where vectors are first shuffled in a butterfly pattern before vector adds and subtracts are performed. In an earlier design of the SODA PE, the SSN was placed in front of the SIMD ALU, so that permute-and-arithmetic operations could be performed in one instruction. However, arithmetic operations that do not require permutations always go through the SSN, increasing the number of pipeline stages and power consumption. So in the final SODA PE design, the SSN was taken out of the arithmetic pipeline, and placed as a separate unit, as shown in Figure 3.2. To support the permute-and-arithmetic operations, a separate permutation operation was needed. The result of this permutation operation is written back to the SIMD register file, only to be read out in the next cycle for the arithmetic operation, thereby

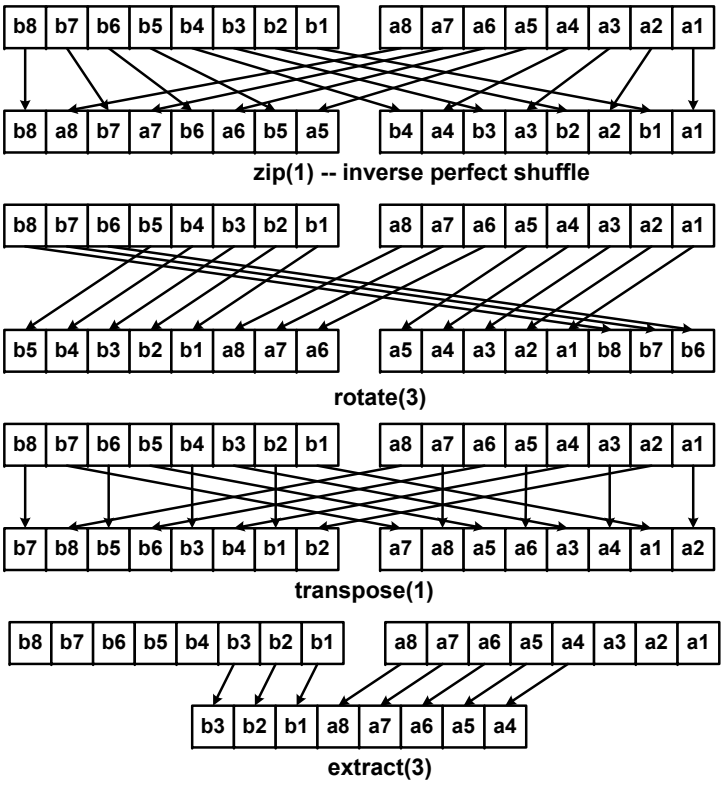


Figure 3.7: SSN shuffling patterns used for matrix transpose.

increasing register file access power in SODA.

The Ardbeg PE addresses this problem by including two shuffle networks. The 128-lane SSN is a separate unit that can support many different permutation patterns. In addition, a smaller 1024-bit 1-stage shuffle network is included in the same pipeline stage in front of the SIMD ALU. This 1-stage shuffle network only supports inverse perfect shuffle patterns between different groups of lanes. This shuffle pattern implements the various pair-wise butterfly operations shown in Figure 3.6. In the figure, the shuffle and add operations are performed in the same cycle. This shuffle network is used to accelerate FFT and various other algorithms that use butterfly-and-addition operations. Because these fused butterfly operations are the majority

of the permute-and-arithmetic patterns, Ardbeg is able to benefit from the best of both designs. A 2048-Point FFT is able to gain 25% speedup using fused butterfly operations.

SIMD Support for Interleaving. Interleavers are common in wireless protocols. They are used to protect the transmission against burst errors by rearranging the data sequence. Unlike most other DSP algorithms, there is no data processing or computations involved in interleaving; interleavers simply rearrange the data sequence in different patterns to account for varying types of transmission environments.

Interleaving is essentially a long vector permutation operation, where the vector width is far greater than the SIMD width. This is a challenge because the SODA and Ardbeg SSNs can only permute vector patterns of SIMD width. If we let N be the size of the vector, then a general purpose permutation algorithm would take $O(N)$ time. However, for certain permutation patterns, different types of SIMD shuffle patterns can be utilized to reduce the permutation latency. As mentioned in Section 3.2.1, the Ardbeg SSN supports a set of predefined permutation patterns for efficient implementation of certain interleaving patterns. For example, one commonly used pattern is the matrix transpose operation, where the input vector is organized as an $M \times N$ matrix, and the output vector is transposed into an $N \times M$ matrix. A $O(log(N))$ algorithm exists that uses the zip, transpose, extract, and rotate shuffling patterns [9] as shown in Figure 3.7. Using these predefined patterns, a 192 element vector can be transposed in just 37 cycles. This translates to an average speedup of 4x for interleaving kernels for Ardbeg in comparison to SODA.

	W-CDMA	802.11a	DVB-T, DVB-H
Throughput	Voice: 12Kbps Data: 384Kbps/2Mbps	24Mbps, 54Mbps	5Mbps, 15Mbps
Filtering	Complex FIR 65-taps	FIR 33-taps	FIR 16-taps
Modulation	Scrambler/Descrambler Spreader/Despreader Combiner	FFT/IFFT 64 points QAM/IQAM 64 points	FFT 2048 points Scrambler/Descrambler QAM/IQAM 4/16/64 points
Synchronization	Searcher	Interpolator	Equalizer Channel Est.
Error Correction	Interleaver Viterbi K=9 Turbo Decoder K=4	Interleaver Viterbi K=7	Bit Interleaver Viterbi K=7

Figure 3.8: DSP algorithms that are used in W-CDMA, 802.11a and DVB, DVB-H wireless protocols.

3.3 Results and Analysis

For the overall protocol performance evaluations, we have implemented three different wireless communication protocols that represent a wide spectrum of wireless communication applications. These are W-CDMA [36], 802.11a [37], and DVB-T/H [23][22]. W-CDMA is a widely used 3G cellular protocol. 802.11a is chosen to represent the workload of a typical Wi-Fi wireless protocol. DVB-H (Digital Video Broadcasting - Handheld) is a standard used for digital television broadcasting for handheld receivers and DVB-T (DVB - Terrestial) is used for stationary receivers. Beyond 3G, many of the protocols are OFDM based such as WiMAX. We analyzed DVB-H and 802.11a as representatives of OFDM-based systems. These protocols are chosen to stress the flexibility of the SODA and Ardbeg systems. Both SODA and Ardbeg are able to support real-time computations for these protocols.

The characteristics of these three protocols are listed in Figure 3.8. These protocols consist of the following four major algorithm categories: filtering, modulation, synchronization, and error correction. Filtering is used to suppress signals transmit-

ted outside of the allowed frequency band so that interference with other frequency bands is minimized. Modulation algorithms translate digital signals into analog wave patterns consisting of orthogonal signals. Synchronization algorithms synchronize the two communicating terminals to ensure lock-step communication between the sender and receiver. Error correction algorithms are used to recover data from noisy communication channels.

The RTL Verilog model of the SODA processor was synthesized in TSMC 180nm technology. The estimated power and area results for 90nm technology were calculated using a quadratic scaling factor based on Predictive Technology Model [40]. The Ardbeg processor was developed as part of the OptimoDE framework [19]. The architectural model was written in OptimoDE's hardware description language. A Verilog RTL model, a cycle-accurate simulator, and a compiler are generated by OptimoDE. The Ardbeg processor was synthesized in TSMC 90nm using Synopsys physical compiler to place and Cadence Encounter to route with clock tree insertion. Ardbeg's PE area is 75% larger than SODA's estimated 90nm PE area. The total system area is comparable between the two systems because SODA contains 4 PEs compared to 2 PEs in Ardbeg. Ardbeg was targeted for 350 MHz, while SODA for 400 MHz.

3.3.1 Wireless Protocol Results

Evaluation results show that an Ardbeg multicore system synthesized in 90nm technology is able to support 3G wireless processing within the 500 mW power budget of a mobile device [62]. Figure 3.9 shows the power consumption required to

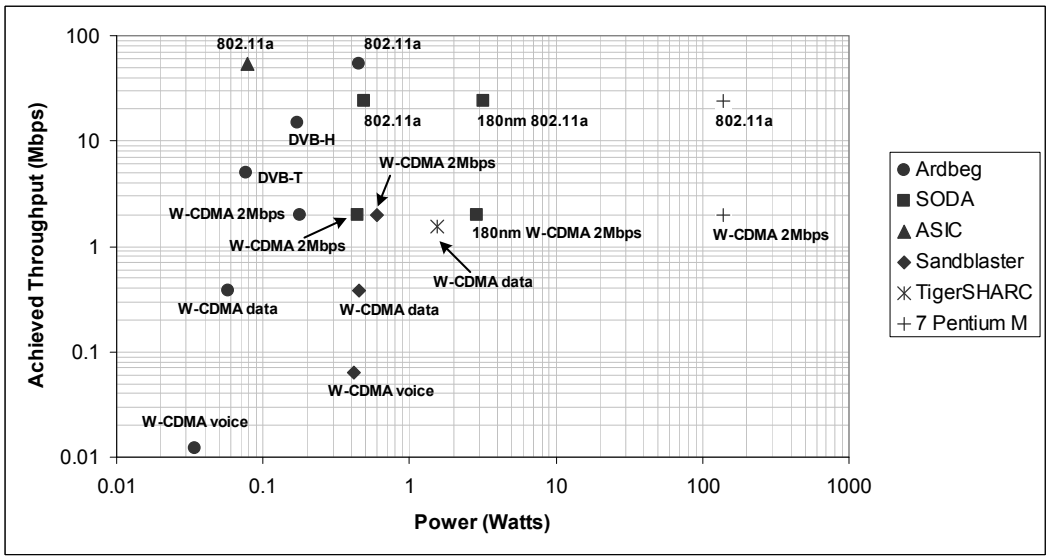


Figure 3.9: Throughput and power achieved for SODA and Ardbeg for W-CDMA, 802.11a and DVB-T/H. ASIC 802.11a, Pentium M, Sandblaster, and ADI TigerSharc results are also included for comparison purposes. Results are shown for processors implemented in 90nm, unless stated otherwise.

achieve the throughput requirement of W-CDMA, 802.11a, and DVB-T/H. The graph includes the numbers for the SODA and Ardbeg systems, as well as an ASIC implementation for 802.11a [78], Sandbridge's Sandblaster, Analog Devices TigerSHARC, and Pentium M implementations. Data for the other processors was estimated using datasheets and publications. General purpose processors, such as Pentium M, require a power consumption two orders of magnitude greater than the 500 mW power budget. On the other end of the spectrum, an ASIC solution is still 5x more power efficient than any SDR solution. Overall, Ardbeg is more power efficient than SODA for all three wireless protocols. Because Ardbeg is designed to handle high-throughput wireless protocols, its performance for low-throughput W-CDMA voice channels is not as efficient. This is because the available vector parallelism is lower and the processing

power of Ardbeg is not fully utilized. In these cases, the scalar datapath in Ardbeg would be utilized more frequently to save power. Both SODA and Ardbeg are very competitive compared to other SDR processors, including Sandbridge's Sandblaster and Analog Devices' TigerSHARC. The major sources of Ardbeg's efficiency are: the restricted LIW execution, application specific instruction set extensions, and larger shuffle network.

3.3.2 Wireless Algorithm Analysis

In this section, we present a performance analysis of the key DSP algorithms in each of the four algorithm categories. Details of the kernels can be found in [49]. The speedups are consolidated in Figure 3.10. The speedup analysis is further broken up into the Ardbeg architectural improvements that were highlighted in the Section 3.2. These improvements include: optimized SIMD ALU, wider single cycle SSN, and LIW execution. The OptimoDE framework used to design Ardbeg generates a compiler that performs optimizations like software pipelining and other compiler optimizations which we also report.

Filtering. Finite Impulse Response (FIR) filters are widely used in wireless communication protocols. Both the SODA and Ardbeg PEs can support the computation requirements of filters for real-time 3G wireless protocol processing. Figure 3.10 shows the Ardbeg PEs speedup over the SODA PE for various filter configurations. On average, Ardbeg achieved a 3.4x speedup over SODA.

Multiply-and-accumulate (MAC) operations are the central arithmetic operation

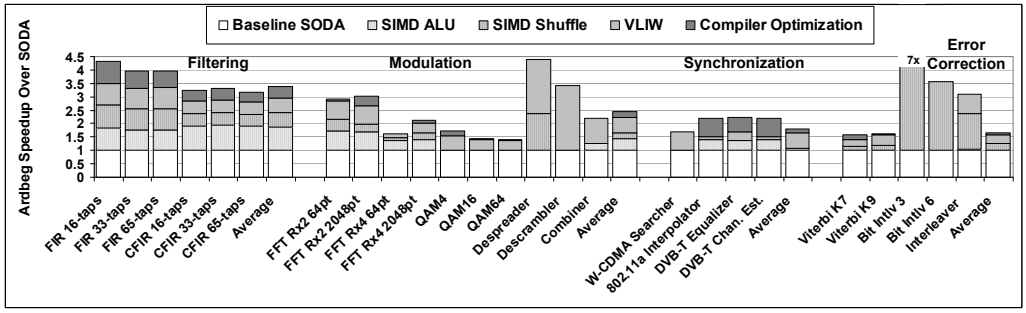


Figure 3.10: Ardbeg speedup over SODA for the key DSP algorithms used in our wireless protocol benchmarks. The speedup is broken down into the different architectural optimizations. These include optimized SIMD ALU, wider 1-cycle SIMD shuffle network, reduced SIMD memory latencies through LIW execution, and compiler optimizations with software pipelining.

for filtering. For complex filter arithmetics, multiplications are even more important as every complex multiplication requires four MAC operations. The SODA PE has a two cycle multiplier, whereas the Ardbeg PE has a single cycle multiplier. A significant portion of Ardbeg's speedup is due to the faster multiplier.

In this analysis, both SODA and Ardbeg implement a vectorized version that requires one 64-wide SIMD vector permutation operation for processing each sample point. The SODA PE only has a 32-wide SIMD permutation network, compared to Ardbeg's 64-wide network. The permutation operation takes 3 cycles on SODA, but only one cycle on Ardbeg. Because memory is accessed for each sample, LIW support on the Ardbeg PE is able to hide the multi-cycle memory latencies. Finally, software pipelining and other compiler optimizations help better utilize Ardbeg's LIW datapath.

Modulation. Fast Fourier Transform (FFT) is widely used in OFDM protocols like 802.11a/g and DVB-T. Figure 3.10 shows the Ardbeg PE speedup over the SODA

PE for various FFT configurations. On average, Ardbeg achieves a 2.5x speedup over SODA. Like the filters, there is about a 50% speedup attributed to single cycle multiplies. This speedup is less for a Radix-4 implementation because multiplications are reduced by 25%. Another 50-100% speedup is attributed to the fused operations. The butterfly operation is implemented efficiently by fusing multiplication with add or subtract operations. Another benefit is that Ardbeg allows specialized shuffle operations, followed by ALU operations to be computed in one cycle. Finally, the LIW scheduling provides the remaining speedup. Ardbeg can overlap the memory loads of the next butterfly with the current butterfly's shuffle operation.

Modulation in W-CDMA consists of three kernels: descrambler, despreader, and combiner. The despreader gains significant speedup (almost half) by utilizing Ardbeg's wide shuffle network. The descrambler implementation on Ardbeg is a direct translation of the SODA version. Ardbeg gains, because in every cycle, it can overlap the memory and ALU operations. The combiner, like the despreader and descrambler, benefits from the LIW scheduling as well as the one cycle multiplication. All three kernels benefit greatly from LIW scheduling because each iteration of the inner-loop is small and independent. This allows the overlap of memory loads and stores, shuffle operations, and ALU operations in the same cycle.

Synchronization. Synchronization in W-CDMA is accomplished by the searcher, which achieves almost 1.5x speedup on Ardbeg. The gain in performance is due to Ardbeg's pipelined memories and LIW scheduling. However, these gains are offset by performance loss due to its SIMD predicate support. The number of instructions

needed to calculate the predicate values on the Ardbeg PE is 4 cycles, whereas the SODA PE can perform the same task in 2 cycles. This is because SODA's predicate values are stored in the SIMD register file, whereas Ardbeg's predicate values are stored in a dedicated register file. Although Ardbeg's dedicated register file is able to compute different predicate patterns more quickly, it takes longer to load the predicate values into the SIMD datapath. Because all of searcher's predicate patterns can be pre-computed, SODA's faster predicate read latency proves to be more beneficial. This accounts for a 20% performance difference. The major benefit of Ardbeg's LIW scheduling is hiding the memory's multi-cycle access latencies. Because half of every loop iteration can be overlapped, the Ardbeg searcher still results in almost 2X speedup despite its inefficient predication support.

802.11a interpolator, DVB-T equalizer, and DVB-T channel estimation are all similar to the FIR operations, and their speedup rationales are similar to those of the FIR. The only difference is that these algorithms have intra-iteration data dependencies that cannot exploit the LIW datapath. Software pipelining is beneficial by scheduling different loop iterations onto the LIW datapath.

Error Correction. There are two commonly used error correction algorithms in wireless communication – Viterbi and Turbo decoding. As mentioned in the previous section, the Turbo decoder in Ardbeg is offloaded to an accelerator. However, the Viterbi decoder is still implemented by the Ardbeg PE. As shown in Figure 3.10, Ardbeg's Viterbi implementation has a speedup of only 1.2x to 1.6x compared to SODA. The small speedup is because the Viterbi computation does not have multiplication

operations, so the optimized SIMD ALU does not help. In addition, there are data dependencies between consecutive loop iterations, so software pipelining techniques do not help. The majority of the speedup comes from hiding the memory access latency through LIW execution on the SIMD pipeline.

Interleavers are widely used in many wireless protocols. As mentioned in the last section, a few SIMD shuffle patterns are added to accelerate these algorithms. As shown in Figure 3.10, the Ardbeg interleaver implementations gain a significant speedup, up to 7x speedup over SODA. The speedup is solely due to the Ardbeg's SSN. Because the majority of the interleaver instructions are SIMD permutation operations, Ardbeg's single cycle 64-wide SSN has a significant advantage over SODA's multi-cycle 32-wide SSN.

3.4 Conclusion

Software defined radio promises to revolutionize the wireless communication industry by delivering a low-cost multi-mode baseband processing solution. Previous work has proposed SODA, a multi-core wide SIMD DSP architecture. Ardbeg is a commercial prototype based on SODA designed by ARM Ltd. Aspects of the SODA design are kept intact, such as the wide 512-bit SIMD datapath and the coupled scalar and SIMD datapath. Application-specific design trade-offs are made to achieve higher computational efficiency while maintaining enough flexibility to support multiple protocols. The evolution of SODA to Ardbeg happened due to optimization in three main areas: wide SIMD design, LIW support for wide SIMD, and algorithm specific hard-

ware acceleration. The results show that Ardbeg's architectural optimizations achieve between 1.5-7x speedup over SODA across multiple wireless algorithms.

CHAPTER 4

The Next Generation Challenge for Software Defined Radio

Wireless communication has grown dramatically over the years. Accessing the web, downloading video, and listening to music is a growing demand with wireless users. Third generation wireless (3G) technologies have been able to provide people with access to these services. With the number of users increasing and the demand for higher quality content, the bandwidth needed exceeds what 3G can provide. Fourth generation wireless (4G) technology has been proposed by the International Telecommunications Union (ITU) to increase the bandwidth to maximum data rates of 100 Mbps for high mobility situations and 1 Gbps for stationary and low mobility situations like internet hot spots. With this increase in bandwidth there will also be an increase in the number of computations needed to process this standard on software defined radio (SDR) systems.

This chapter outlines the 4G physical layer. Previous works have characterized

the computational requirements of 3G [49]. This is one of the first characterizations of 4G as a whole. The aim is to show the requirements that are needed to process the 4G physical layer and also to identify computational patterns that might suggest an architecture that can support 4G.

The 4G system we will study is based on orthogonal frequency division multiplexing (OFDM) that uses a 1024-point FFT/IFFT, a 4x4 16QAM multiple input multiple output (MIMO) antenna system, and a low density parity (LDPC) encoder and decoder. Detailed analysis of the major algorithms that make up these components and their computational characteristics show the following repeated computational pattern: load data from a memory element (initially this is the received data), permuting that data, performing one or two ALU operations, and storing the processed data back to memory. These patterns are similar to those found in 3G kernels. The architectures that are designed to support them, such as SODA, will not be able to meet the 4G requirements through technology scaling alone. As we will show, other techniques will have to be enlisted such as wider SIMD engines, special purpose functional units, and special memory systems.

This chapter is organized as follows. In the next section, we begin by presenting a simplified 4G system and by describing some of major kernels: an OFDM modulator/demodulator, a MIMO modulator/demodulator, and a channel decoder for LDPC. In section 3, we give a brief overview of the SODA architecture and use it as a baseline to identify the dominate workload profiles and common computational patterns of the kernels. In section 4, we present programmable hardware support for

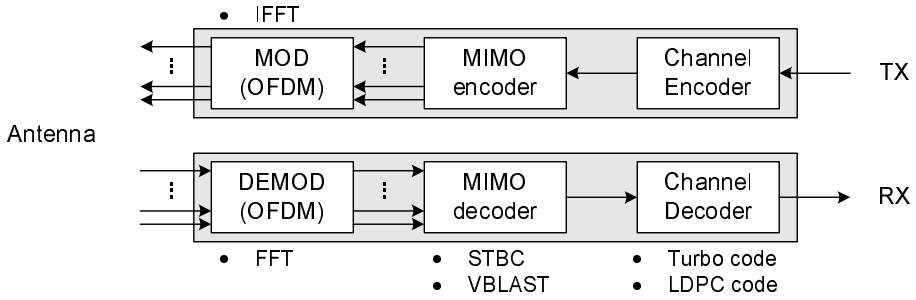


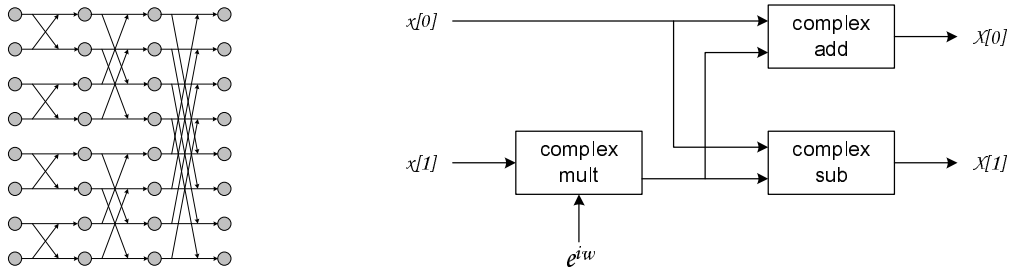
Figure 4.1: The physical layer for a 4G terminal

implementing these kernels efficiently to meet the high throughput required for 4G. The summary and concluding remarks are given in section 5.

4.1 4G Physical Layer

Figure 4.1 shows a 4G wireless terminal. Like other wireless communication system, its major blocks are a channel encoder/decoder and a modulator/demodulator. The role of the channel encoder is forward error correction that enables receivers to correct errors without retransmission. Modulation maps input data sequence onto signal waveforms which are specifically designed for the wireless channel. Demodulation estimates the transmitted data sequence from the received waveform, which have been corrupted by noise and interference when they traversed the wireless channel.

In order to satisfy the gigabit level throughput requirement, 4G systems employ three techniques not found together in 3G: 1) orthogonal frequency division multiple access (OFDMA); 2) MIMO to support multiple antennas; and 3) LDPC codes for the channel encoder/decoder.



(a) Data movement patterns

(b) Computation patterns

Figure 4.2: The data movement of an 8 point FFT and the computations in a 2 point FFT

4.1.1 OFDMA

OFDMA is a modulation scheme which transmits input signals over multiple narrow sub-channels. Both modulation and demodulation in OFDMA systems can be implemented with fast fourier transforms (FFT). Although additional synchronization procedures are required in OFDMA receivers, we can ignore them because their contribution is small.

4.1.1.1 FFT

As shown in Figure 4.1, the transmitter uses an inverse FFT (IFFT) for modulation and the receiver uses an FFT for demodulation. Because FFT and IFFT are almost identical, we will just analyze the FFT.

The FFT operation consists of a data movement followed by multiplication and addition on a complex number. If we assume an N point FFT, it consists of $\log_2 N$ stages. As an example, Figure 4.2(a) shows the data movement pattern of an 8 point FFT. It consists of 3 stages. Each stage shows a different but regular data

movement pattern. The operation of each stage can be divided into several 2 point FFT operation as depicted in Figure 4.2(b).

The FFT allows wide data level parallelism because all 2 point FFT operations required for proceeding from one stage to the next can be done in parallel. It is important to exploit this type of data level parallelism to meet power and performance requirements of 4G system, because the FFT width of 4G systems can be as large as 2048.

4.1.2 MIMO

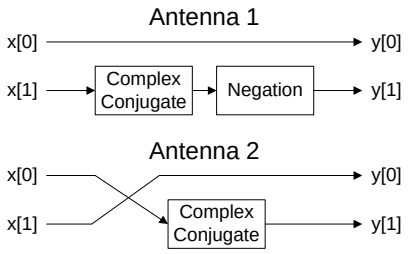
MIMO is a technique that uses multiple antennas both for the transmission and reception. It can be used for two purposes: signal quality enhancement by transmitting identical signal through multiple antennas and channel capacity enhancement by transmitting different signals on multiple antennas. Space time block codes (STBC) is a popular MIMO technique for the signal quality enhancement and the vertical Bell Laboratories layered space-time (V-BLAST) technique is popular for channel capacity enhancement.

4.1.2.1 STBC

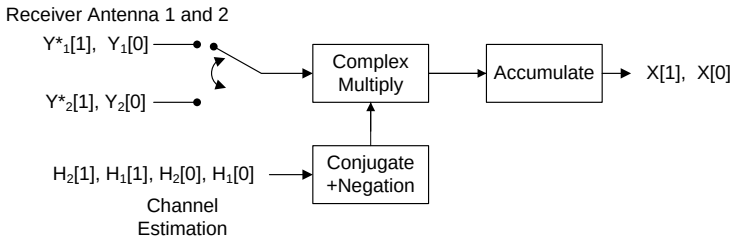
This is used to increase the signal quality by transmitting the same signal multiple times through different antennas. Signal quality is increased by receiving those redundant copies of the same signal and using the information from each receiver to optimally combine them to produce a better signal. The implementation we used is

Time slot, T	Antenna	
	Tx 1	Tx 2
1	x_1	x_2
2	$-x_2^*$	x_1^*

(a) Transmission Matrix—the * indicates complex conjugate.



(b) Computation patterns of an STBC encoder



(c) Computation pattern of an STBC decoder

Figure 4.3: Transmission code matrix and computation patterns of the Alamouti 2x2 STBC

based on Alamouti's 2x2 scheme [5], which uses 2 transmit and 2 receive antennas.

STBC Encoder The encoder orders and transmits data based on the transmission matrix shown in figure 4.3(a). The operation consists of transmitting two different symbols at the first time instance, then transmitting the conjugate of the same two symbols with antennas switched (see the matrix in figure 4.3(a)). Figure 4.3(b) shows the computation needed to perform this operation. First the data is sent to each modulator and then the conjugate and negation are performed. This corresponds to a simple predication operation to obtain the real and imaginary values. This is highly parallelizable, and a 1024 point FFT could be run in parallel on a 1024 wide SIMD (Single Instruction, Multiple Data) processor.

STBC Decoder The decoder takes the transmitted data from both time instances and combines them together to create the original two symbols. The decoder operation consists of performing complex multiplications between each of the received signals and the channel estimation for each antenna and then summing the values. Figure 4.3(c) shows this operation pattern. Calculating both symbols can be done at the same time with the least amount of data movement. Once again, because subcarriers are totally independent, this algorithm is highly data parallel, and a 1024 point FFT could be run in parallel on a 1024 wide SIMD.

4.1.2.2 V-BLAST

This is one of the spatial multiplexing schemes that improves multiplexing gain by transmitting independent data streams over different antennas. This technique combines multipath signals to obtains higher data rate compared to STBC. The V-BLAST algorithm that was used was based on work from [32] which reduces the computational complexity of V-BLAST.

V-BLAST encoder The V-BLAST encoder is similar to the STBC encoder. It also uses a transmission matrix to decide ordering, conjugating and negating for a block of data. Therefore, the pattern of required operations is: load the real and imaginary received data, permute the data based on the transmission matrix, then negate and store the result before sending it to the OFDM modulators associated with the multiple antennas. The computation pattern would be the same as figure 4.3(b) except the matrix for V-BLAST is 4x4.

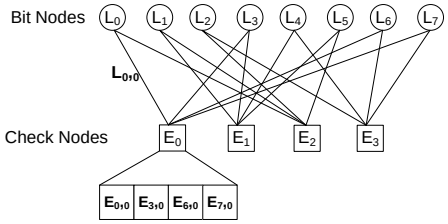
V-BLAST decoder The decoding process of V-BLAST consists of two major steps: channel estimation and signal detection. The channel matrix is estimated based on pre-defined training symbols. The operations for channel estimation are relatively simple with shift and sign-change operations. Once the channel matrix has been estimated, the detection order is determined. The detection order is based on signal strength found among all the signals received. The strongest signal is selected and extracted from the received signal. This process is repeated for the remaining signals. This process is iterative and is referred to as successive interference cancelation. The signal detecting operations can be described by the following steps: 1) load the received signal; 2) vector multiplication for obtaining the stongest signal; 3) vector multiplication and subtraction for canceling the strongest signal; and 4) repeat.

4.1.3 Channel Encoder/Decoder

4G systems are expected to use both Turbo codes and LDPC codes as channel coding schemes. We limit our discussion to the characteristics of the LDPC codes in this section, because Turbo codes have already been used in 3G systems and their characteristics have been well documented elsewhere [55] [51].

4.1.3.1 LDPC

Figure 4.1 shows the channel encoder and decoder for LDPC. It is currently used in IEEE 802.16e and 802.11n. The encoder for LDPC is trivial in the sense that for each LDPC code there are a set of codewords available. For different data rates there



(a) Graphical representation of LDPC code

1. Initialization $E_{n,m} = 0, L_{n,m} = I_n$

2. Bit Node Operation $L_{n,m} = L_n - E_{n,m}$

3. Check Node Operation $E_{n,m}^{New} = -\prod_{n' \in N(m) \setminus \{n\}} sign(L_{n',m}) \cdot \min_{n' \in N(m) \setminus \{n\}} |L_{n',m}|$

4. Bit Update $L_n^{New} = L_{n,m} + E_{n,m}^{New}$

(b) LDPC decoding in 4 steps

Figure 4.4: LDPC graphical representation and decoding operations

are different number of codewords. In order to transmit data a codeword is picked and sent through the transmitter. Because the operation is fairly simple we will only discuss the LDPC decoding operation.

Decoding is based on an architecturally aware design for LDPC codes given in [94]. The code rates and the block sizes used were based on the IEEE 802.16e standard [2] and picked in order to meet the 100Mbps and 1Gbps target data rate.

The graphical representation of LDPC is shown in figure 5.8(a). The check nodes represents the number of rows in the parity check code and the bit nodes represent the number of columns. The edges connecting the check nodes and bit nodes are the 1's in the parity check code matrix—all other values are 0. The LDPC decoding operation is broken down into 4 stages as shown in figure 5.8(b). These four stages are the Initialization, Bit Node, Check Node, and Bit Update operation. This implementation is based on the Min-Sum algorithm.

The major operation in the implementation of LDPC is to first load the L_n and $E_{n,m}$ values. The next step is to permute the L_n's so they align with the $E_{n,m}$ values. Then it is possible to compute $L_{n,m}$ by performing an subtraction. Finally we do a

compare and select to find the first and second minimum. This operation performs the Bit Node operation and the Check Node operation. The Bit Update operation first loads the L_n, then it does a comparison to determine whether the location of the minimum $E_{n,m}$ is the same as the L_n position. If it is not, then it will use the first minimum as the minimum $E_{n,m}$. Otherwise it will use the second minimum. Finally, it adds the new $E_{n,m}$ value to L_n , updating the L_n value. This operation is done for each block row of the code. After all block rows have been updated an iteration is complete.

LDPC exhibits considerable data level parallelism. For each $E_{n,m}$ we process one L_n at a time. Potentially we can do an N SIMD wide operation for the Bit Node and Check Node operation where N is the number of Check Nodes.

4.2 Computational Analysis

4.2.1 Baseline Architecture

In order to calculate the workload characteristic we took an existing architecture for 3G and programmed the 4G algorithms onto it. The architecture we used is SODA [54]. The SODA multiprocessor architecture is shown in Figure 4.5. It consists of multiple processing elements (PEs), a scalar control processor, and global scratchpad memory, all connected through a shared bus. Each SODA PE consists of 5 major components: 1) an SIMD pipeline for supporting vector operations; 2) a scalar pipeline for sequential operations; 3) two local scratchpad memories for the SIMD

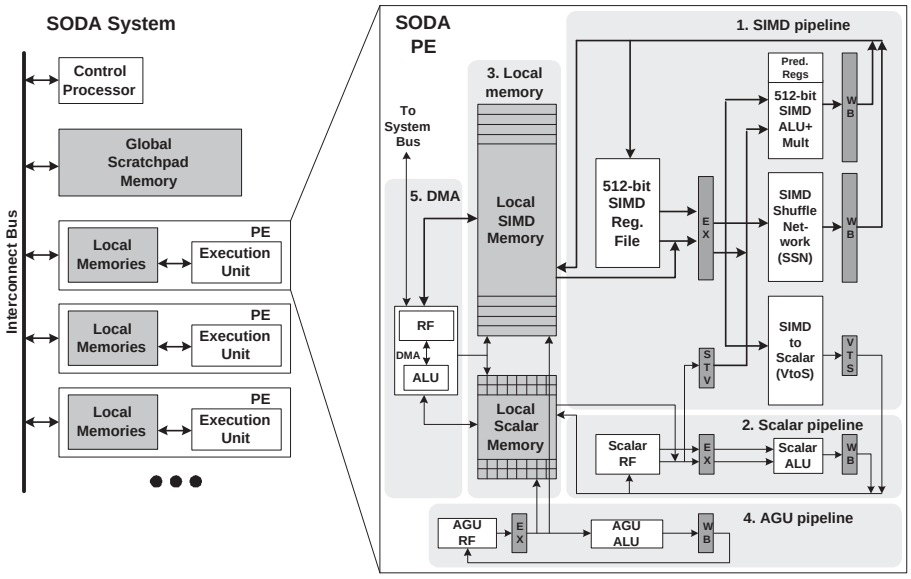


Figure 4.5: SODA Architecture for SDR

pipeline and the scalar pipeline; 4) an AGU (address generation unit) pipeline for providing the addresses for local memory access; and 5) a programmable DMA unit to transfer data between memories and interface with the outside system. The SIMD pipeline, scalar pipeline and the AGU pipeline execute in VLIW-styled lock-step, controlled with one program counter.

The SIMD pipeline consists of a 32-way 16-bit datapath, with 32 arithmetic units working in lock-step. It is designed to handle computationally intensive DSP algorithms. Each datapath includes a 2 read-port, 1 write-port 16 entry register file, and one 16-bit ALU with multiplier. The multiplier takes two execution cycles when running at the targeted 400MHZ. Intra-processor data movements are supported through the SSN (SIMD Shuffle Network). The SIMD pipeline can also take one of its source operands from the scalar pipeline. There are also several SIMD reduction opera-

Algorithm Name	100Mbps Data Rate	1Gbps Data Rate
	MCycle/s	MCycle/s
FFT	360	360
IFFT	360	360
STBC	240	-
V-BLAST	-	1900
LDPC	7700	18500

Table 4.1: Cycle Count of Major 4G Kernels on SODA

tions that are supported, including vector summation, finding the minimum and the maximum.

4.2.2 Workload Profile

The breakdown of the major algorithms in our 4G protocol is listed in table 4.1. This analysis is based on the algorithms as they would be programmed for the SODA architecture. We calculated the number of cycles per second needed to support the data rate shown. Referring back to the system diagram in figure 4.1: for the 100Mbps rate we assume the STBC algorithm based on the Alamouti scheme which uses 2 transmit and 2 receive paths; and for the 1Gbps rate we assume a 4 transmitter and 4 receiver multiplexing diversity scheme based on V-BLAST. In the STBC algorithm we require that each receiver performs one FFT but only one STBC decoder for all the receivers. Each receiver is independent of the other's operation so both FFTs can run on separate processors. For the multiplexing diversity scheme each receiver processes separate data. That means that for the 1Gbps data rate we have 4 independent streams of 250Mbps being processed, but still only one V-BLAST decoder has to be

Algorithm Name	Load	Permute	First ALU Op	Secondary Op	Store
FFT	X	X	X		X
IFFT	X	X	X		X
STBC	X	X	X	X	X
V-BLAST	X	X	X	X	X
LDPC	X	X	X	X	X

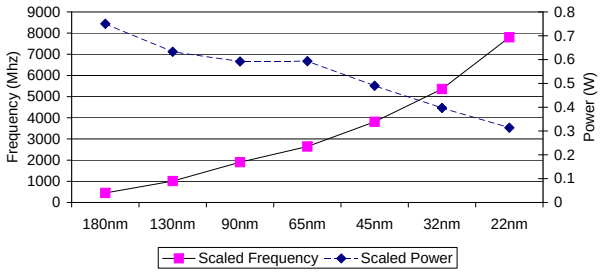
Table 4.2: Computational Pattern of 4G algorithms

performed.

From the table we can see that the channel coding algorithm is the dominate workload. Assuming we were processing each multiplexing diversity stream on one processor it would require us to run SODA at more than 10GHz for the 100Mbps case and almost 30Ghz for the 1Gbps case. An alternative approach would be to have one processor for each kernel. This would mean we would need the maximum frequency of SODA to be 8GHZ and 20Ghz for the 100Mbps and 1Gbps cases respectively. Though it may seem that the FFT, IFFT, STBC and V-BLAST algorithms are somewhat negligible compared to the channel coding we should not forget that the workload of channel coding is related to the data rate. As the data rate decreases the workload of the channel coding also decreases but the other kernels do not. At low data rates the other algorithms become comparable in cycle count and the optimization for these algorithms will then be key to an efficient design.

4.2.3 Computational Patterns

Analysis of each algorithm reveals that there is a consistent computational pattern. Table 4.2 shows each kernel's inner loop broken down into simpler operations.



(a) Technology Scaled SODA

Technology Node	180nm	130nm	90nm	65nm	45nm	32nm	22nm
Vdd (V)	1.8	1.3	1.1	1.1	1	0.9	0.8

(b) Vdd Voltage Scaling

Figure 4.6: Technology scaling from 180nm to 22nm with respect to Frequency, Power, Vdd on SODA for 4G

The pattern of loading the received data, permuting the data, performing an ALU operation, then a secondary ALU operation and finally storing the result back is very common to all the algorithms. These patterns make up the majority of the cycle time and are repeated for all the data being streamed in.

Another point to the note is that the data is streamed through the operations. Once the data is consumed we do not refer back to it until the next iteration, or a summation, or a max/min is performed. Often sequences of operations are performed before having to store results. This suggests that there is little temporal locality of the data. Once the data is consumed we do not expect it to be used again. This is true for most DSP applications [76].

Data alignment is a key problem in each of the algorithms. Each algorithm has to align data before any computation can be performed. In the SODA architecture we use the SSN which includes a perfect shuffle network to perform this operation.

4.3 Architectural Implications

The frequency that the SODA processor would need to operate at in order to processes 4G was estimated at 20Ghz. Based on data from the ITRS roadmap [1] and [77] we show in figure 4.6(a) that technology scaling will still leave us a factor of 3x behind in frequency for a given power budget at 22nm. The power budget was set at $3W/mm^2$ combined for all cores. It is set by limitations of cooling and packaging based on data from ITRS. At 22nm this would be around 1W. Until recently technology scaling has also been accompanied by a scaling in supply voltage. As we get to smaller technology nodes this is no longer the case and the supply voltage is not scalling as much [58]. Figure 4.6(b) shows the decrease in supply voltage with technology node. The table shows that power consumption will be decreasing more slowly and also that frequency scaling and voltage scaling will be less effective in terms of power reduction.

From the figure we see that at 22nm we could support the 100Mbps data rate on SODA and still meet the power requirement. The 100Mbps solution would require 2 SODA processors running at 10Ghz. If our projections are correct, this is a possible future solution, because the 22nm technology node is expected to be in production in 2011 [16] which coincides with when ITU expects 4G networks to be deployed. This still does not leave us with any solution for the 1Gbps data rate. However, there are many features of the algorithms which we can exploit architecturally to help us reach the goal of 1Gbps and still retain the flexibility of a programmable SDR processor.

4.3.1 Multi-Processor

Most of the 4G algorithms can be divided onto multiple processors especially for FFT, and STBC, and even LDPC. The workload can be divided evenly among the processors. However, as we subdivide the algorithms across processes we get an increase in data communication. Although each stage of an algorithm is highly data parallel, stages requires data movement between different subcarriers in the FFT and between different check nodes in the LDPC. As we subdivide the algorithms, communication will increase, but, because the operations of each stage are streamed, we may be able to hide the latency of this communication under the computations itself. This would require an efficient routing and interconnect network and also scheduling that would be able to meet the constraints of data communication when multiple processors are used.

By dividing the workload across multiple processors we would be able to meet the frequency target for the 4G 1Gbps workload but we would still be 3x off the power budget. Multicore designs themselves cannot solve the problem of meeting the 4G requirement.

4.3.2 Wider SIMD

Increasing the SIMD width of the processors takes advantage of the highly data parallel nature of the algorithms. Based on historical transistor growth, at the 22nm node we can expect to grow from a 32 wide SIMD to a 2048 wide SIMD machine. This assumes a fixed area constraint. This increase in width would allow us to reduce

the cycle count to compute any size FFT as long as N is greater than or equal to the SIMD width. For FFT, the data movement can be accomplished by the SSN shuffle network.

For LDPC this increase in SIMD would also be beneficial because we can process more parity check nodes for LDPC at once. LDPC though would not gain the same data movement advantages as FFT, because it needs to align the check nodes and the bit nodes. However, this would not increase the amount of data movement dramatically.

STBC would also benefit, because it would be possible to process more subcarriers at one time. Because there is little data movement within the STBC we can expect gains equal to the increase in width.

4.3.3 Special Purpose Functional Units

Currently in SODA the operations are RISC like in that after every instruction is simple and then writes back to the register file. This can be costly in terms of power and latency, because, as we stated earlier, the algorithms are streaming in nature. Writing back the data may not be very efficient. This suggests that functional units that chain operations will be beneficial not only in performance but also power. There has been work [41] that shows that using special functional units to streamline common operational patterns may not only increase performance but also will be more area and energy-efficient.

LDPC would also benefit from having special minimum and maximum registers

embedded into the ALU. For each row operation of the parity check matrix that is performed the result will be compared with the current state of the register and swapped if the condition is met. In comparison with SODA, by implementing this special functional unit, LDPC can be reduce in cycle count by about 30 percent.

4.3.4 Memory System

Most of the algorithms like LDPC, FFT and STBC all treat each row of the SIMD as independent. The data is loaded from memory then permuted and stored back. There is no instance in those algorithms where two rows have to access the same data at the same time. This suggests that the memory system does not have to be a large global shared memory. Instead it can be divided into banks. Banking the memory as much as possible will reduce the cost of reading and writing data into a large global memory. Banking will allow us to reduce the size of each memory, increase the speed, and lower power of the memory system. In algorithms like LDPC, which may need block sizes that are larger than currently used, we would be able to efficiently scale the size of the memories too.

Algorithms would also benefit from a smarter memory systems that support flexible gather/scatter accesses. Currently many cycles are wasted in LDPC aligning the check nodes and bit nodes. V-BLAST would also benefit, because the algorithm has to read and write back data in changing orders.

4.4 Summary

The power/performance requirements for 4G presents a significant challenge for computer architects, especially if some degree of programmability is to be retained. Currently technology is not capable of processing a 4G system on a single processor. In this section we have analyzed a 4G system in the context of the SODA architecture and have shown that 3G solutions cannot meet the performance of 4G even if technology scaling is taken into account. We have presented architectural options that can improve the performance and reduce the power consumption of 4G solutions. We have argued that one solution to the power/performance challenge for 4G will increase the number of cores, and that each core will include a very wide SIMD processor with special purpose function units and highly banked memories.

CHAPTER 5

Algorithmic Analysis and Architectural Implications of the Next Generation Software Defined Radio

Baseband signal processing for mobile terminals has been a computing challenge for computer architects. Several architectures have been successful at achieving the super computer like workloads of 3G wireless systems while maintaining the mobile device power budget. Examples are Sandbridge's Sandblaster [81] and SODA [54]. But these architectures are not able to handle the almost 10-1000x increase in throughput required for 4G systems. We need even more efficient designs to complete these tasks. We need more powerful processors that consume less energy because device scaling is delivering less performance improvements and not reducing power consumption for existing designs.

In the past ITRS [1] had suggested that in future generations of process technologies, we would still be able to scale frequencies higher. ITRS has been very optimistic

with its predictions and manufacturers in the past have been able to meet these targets through material and process improvements. This continuation of process technology improvement is not sustainable anymore and current data shows that frequency and power are reaching a plateau [34]. The only benefit for changing technologies would be reduction in transistor size allowing us to pack more logic onto the same size die.

This slowdown in performance gain requires us to extract more computation with better architectural design. One way to extract more computational performance is by increasing the width of the datapath and exploiting more data-level parallelism (DLP) through the use of SIMD. Not only does this support increased performance, we can do it with little or no increase in energy. Though wider SIMD widths will consume more energy per operation, we can take advantage of the DLP and reduce the amount of control code and memory accesses thus reducing the total number of cycles and total energy overall. In our study, we analyze the effect that SIMD width has on computational performance and energy for 4G wireless systems. We will show the maximum possible DLP that can be extracted from each kernel and the percent of the algorithm which will benefit from exploiting the DLP.

Another way to extract more computational efficiency is by reducing the total energy consumed. If performance stays the same but energy is reduced the total power is also reduced. We found that register file (RF) access energy accounts for a large percentage of the total energy in many processors. In order to reduce this large percentage, we analyzed the RF access patterns within the kernels and found that many of the kernels have instructions which produce values which are consumed

by the next instruction and never used again. By storing these values into a small temporary RF or forwarding the data without writing the value into the RF, we can save the wasted energy of writing and reading these values from the main RF.

This chapter is organized as follows. In Section 5.1, we present the SDR architecture used for the SIMD width and RF access study. In section 5.2, we present a simplified 4G system and describe some of the major kernels: an OFDM demodulator /modulator, a MIMO encoder/decoder, and a channel decoder. In Section 5.4, we analyze the kernels and show their potential DLP and also the instruction breakdown of the algorithm. We also present the effects that varying SIMD width has on the computation and energy efficiency. In Section 5.5, we analyze the RF access patterns in the kernels and we show the amount of energy saved by not writing back unneeded values into the main RF. The summary and concluding remarks are given in Section 5.6.

5.1 SDR Processors

Most current processor solutions for SDR utilize SIMD to exploit the large amounts of DLP. These include Infineon's MuSIC [13], Analog Device's TigerSHARC [25], Icera's DXP [42], Phillips's EVP [87], and Sandbridges Sandblaster [27]. Though all these processors are SIMD processors, the widths for each are different. Recent work by Naessens[60], suggests building wide SIMD processors of up to 96 lanes in width to support the different error correction algorithms within SDR. For our work, we used the SODA processor [54] to explore the scalability of SIMD width and to analyze the

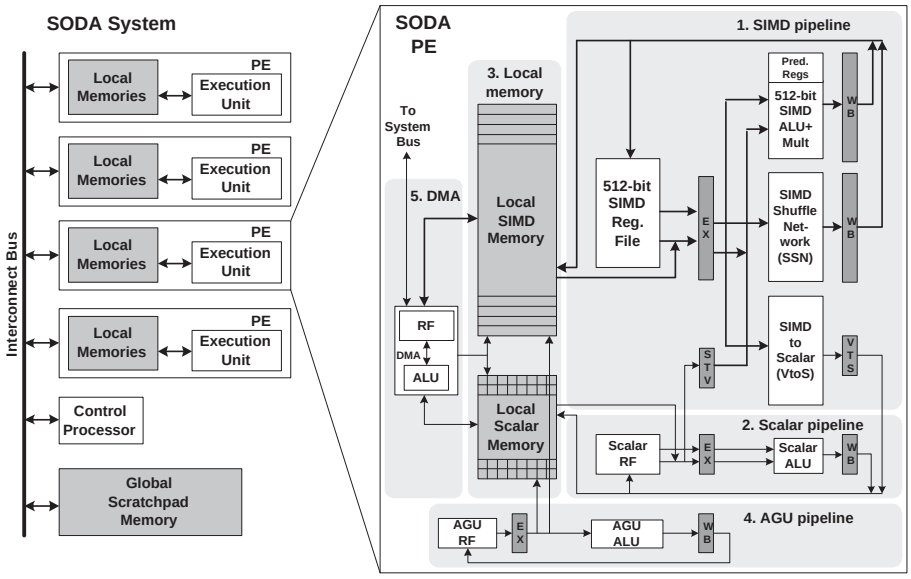


Figure 5.1: SODA Architecture for SDR. The system consists of 4 data processing elements (PEs), 1 control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a 32–wide 16–bit SIMD pipeline, a 16–bit scalar pipeline, two local scratchpad memories, an Address–Generation–Unit(AGU) for calculating memory addresses, and a Direct–Memory–Access (DMA) unit for inter-processor data transfer.

potential for reduced register accesses.

A block diagram of the architecture is shown in figure 5.1. SODA is a control-data decoupled multi-core architecture. The SODA architecture consists of processing elements which uses a wide 512-bit SIMD unit that is capable of operating on 32 16-bit elements concurrently. SODA also has a non-uniform memory architecture, with local memories on the processing elements and a shared global memory. In SODA, RF accesses are the same as typical DSPs where each instruction accesses one or two registers from the RF and then write the results back into the RF. Even if the next instruction accesses the register value being written to, the data is written into the RF.

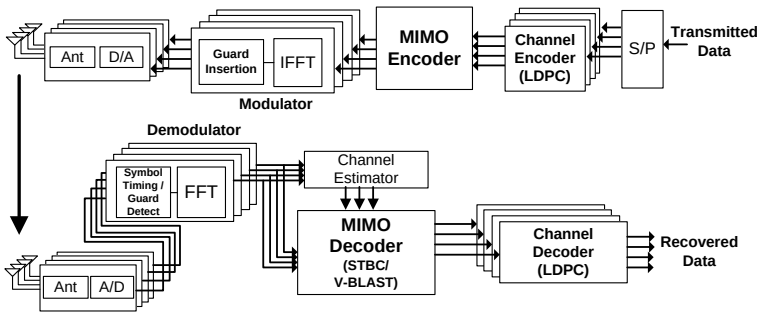


Figure 5.2: Block Diagram Overview of a 4G System.

5.2 4G Wireless Kernels

Though there is no standard yet for 4G, Figure 5.2 shows a high level block diagram of the physical layer of a NTT DoCoMo test 4G wireless system setup [84]. The major components of the physical layer consists of 3 major blocks: de/modulator, MIMO encoder/decoder, and the channel encoder/decoder. These blocks compose the majority of the total computation in 4G. The role of the modulator is to map data sequences into amplitudes and phases which then are converted to the time domain and transmitted. The demodulator performs the operations in reverse order to reconstruct the original data sequences. This is typically done by the Fast Fourier Transform (FFT) algorithm.

The Multiple Input Multiple Output (MIMO) encoder multiplexes many data signals over multiple antennae. The MIMO decoder receives all the signals from the antennae and then tries to either decode all the streams for increased data rates or combine all the signals in order to increase the signal strength. The algorithm used to increase data rate is Vertical Bell Laboratories Layered Space-Time (V-BLAST)

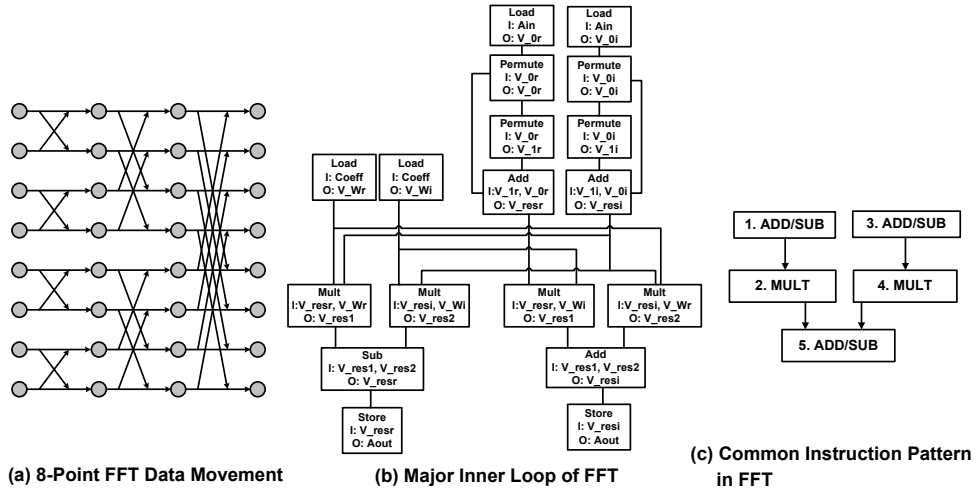


Figure 5.3: For the 8-point FFT, the butterfly pattern is the cross between two different elements in the vector. The major inner loop of FFT performs the operation of the butterfly across a SIMD. The nodes represent SODA instructions and the edges represent data dependencies. The instruction pattern represents the actual butterfly operation that is performed in each lane.

and the algorithm used to increase the signal quality is Space Time Block Codes (STBC). Finally the channel encoder and decoder performs forward error correction that enables receivers to correct errors in the data sequence without retransmission. There are many FEC algorithms which are used in wireless systems but LDPC is the most computationally intensive kernel and used for the highest data rates. LDPC has also been proposed in many standards like TGnSync and WWise [52] for IEEE 802.11n, which leads us to believe it may be used in 4G systems as well.

5.2.1 Fast Fourier Transforms (FFT and IFFT)

The transmission path uses an inverse FFT (IFFT) for modulation and the receiver uses an FFT for demodulation. We will only discuss the algorithm for FFT because

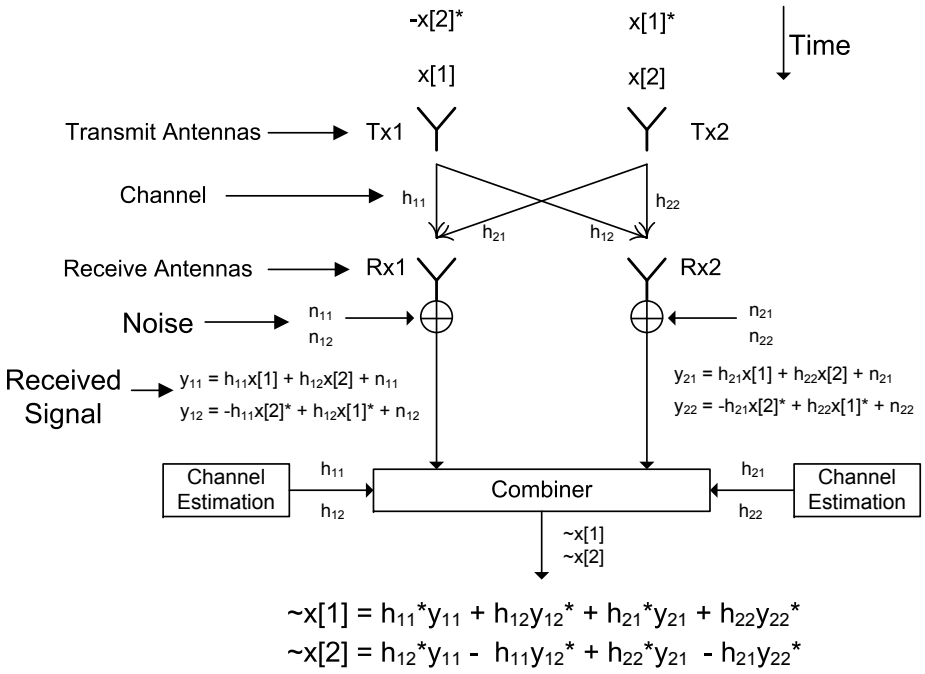


Figure 5.4: STBC general operation of Alamouti scheme.

IFFT is almost identical. The FFT operation consists of a data movement operation followed by a multiplication and addition on a complex number. An N-point radix-2 decimation in frequency (DIF) FFT consists of $\log_2 N$ stages. Between each stage, the N points of data are shuffled in a butterfly pattern. As an example, Figure 5.3a shows the data movement pattern of an 8-point FFT which consists of three stages. Each stage shows a different but regular data movement pattern. The operation of each stage can be divided into several 2-point FFT operations. We used a radix-2 FFT because other FFT implementations have more complex shuffle patterns which require more cycles to implement even though the arithmetic may be simpler (as is the case with radix-4 FFT).

Figure 5.3b shows the instruction execution flow (IEF) on SODA for one of the

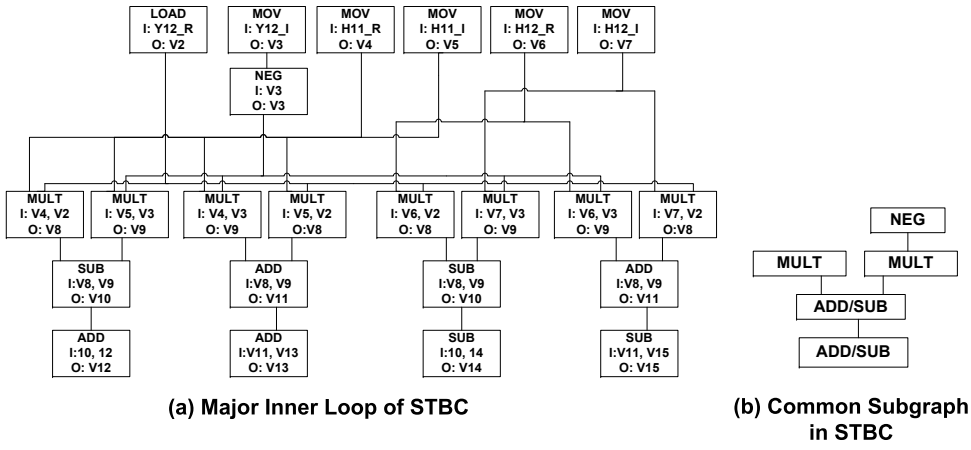


Figure 5.5: The inner loop of STBC shown calculates the values x[1] and x[2] of Alamouti Scheme. The common instruction pattern as shown is the majority of the inner loop code.

major inner loops of FFT. Each node represents the SODA instruction and the edges represent the data dependency. These instructions implement the butterfly operation on the SODA processor [54]. There are many edges within the IEF where there is only one producer immediately followed by the consumer. This means that the values produced by those instructions will only be used once and then never referenced again. From this IEF we can see there is a common instruction pattern that occurs as shown in figure 5.3c. From this pattern we can see that only one of the multiply values produced needs to be stored if we were executing the instructions in the shown order.

The vector width of an N-point FFT is equal to the number of data points, N. For a 1024-point FFT used in the NTT DoCoMo test setup, we perform 1024-point FFTs, meaning that the vector width is also 1024. FFT has a large amount of DLP because all 2-point FFT operations required between stages can be done in parallel. This means that the SIMD can be utilized almost 100%, suggesting we should increase the

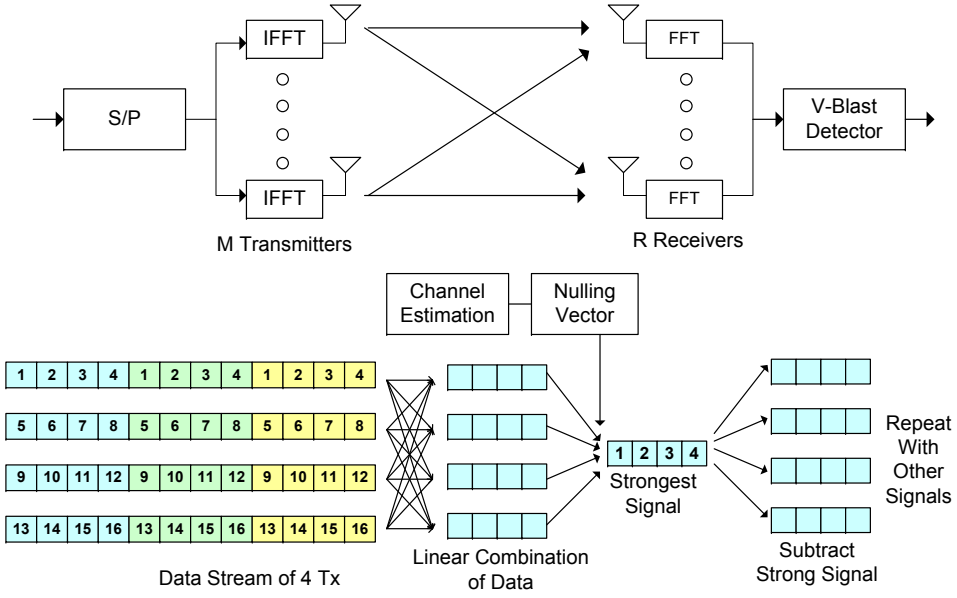


Figure 5.6: General decoding process of V-BLAST

SIMD width to be as large as the vector width of the algorithm in order to achieve maximum performance.

5.2.2 Space Time Block Codes (STBC)

As stated before, STBC is used to increase the signal quality of the transmitted signal. Figure 5.4 shows the general operation of STBC. The same data is transmitted through each antenna but its representation is different for each antennae. Specifically, the signal is transmitted through multiple antennae in conjugate forms with different orderings. Signal quality is increased by receiving the redundant copies of the same data signal and optimally combining the information from each receiver to produce better quality estimations of the original data signal. The implementation we used is based on Alamouti's 2x2 scheme[5].

In the STBC encoder and decoder, the vector width is only 4 elements. Though the vector width is small, each data set is independent. This means that we can join many data sets together and process one large set. The set size would be limited only by the amount of data the FFT provides. In our case, this would suggest that a 1024 width data set would be most optimal though this will be dependent on the final 4G standard.

Figure 5.5a shows the IEF for one of the major inner loops of STBC. These instructions implement part of the STBC signal detection operation on the SODA processor. Like FFT there exists many edges in the IEF where the values produced by instructions will only be used once and then never referenced again. We see a common instruction pattern that occurs as shown in figure 5.5b. From this pattern we can see that only one of the multiply values produced needs to be stored if we were executing the instructions in order.

5.2.3 Vertical Bell Laboratories Layered Space-time

V-BLAST is a spatial multiplexing scheme that improves the data rate by transmitting independent data streams over multiple antennae. This technique combines the multiple signals to obtain higher data rate rather than combine the same signal like STBC.

The general decoding process of V-BLAST consists of two major steps: channel estimation and signal detection as shown in figure 5.6. The channel matrix is estimated based on pre-defined training symbols. The operations for channel estimation

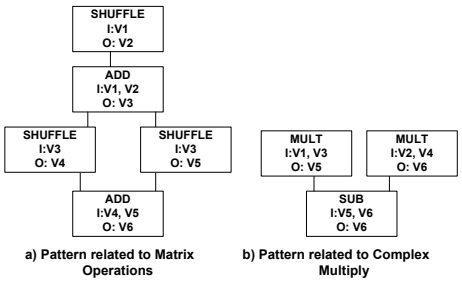
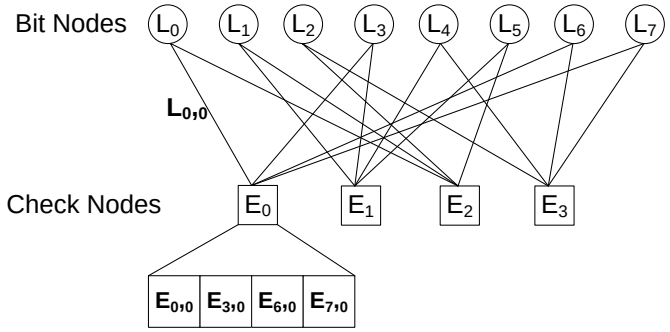


Figure 5.7: These two instruction patterns are reused many times within V-BLAST. They are common in many of the matrix operations.

are relatively simple with shift and sign-change operations. Once the channel matrix has been estimated, the detection order is determined. The detection order is based on signal strength found among all the signals received. The strongest signal is selected and extracted from the received signal. This process is repeated for the remaining signals. This process is iterative. The V-BLAST algorithm we implemented was based on work by Zhu [93], which reduces the computational complexity of the general V-BLAST decoder.

Because our system is based on a 4 transmit and 4 receive V-BLAST, the vector width of V-BLAST is 4 elements. The dimension of the channel matrix is 4x4 and the data signal is 4xN, where N is the number of data points in the FFT. The calculations performed are matrix operations with the channel matrix. Though the vector width is only 4, we can exploit larger SIMD widths because we can process larger sections of the 4xN data signal. The algorithm itself can support SIMD widths up to $4N$.

Unlike FFT and STBC, V-BLAST has lots of control and predicate operations within it's algorithm. Though this control and predication can be parallelized, the IEF becomes very complicated. Figure 5.7 shows some of the few common instruction



(a) Graphical representation of LDPC code

1. Initialization $E_{n,m} = 0, L_{n,m} = I_n$

2. Bit Node Operation $L_{n,m} = L_n - E_{n,m}$

3. Check Node Operation $E_{n,m}^{New} = - \prod_{n' \in N(m) \setminus \{n\}} sign(L_{n',m}) \cdot \min_{n' \in N(m) \setminus \{n\}} |L_{n',m}|$

4. Bit Update $L_n^{New} = L_{n,m} + E_{n,m}^{New}$

(b) LDPC decoding in 4 steps

Figure 5.8: LDPC graphical representation and decoding operations

patterns that are found in the algorithm. Figure 5.7a is often used within matrix operations. The shuffle and add operations help align the different rows and columns together and perform the complex number operations. Figure 5.7b is used frequently in complex number operations. Because this V-BLAST deals with complex numbers the most common operation is the complex multiply. Within these two operations there are values which are produced and consumed and never used again outside the small set of instructions. These values potentially do not have to be written into the RF.

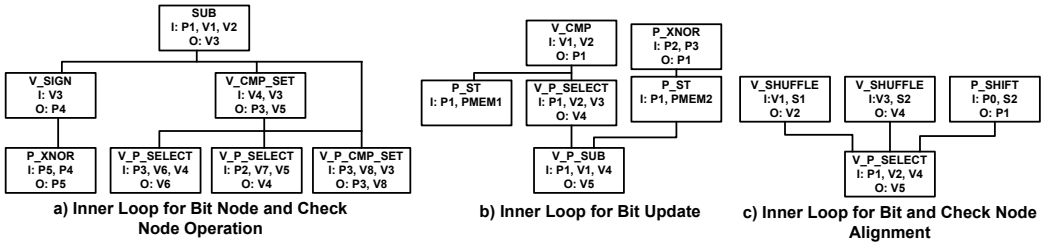


Figure 5.9: The inner loop operations of LDPC. These three operations represent the majority of the LDPC workload. They correspond closely to steps 2-4 of the LDPC decoding operation.

5.2.4 Low Density Parity-Check Codes (LDPC)

LDPC is an error correcting code that can perform closer to Shannon's limit than any other code. This means that LDPC can be used to achieve the highest data transmission rate possible over a wireless channel. LDPC is made up of only simple adds, subtracts and compares. LDPC has no serial dependency in operation unlike Turbo Codes that have to process SISO decoders serially after the interleaver. Our implementation of LDPC is based on Seo's work [83] which was optimized for the SODA processor and 802.16e.

The graphical representation of LDPC is shown in figure 5.8(a). The check nodes represents the number of rows in the parity check code and the bit nodes represent the number of columns. The edges connecting the check nodes and bit nodes are the 1's in the parity check code matrix—all other values are 0. The LDPC decoding operation is broken down into 4 stages as shown in figure 5.8(b). These four stages are the Initialization, Bit Node, Check Node, and Bit Update operation. This implementation is based on the Min-Sum algorithm.

Figure 5.9 shows the main operations within the inner loops of the LDPC decoder. These operations are responsible for the majority of the work within LDPC. A large amount of time in LDPC is spent within the bit and check node alignment. The alignment operation, as shown in figure 5.9c, shuffles two vectors and then combines different sections of the two vectors into one vector. There is no need to save the values that are produced by the two shuffle operations because they are used only by the following select instruction. Outside this operation the values produced by the shuffles are never used.

LDPC is naturally parallel unlike other error correction codes. The vector width of the algorithm is related to the z size of the circulant shifted identity matrix *(zxz)*. The z value we used was 96 which corresponds to the maximum LDPC block size in 802.16e [2], which we assume is the highest data rate because it allocates the most number of subchannels. This means that we can benefit from SIMD widths up to 96 elements using this z value. After this limit there is no performance benefit. Unlike the other algorithms, we cannot overlap multiple z wide block rows of the LDPC matrix because there exists a data dependency between block rows. This prevents us from utilizing SIMD widths larger than 96 elements. Though 802.16e uses a z values of 96, 4G may use larger z values allowing SIMD widths larger than 96 to be beneficial.

Algorithm	Overhead Workload (%)	Scalar Workload (%)	Vector Workload (%)	Vector Width (Elements)
FFT/IFFT	61	5	34	1024
STBC	14	5	81	4
V-BLAST	24	6	70	4
LDPC	33	18	49	96

Table 5.1: Data level parallelism analysis for major 4G kernels.

5.3 Methodology

We used the SODA architecture as our SDR SIMD architecture to explore the scalability of SIMD width and also the modified RF. We used SODA because we could modify the implemented Verilog hardware model for different SIMD widths and change the RF. We modify the kernels' assembly code to support these multiple widths and to use the modified RF. All assembly code was written based on implementations cited previously in SODA assembly. All of the optimizations and assembly code were done by hand. We synthesized SODA using Synopsys' Physical Compiler in 0.13 micron technology for 400MHz. Energy values were then extracted from the models and total energy was estimated based on the execution of the kernels.

5.4 SIMD Width Analysis

In Table 8.1, we analyze the DLP contained within the kernels. The instructions are broken down into 3 categories: overhead workload, scalar workload and vector workload. Overhead workload consists of all the instructions that assist SIMD computations, for example SIMD loads, stores and shuffle operations. The scalar workload

consists of all the instructions that are not parallelizable and have to be run on the scalar unit. The vector workload consists of all the raw SIMD computations that use the ALU, multiplier, and shift units.

From the table, we can see that FFT is dominated by the overhead workload of loading the SIMD data and shuffling it. STBC and V-BLAST have a very high SIMD computation which suggests that these algorithms are dominated by raw computations and may be adaptable to very wide SIMDs. Finally, LDPC seems to have a mixture of all three types of instructions that suggests that performance may be limited by the non-vector workloads.

The table also presents the natural vector widths of the algorithms. Because we are using a 1024 point FFT the natural vector width is 1024. This means that this algorithm can support a SIMD of up to 1024 and still show performance improvement. This is very different from the other 3 algorithms whose vector widths are much narrower. For STBC and V-BLAST the vector widths may be small, but each set of elements are independent of each other allowing us to map multiple sets of 4 element computations onto any larger sized SIMD to increase performance. LDPC vector width may be its limiting factor because after a SIMD width of 96, the algorithm is constrained by the overhead and scalar workload, which prevents mapping of multiple sets of the vector elements. Any SIMD width larger than 96 will see no benefit in performance.

We took each of the major kernels of the 4G system and mapped them onto wider versions of the SODA architecture. Most of the algorithms parallelized quite easily.

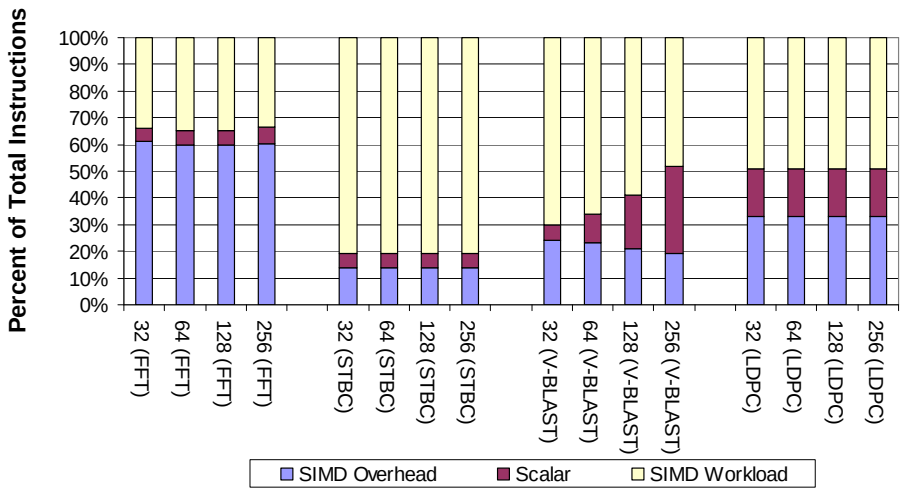


Figure 5.10: Instruction breakdown of each kernel with respect to SIMD width.

FFT and STBC were especially easy to parallelize. As we can see in Figure 5.10, the instruction breakdown hardly changed from widths 32 to 256 because these algorithms are composed mainly of loops containing SIMD computations. By increasing the SIMD width, only the number of loop iterations changed. Lastly, V-BLAST is somewhat of an exception, because as we increase the width, the scalar instructions start to dominate. Thus the performance of V-BLAST will eventually be bounded by the scalar workload but, as we can see, the benefit of SIMD width can still provide major benefits.

In Figure 5.11, we show the normalized speed up of the kernels for different SIMD widths. All the values were normalized to the 32 wide SODA implementation. For comparison purposes, the 32 wide SODA implementation can perform FFT almost 10 times faster than the TI TMS320C6203. For FFT, STBC and V-BLAST, the speed up is linear with width: doubling the width, yields 2x return for each of the

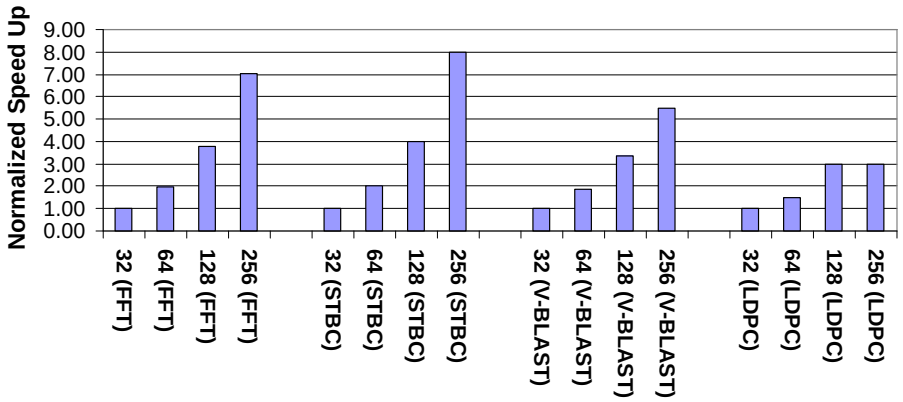


Figure 5.11: Normalized speedup of each kernel with respect to SIMD width.

algorithms. The performance benefit of increasing SIMD width is apparent. LDPC, though, is a different story. Because of the natural vector width of 96 we can only extract limited parallelism within the kernel. For SIMD widths greater than 128, there is no improvement in performance. The large jump between 64 and 128 width SIMD occurs because we cannot map all 96 values onto on a 64 wide SIMD machine. This forces us to do two iterations instead of the one possible with the 128 and 256 width SIMD. Because LDPC is the major performance bottleneck of 4G, this suggests that increasing SIMD alone may not meet the processing requirements.

Finally in Figure 5.12, we show the energy consumption of each kernel for different widths. We computed the energy consumed taking into account leakage. As we can see, for FFT, STBC and V-BLAST there is a great benefit from increasing SIMD width. We get the greatest performance increase with reasonable increase in energy consumption. The exception again is LDPC. Going from 32 to 64, we take a large energy penalty mainly due to the fact that many SIMD lanes are wasted because

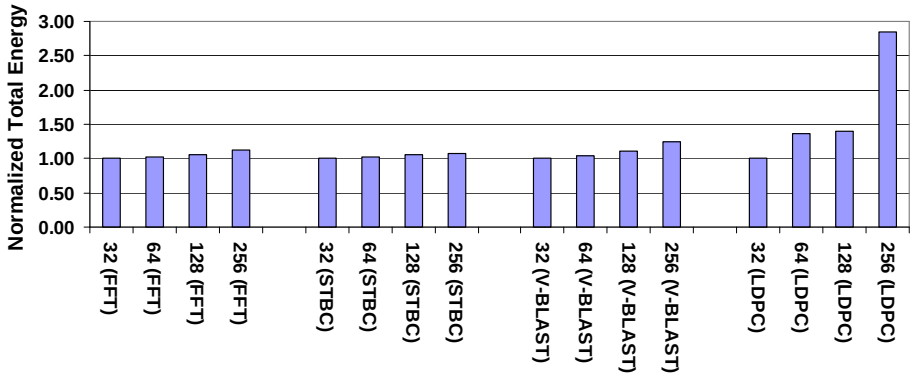


Figure 5.12: Total energy consumption of each kernel with respect to SIMD width.

the algorithm cannot map perfectly onto widths that don't divide into the natural vector width. Between 64 and 128, there is not much change because the algorithm is actually mapped relatively efficiently on the SIMD. The biggest jump is at 256 because all SIMD widths greater than 96 will waste energy doing unneeded work on the remaining 160 lanes. SODA does not support clock gating [57], which can help alleviate this problem by shutting off the lanes not being used, but will still result in inefficient use of silicon area for this algorithm and wasted potential performance.

We find that for these kernels increasing SIMD width does give us a good increase in performance with little increase in energy. The exception to this was seen in LDPC because the algorithm did not gain any improvement when SIMD width was greater than 128. As stated in section 5.2, future implementations of LDPC may have larger z values which would allow us to efficiently use larger SIMD widths.

Algorithm	FFT/IFFT	STBC	V-BLAST	LDPC
RF Reduction (%)	59	53	12	37

Table 5.2: Percent of total register file access that are produced and consumed within a subgraph. These register file accesses can use the small register file

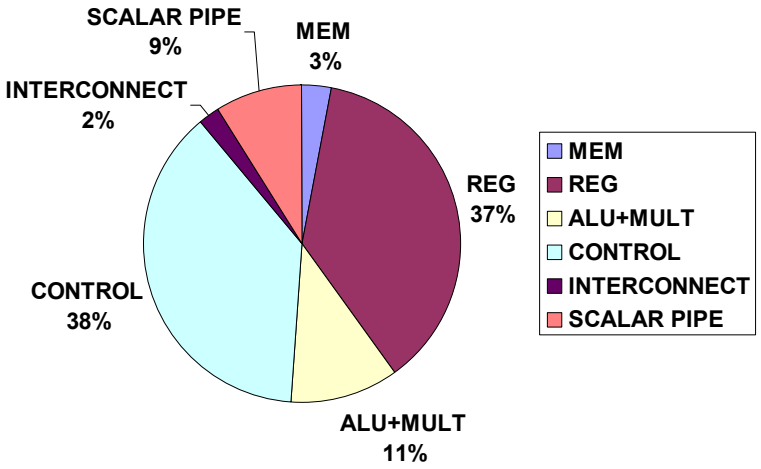


Figure 5.13: Power breakdown for the kernels on SODA

5.5 Register Access Analysis

Figure 5.13 shows the power breakdown of SODA running the 4G kernels. We can see the vector RF consumes almost 37% of the total energy. Others have shown that for other processors, RF power consumed almost 25% of the total power [31]. By analyzing the RF access patterns of the kernels we find that we can reduce the total energy consumed by the RF.

In section 5.2, the inner loop operations of each kernel were discussed. Each kernel had instruction patterns within the inner loops where register values produced were then used by the next following instruction and then the value was never referenced

again. This was apparent in FFT and STBC which had simple inner loops. In the SODA architecture, as with many common DSPs, the result value of each instruction is written back to the RF and when the value is needed it is read out of the RF. If the value is only referenced within a few instructions then there is no reason the value itself has to be written back into the main RF. Energy is wasted by writing and then reading the register value. Instead, the value can be either forwarded or stored in a small temporary buffer which can be accessed by any instruction. Techniques to implement this have been shown in [95][67][20].

By analyzing the kernels running on SODA, we found the percent of RF access that had potential for registers not writing into the main RF. This is listed in Table 5.2. From the table we find that FFT, STBC and LDPC have large amounts of RF accesses that can bypass the main RF. Our implementation of V-BLAST does not have large amounts of registers that can be bypassed because there are large amounts of control within the algorithm. As seen in the SIMD width analysis, the bottleneck of this kernel are the scalar control instructions. Because of this control, the registers need to be kept until the condition that chooses the value is calculated. This requires the use of the full RF. Though there are lots of register value which need to be kept, there are operations within the kernel which can be bypassed. These operations are related to all the matrix and complex number arithmetics that are done within the kernel.

The modification to SODA, shown in figure 5.14, was a small two-entry RF and register write bypass which is explicitly controlled by the instruction. The two-entry

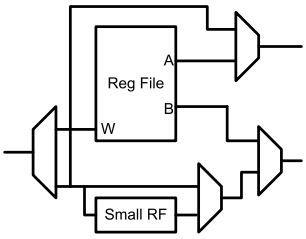


Figure 5.14: Modified Register File. Data can either be written to the main register file, the small register or bypassed completely from both allowing data to be forwarded directly to the next functional unit.

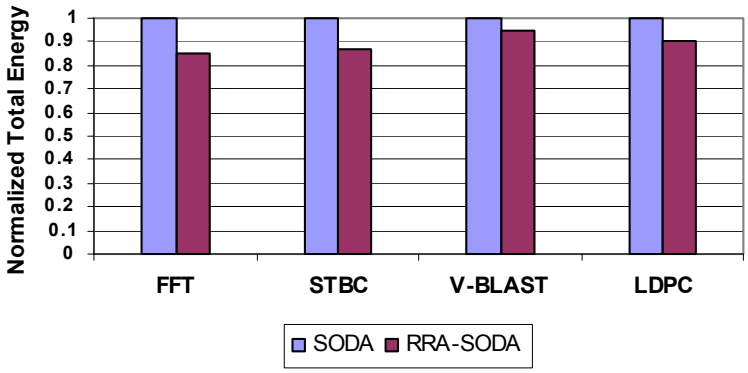


Figure 5.15: Total energy consumption of each kernel with reduced register file access. SODA is the baseline processor where each instruction always writes back to the register file then is read when needed. RRA-SODA is the modified SODA architecture that can write the values to a smaller register file.

RF is a partition of the main RF similar to split register files [95]. We implemented it by mapping two registers of the main RF registers to the smaller RF. When a request for those registers is made, we disable the read/write to the main RF and mux the data from/to the smaller RF. Register write bypass prevents the value from being written to the RF and forwards it directly to the input of the next instruction. This is done by an extra bit in the instruction. If the bit is set then the instruction will not write the value into the RF because the next instruction will use the value and the

value will not be used again. The next instruction, which references the same register as the current instructions' register destination, will get the data forwarded directly from the write port of the RF to the correct read port without the value being written to the RF. Figure 8.2 shows the amount of energy savings that can be achieved on a modified SODA architecture. From the figure we see that we are able to save between 6-15% of the total energy in the processor while maintaining the same performance. These results were as expected, following what we saw in Table 5.2. FFT, STBC and LDPC showed substantial energy reduction by reducing RF accesses while V-BLAST showed far less reduction in energy. This total reduction in energy translates into lower total power of the processor for these set of kernels.

5.6 Summary

Though the power and performance requirements of 4G is a significant challenge for designers, scaling SIMD width and reducing register accesses can help us gain major performance increases and reduce power consumption. We have seen almost a doubling of performance with doubling SIMD width. Not all kernels benefited from the increase in SIMD width. LDPC clearly is a major limiting factor in 4G. By increasing the SIMD width, FFT, STBC and V-BLAST benefits but LDPC benefits less. This suggests that LDPC may better be implemented on an accelerator or another specialized core with a different SIMD width. By reducing the RF accesses, we reduced the total energy by between 6-15% across the kernels. While FFT, STBC, and LDPC had the most energy reduction because many of the RF accesses could be

reduced, V-BLAST had less reduction in energy because many of the values had to be kept.

SIMD scalability and register access reduction are just two techniques in the processor design where we can extract more performance and reduce energy. Referring back to section 5.2, we notice that many of the kernels had similarities between their common subgraphs. In future work we will see if we can take advantage of these common subgraphs to improve performance. Another open issue is compilation for SIMD architectures. In this work, the kernels were hand written in assembly and optimized for varying with SODA. This prevents code compatibility among other types of systems with varying widths. Further work has to be done on kernel compilation for SIMD architectures and scheduling the code for varying widths. With a combination of the two architectural techniques presented and future work in SDR processor design, we may eventually be able to process 4G efficiently, within the power and performance requirements.

CHAPTER 6

Multimedia Algorithms

Video compression standards, such as MPEG-4 and H.264, are being actively considered for mobile communication systems because of the increasing demand for multimedia content on handheld devices. In this chapter, H.264 is selected as the multimedia benchmark because it achieves better compression compared to previous standards and also contains most of the basic functional blocks (prediction, transform, quantization, and entropy decoding) of previous standards. Of the three profiles in H.264, we focused on the *Baseline* profile due to its potential application in videotelephony, and videoconferencing.

Figure 6.1 shows a block diagram of a H.264 decoder. The H.264 decoder receives a compressed bitstream from the network abstract layer (NAL). The first block is the entropy decoder which is used to decode the bitstream. After reordering the stream, the quantized coefficients are scaled and their inverse transform is taken to generate the residual block data. Using header information in the NAL, the decoder selects prediction values for motion compensation in one of two ways: from a previously

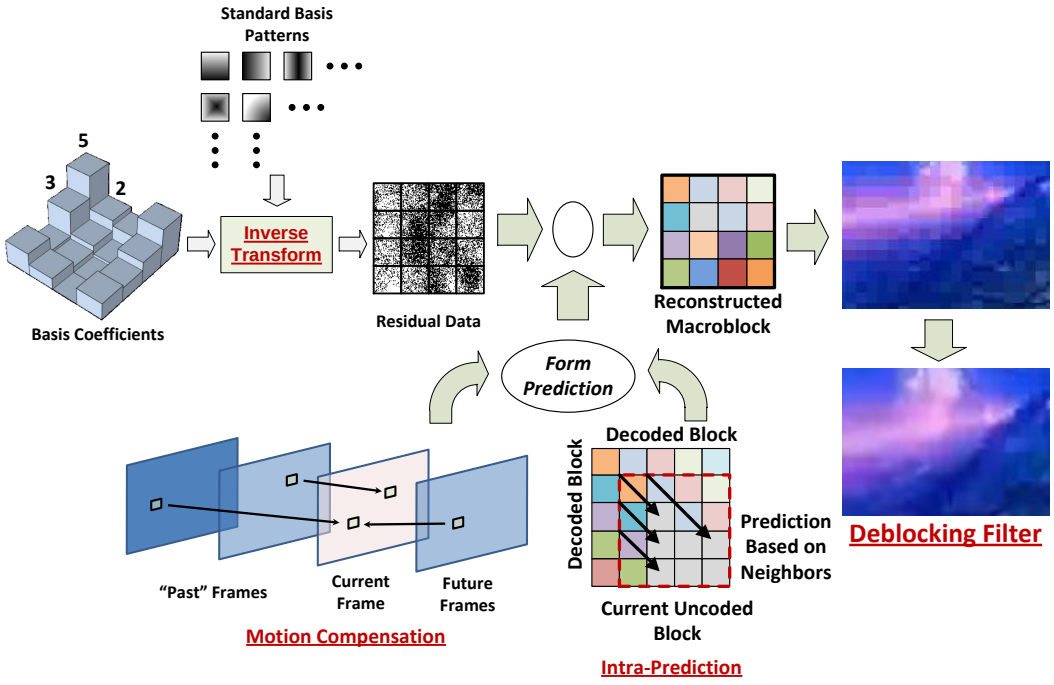


Figure 6.1: Block Diagram of H264

decoded frame or from the filtered current frame (intra-prediction). The pixelated block is then put through a deblocking filter in order to smooth out the block. According to the power profile of H.264, about 75% of the decoder power consumption is attributed to three algorithms: deblocking filter (34%), motion compensation (29%), and intra-prediction (10%) [53]. Therefore, these three algorithms are selected as the H.264 kernel algorithms in this study.

6.1 H.264 Algorithm Analysis

In this section, we describe the main H.264 kernels.

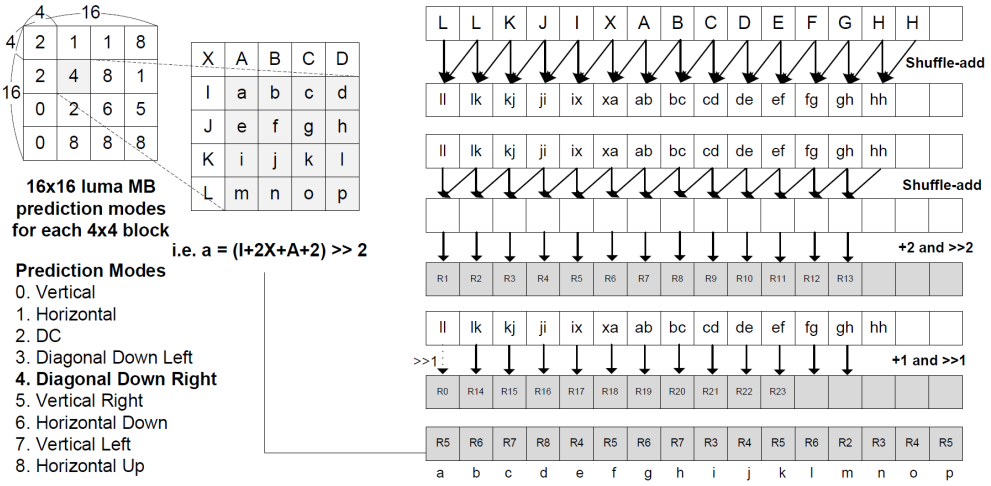


Figure 6.2: Mapping a 16x16 luma macroblock intra-prediction process on the proposed architecture. Example of the Diagonal Down Right intra-prediction for a 4x4 sub block (grey block) is presented with fused operations.

6.1.1 Intra Prediction

In H.264 intra-prediction, there are nine prediction modes - Vertical, Horizontal, DC, Diagonal Down Left, Diagonal Down Right, Vertical Right, Horizontal Down, Vertical Left, and Horizontal Up. A 16x16 luma macroblock is broken into sixteen 4x4 sub blocks. The 16 prediction values $(a, b, ..., p)$ for each 4x4 sub block is calculated with neighboring pixels $(A, B, C, D, I, J, K, L, X)$. At the encoder, all the prediction modes are calculated and the best predicted one is chosen. At the decoder, the sub block is generated based on the prediction mode in the header information sent by the encoder.

There is significant overlap in the computations of six of the modes. The other three modes, namely, Horizontal, Vertical, and DC mode, are computed using only a swizzle operation and an summation. Fig. 6.2 shows how to compute the partial

Prediction Mode	Swizzle Pattern
Diagonal Down Left	7,8,9,10,8,9,10,11,9,10,11,12,10,11,12,13
Diagonal Down Right	5,6,7,8,4,5,6,7,3,4,5,6,2,3,4,5
Vertical Right	18,19,20,21,5,6,7,8,4,18,19,20,3,5,6,7
Vertical Left	19,20,21,22,7,8,9,10,20,21,22,23,8,9,10,11
Horizontal Down	17,5,6,7,16,4,17,5,15,3,16,4,14,2,15,3
Horizontal Up	16,3,15,2,15,2,14,1,14,1,0,0,0,0,0,0

Table 6.1: Shuffle patterns for six intra prediction modes for 4x4 luma

intermediate values that are reused for the six prediction modes. Two sets of partial sums are generated for a 4x4 sub block. After generating R0 to R23, these intermediate values are distributed. Table 6.1 shows how to swizzle the partial sums for each prediction mode.

6.1.2 Deblocking Filter

H.264 deblocking filter smoothes the block edges to reduce blocking distortion without affecting the real edges. Based on block strength, the function of this filter varies dynamically (three-tap, four-tap, or five-tap filter). Furthermore, there is an order in which the edges have to be filtered. Fig. 6.3 shows the deblocking filter process. Edges A-B, E-F, I-J, M-N are filtered in parallel. To avoid memory access conflict, sub blocks A,B,I,J (which belong to four different sub banks) are loaded first, followed by E,F,M,N, etc. The four groups of 16 pixel values are permuted in the memory system to generates eight groups of horizontally aligned eight pixel values.

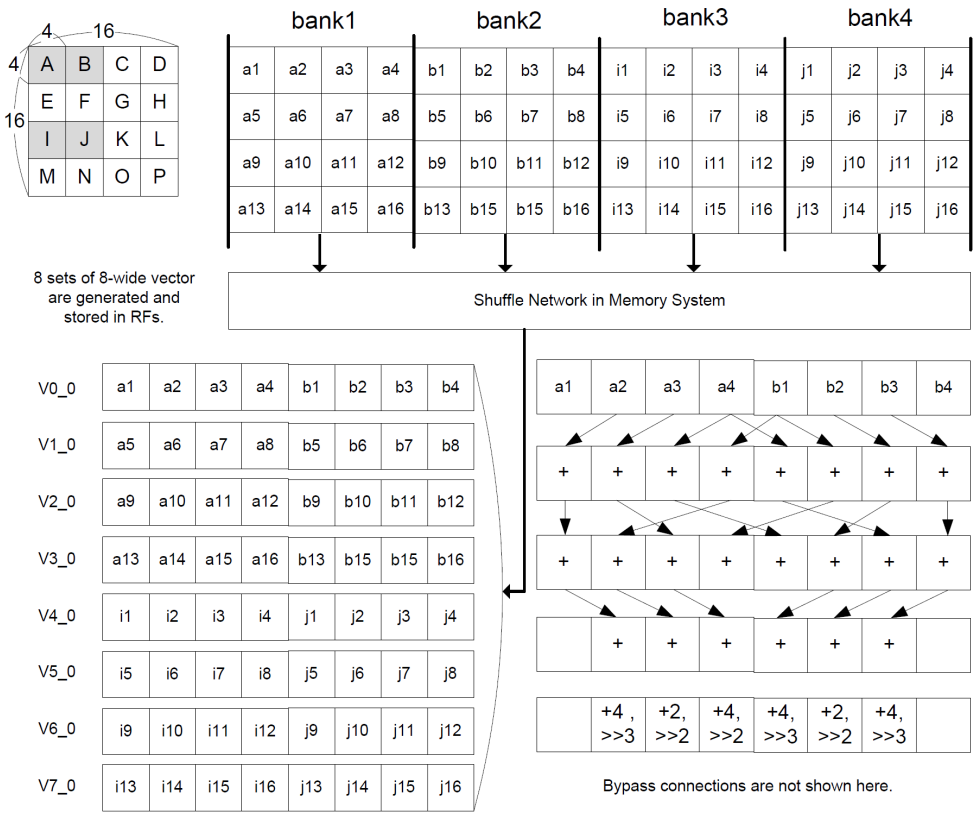


Figure 6.3: Mapping a deblocking filter process when BS (Block Strength)=4.

6.1.3 Motion Compensation

In H.264, the size of the motion compensation block can be 16x16, 16x8, 8x16, 8x8, 4x8, and 4x4, and the resolution can be integer-pixel, half-pixel, quarter-pixel, or eighth-pixel. Because sub-sample positions do not exist in the reference frames, the fractional pixel data are created by interpolation. Half-pixel interpolations are derived by a six tap filter as shown in Eq.1 in Fig. 6.4. The equation is modified to reduce multiplications and to express the six tap filter in terms of partial sums and differences of the original pixel values. This helps in exploiting the re-usability of computations for subsequent half pixel interpolations (Eq.2) in Fig. 6.4.

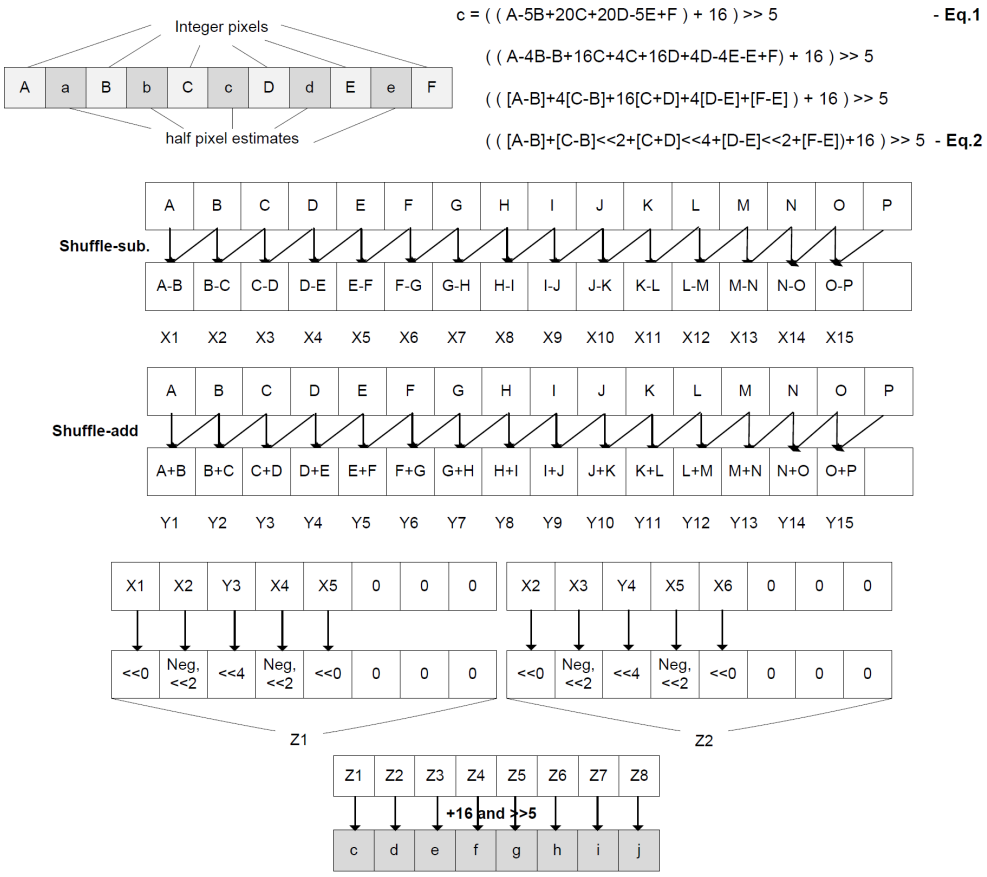


Figure 6.4: Example of interpolation of motion compensation (half-pel).

6.1.4 Motion Estimation

Motion estimation of an MxN block involves finding an MxN sample region in a reference frame that closely matches the current block. An area in the reference frame of size $2M$x$2N$ centered on the current block position is searched, and the minimum SAD value is needed to determine the best match. Fig. 6.5 shows the mapping method for a 4x4 block (current frame) in an 8x8 search area in the reference frame. The pixels of the current 4x4 block $(a, b, c, ..., p)$ are loaded from the memory, and the pixels in the shaded 4x4 block $(f1, g1, h1, e2, j1, ..., a4)$ in the search area are obtained

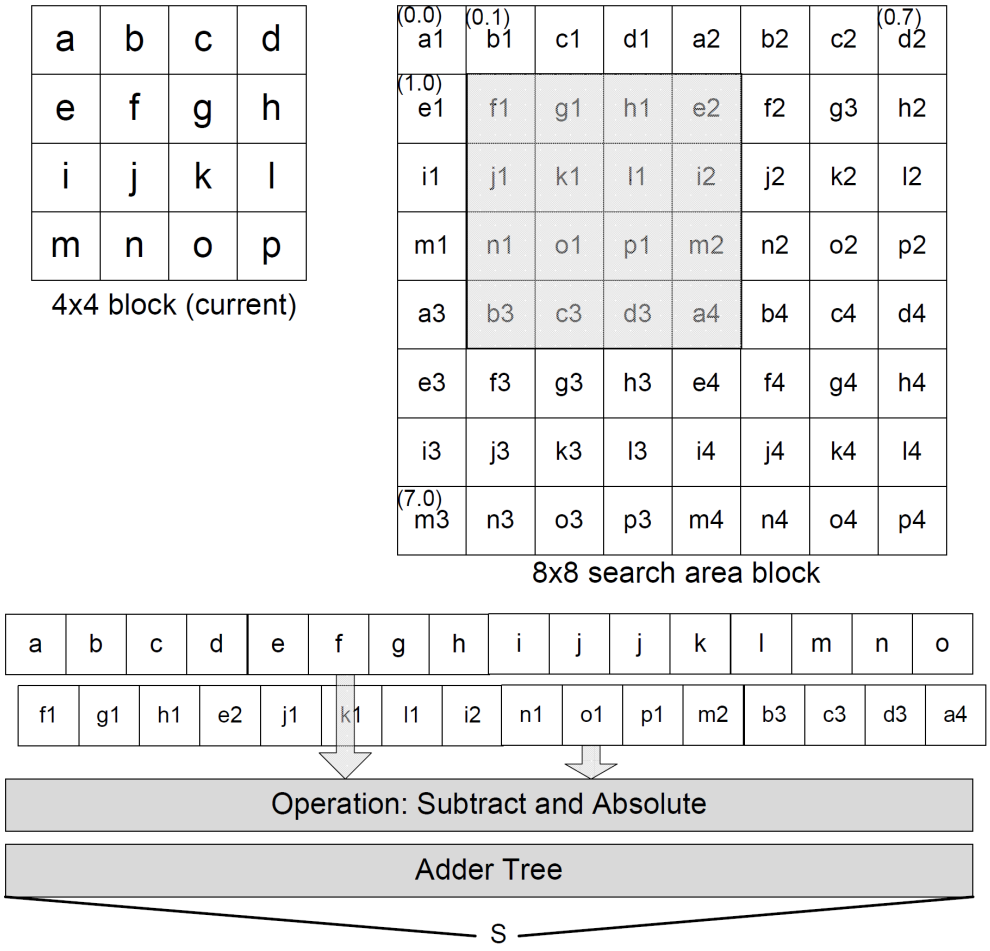


Figure 6.5: Mapping a motion estimation process for a 4x4 block on the proposed architecture; The search area is 8x8.

using memory loads and swizzles. The SAD value is calculated by a fused operation (sub-abs) and summation using the adder tree. The first SAD value is stored as the minimum SAD and is updated during subsequent computations. This process repeats for 25 possible positions in the 8x8 search area.

Algorithm	Kernel Operation	SIMD Workload	SIMD Width	TLP Level
Intra-pred (dec.)	13-tap filter	75.48 %	16	Med.
Intra-pred (enc.)	13-tap filter	91.06 %	16	High
Deblocking Filter	3,4,5-tap filter	86.61 %	8	Med.
Interpolation (MC)	2,4,6-tap filter	81.59 %	8	High
Motion Estimation	SAD (16)	62.46 %	16	High

Table 6.2: Kernel operations, SIMD workload, required SIMD width, and the amount of thread level parallelism (TLP) for H.264 encoder/decoder algorithms

6.2 H.264 Architectural Implications

In this section, we analyze key algorithms in H.264 and propose several architectural design decisions to improve the processing performance and power efficiency.

6.2.1 Multiple SIMD Widths

Table 6.2 shows the workload profiling for the key H.264 kernel algorithms. The other important computational kernels such as transform, quantization, and entropy coding are not included in this study because the transform/quantization kernel is easily parallelizable and is not the performance bottleneck, and the entropy coding is completely sequential and can be mapped only to a scalar processing unit. The available data level parallelism (DLP) expressed in terms of SIMD workload, natural SIMD width, and the thread level parallelism (TLP) for the key parallel H.264 algorithms are presented in Table 6.2. The SIMD workload consists of the arithmetic and logical computations that can be mapped to the SIMD pipeline. The scalar workload represents the instructions that are not parallelizable such as loop control and ad-

dress generation, which run on the scalar pipeline and the AGU pipeline respectively. The overhead workload includes all the instructions that support SIMD computations such as SIMD memory operations and memory alignment operations.

As can be seen in Table 6.2, most of the H.264 kernel algorithms can exploit the SIMD datapath, but the required SIMD width varies. While the deblocking filter and interpolation have SIMD width of 8, intra-prediction and motion estimation have a SIMD width of 16. Kernels such as intra-prediction mode decision and motion estimation have high TLP, which means that independent threads corresponding to different macroblocks can be mapped onto the SIMD datapath. For these kernels, the wide-SIMD pipeline helps to increase the processing performance. Kernels such as intra-prediction and deblocking filter are not easily parallelizable, and a wide SIMD width does not guarantee higher performance. Therefore, even though it is easier to design SIMD architectures with a fixed SIMD width, we propose to support multiple SIMD widths to maximize the SIMD utilization.

6.2.2 Diagonal Memory Organization

Multimedia algorithms use two or three dimensional data unlike wireless signal processing algorithms that typically operate on single dimensional data. For example, the deblocking filter algorithm operates on horizontal edges followed by vertical edges. Row or column order memory access works well for one set of edges, but not for the other. A diagonal memory organization is more suitable here since blocks of pixels along a row or column can be accessed with equal ease.

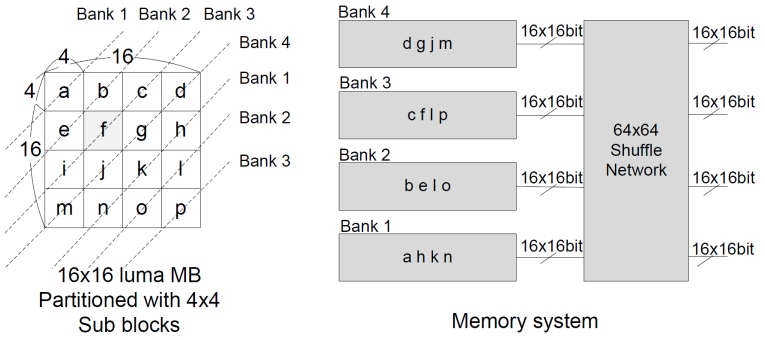


Figure 6.6: Diagonal memory organization and shuffle network, which allows the horizontal and vertical memory access without conflict. The 64x64 shuffle network realigns 64 16-bit data.

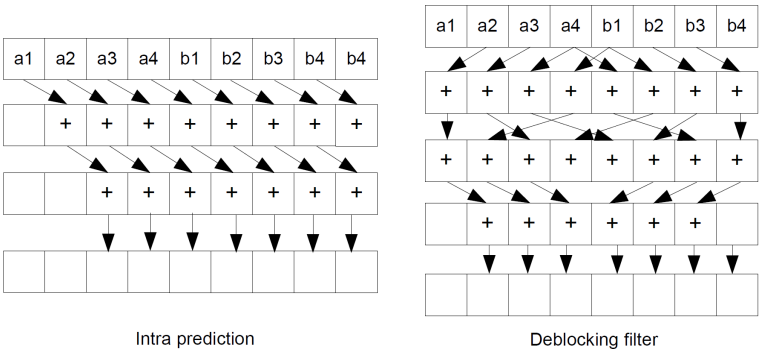


Figure 6.7: Subgraphs for the inner loops for two H.264 kernels; The bypass path is not shown for simplicity.

Fig. 6.6 shows how a 16x16 macroblock is stored in the proposed diagonal memory organization. The 16x16 macroblock is broken into 4x4 sub blocks (a, b, ... p) each containing 16 pixels. Groups of sub blocks (a, h, k, n), (b, e, i, o), (c, f, i, p), and (d, g, j, m) are stored in separate memory banks. This allows neighboring blocks which share horizontal and vertical edges to be accessed at the same time.

Algorithm	Shuffle-ALU	Add-Shift	Sub-Abs	Neg-Add
Intra-Pred.(Enc)	21.43 %	7.14 %	28.57 %	-
Intra-Pred.(Dec)	30.77 %	30.77 %	-	-
Deblocking Filter	49.48 %	16.49 %	-	-
Interpolation(MC)	30.09 %	3.76 %	-	15.05 %
Motion Estimation	24.04 %	-	48.08 %	-

Table 6.3: Instruction pair frequency for H.264 kernel algorithms

6.2.3 Bypass and Temporary Buffer Support

Fig. 6.7 shows the subgraphs for inner loops of two H.264 kernel algorithms. We see that there exists large amount of data locality. Moreover, intermediate data do not need to be stored in the register file (RF) because the values are usually consumed by the very next instruction and all not used anymore. Thus, it is sufficient to store these values in a temporary buffer or bypass them. These features have been inspired by recent works in [28] and [24], which show that storing short-lived data and bypassing RF reduce the power consumption and increase the performance.

6.2.4 Fused Operation

Many operations in DSP algorithms occur in pairs or tuples. The most common example is the multiply followed by accumulate, which has been exploited by many architectures. Table 6.3 shows the breakdown of the most frequent instruction pairs of H.264 kernel algorithms. Among all pairs, the shuffle-ALU pair is heavily used because most of the time, data must be aligned before being processed by the SIMD datapath. The frequencies of add-shift and sub-abs pairs are also very high. The sub-abs instruction pair is used in the SAD (Sum of Absolute Differences) operations

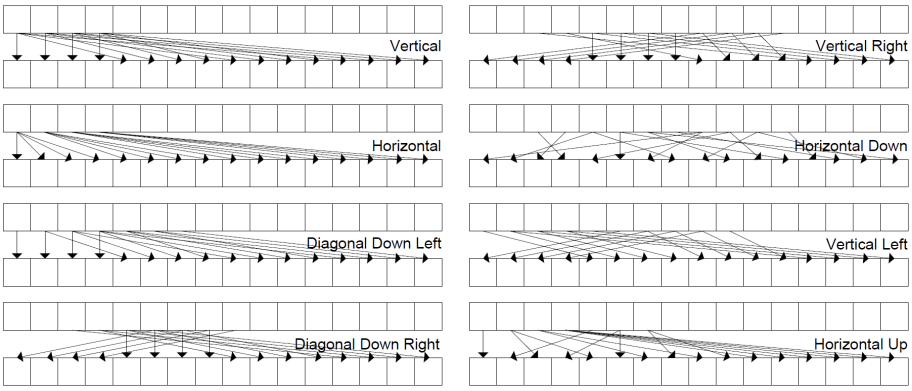


Figure 6.8: Permutation Patterns for H.264 Intra-prediction Modes

in motion estimation. The add-shift instruction pair represents the round operation, which is one of the most used operations in H.264 algorithms.

Based on this analysis, we propose to fuse the frequently used instruction pairs. This would increase performance and lower power consumption because unnecessary RF access can be significantly reduced.

6.2.5 Programmable Crossbar

Fig. 6.8 shows some examples of the SIMD permutation patterns that are found in H.264 intra-prediction algorithm. Even though the permutation patterns look very random, each H.264 algorithm - intra-prediction, deblocking filter, interpolation, and motion estimation - has a predefined set of shuffle patterns, and the number of distinct sets is typically less than 16.

Most commercial DSP processors and GPP multimedia extensions support some types of data permutations. These features are even more important in SIMD architectures for aligning data before the SIMD computation units. For instance, the

perfect shuffle network in SODA [54] supports a few sets of permutations in one clock cycle. But, if complex permutation patterns are required, multiple instructions need to be executed. These additional clock cycles degrade the timing and power performance. To support complex data access patterns in H.264 algorithms, we propose small low-power programmable fixed pattern crossbars. We place one of these between memory and register file to align data before loading and storing, and another between the register file and SIMD functional units to shuffle data before processing.

CHAPTER 7

Low Power SIMD Architectures using a Scalable SRAM Based Swizzle Interconnect

Not so long ago, growth in computing was dominated by PCs in the form of laptops and desktops. Today, smartphones and handheld devices like the iPad and iPhone are already poised to outsell PCs within the next few years. The capability of these devices has increased rapidly and now provide features like high definition video, high bandwidth internet access, and 3D graphics all within the same die. These devices will continue to grow in terms of capabilities and performance while still needing to adhere to a strict power budget.

Continued scaling of VLSI technology due to Moore's Law allows us to integrate an increasing number of transistors on a single die. Both in industry and academia we have seen one of the most power efficient way to utilize this transistor area is through integrating multiple processing elements (PE) within a die [14]. This is represented by many architectures in the form of increased number of single instruction multiple

data (SIMD) lanes in processors like[82], and the shift from multi-core to many-core architectures like [17][26]. As presented in previous chapters, these architectural innovations are the core to designing and efficient mobile signal processing architecture for wireless and high definition video.

As we scale the number of PEs in these architectures, traditional on-chip interconnects like buses and crossbars are unable to keep up with the bandwidth required to fully utilize the PEs. Moreover, traditional interconnects do not scale well in terms of power and area as the number of PEs increase. The National Science Foundation held a workshop trying to identify the most critical challenges facing on-chip interconnects and their findings showed that power and latency were the biggest challenges that needed to be solved [65]. Network-on-Chip architectures like those used in [89][47] show that the crossbar itself consumes between 30% to almost 50% of the total interconnect power. Another critical problem is that existing circuit topologies in traditional interconnects do not scale well because of the complexity in control wire and control signal generation logic which directly effects the delay and power consumption. This scaling problem in terms of area and power is one of the critical bottlenecks that limits our increase in the number of PEs in the future.

In this chapter we study a low power and scalable crossbar design, called XRAM [79], which provides a solution to the bandwidth and scaling problem seen in low power SIMD architectures. XRAM implements novel circuit techniques to solve the scaling problem while providing high bandwidth. Unlike other interconnect networks like [11][45], the XRAM is able to perform all permutations and swizzle operations,

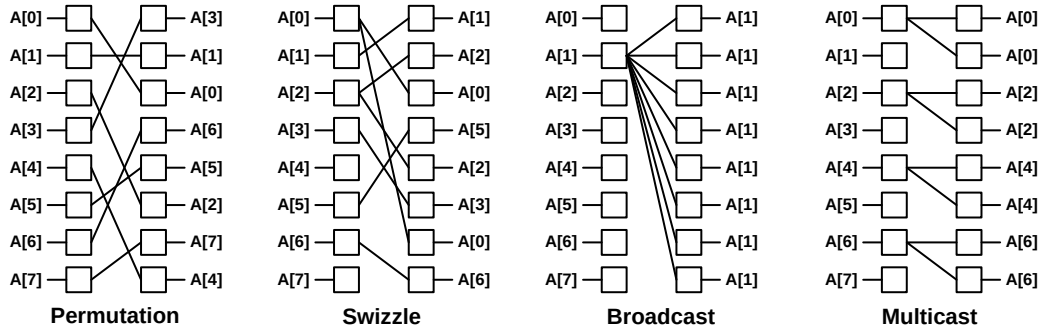


Figure 7.1: Permutations are 1-to-1 mappings of input to output ports. Swizzles are the generic form of duplication and permutation combined where the output does not need to contain all the inputs. Broadcast is a special form of swizzle where one of the values is broadcasted to all the outputs. Multicasting is another special form of swizzle where multiple input ports are duplicated in a regular pattern to the outputs.

including multicasting and broadcasting in a non-blocking way described in Figure 7.1. One circuit technique that helps solve the control complexity problem is embedding the cross point control within the cross points of a matrix style crossbar using SRAM cells rather than having it driven by an external controller. Other circuit techniques, like using the same output bus wires to program the cross point control, helps reduce the number of control wires needed within the XRAM. Finally borrowing techniques that are currently used in SRAM arrays helps improve performance and lower energy used in driving the wires of the XRAM. Though these techniques help solve the performance and scaling problem of traditional interconnects, one drawback is with flexibility; the XRAM is only able to store a certain number of swizzle configurations at a given time. These configurations are not static and can be changed and reprogrammed at run-time. Though this may seem like a major drawback, we show case studies for many applications where this is not a bottleneck and also architectural

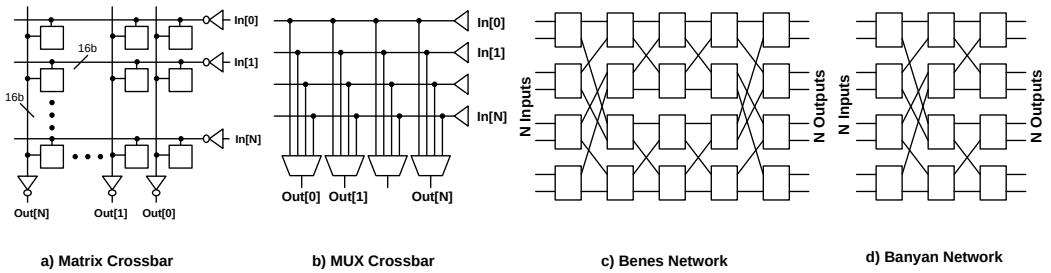


Figure 7.2: Different Commonly Used Network Toplogies

techniques that mitigate this flexibility problem.

The the chapter is organized as follows. Section 7.1 will discuss the traditional approaches that have been used in on-chip interconnects and the problems that each is faced with as we scale. Section 7.2 will introduce and discuss the operation of XRAM, a low power and scalable crossbar network. Section 7.3 will present case studies of the XRAM in use in a multiple of high performance applications and provide power and performance comparisons against other interconnects topologies. Finally, Section 7.4 will discuss future usage models that XRAM is an enabler for and Section 7.5 will present our conclusions.

7.1 Related Works

Many techniques are currently used to implement permutation networks and swizzle networks. In this section we discuss three current techniques used and show how each of these techniques perform with respect to scalability, speed, energy, and flexibility when implementing an NxN interconnect where N is the number of input and

output ports. Figure 7.3 summarizes the results for each of these techniques.

7.1.1 Crossbars

Typically in modern designs when permutation are needed they are implemented using fully connected networks like crossbars. Crossbars are able to realize all permutations and swizzle operations through its network. These networks are implemented in two different ways: MUX based crossbars or matrix style crossbars. Figure 7.2a and b shows the topology of these two networks. MUX based crossbars utilize Nx1 MUXes to drive the outputs while matrix style crossbars use switching elements like transmission gates or tristate buffers to drive the output at each cross point. Current SIMD processors which have narrow SIMD widths between 4-8, implement these style of crossbars. The problem with these networks is that as the number of SIMD processing elements increase, the complexity of these networks grows quadratically. In a MUX based crossbar, the data passes through log2(N) number of 2x1 MUXes which, combined with driving the wires, determines the critical path delay of the network. The area of the MUX based crossbar increases by $(2N\text{-}1)N^2$. As we start increasing N to large numbers, the area and energy consumption increase dramatically. With matrix style crossbars, the area increases N^3 with respect to the numbers of ports. This increases the wire capacitance that the input and output drivers need to drive, slowing the delay but more importantly dramatically increasing the energy consumption. Generating control for these types of networks is simpler in that in both cases a simple decoder is used to either control which cross point to enable or which set of

MUXes to select. Though the control generation is simplified, routing the wires for the control signals is very costly, increasing both area and power of the total network. Because of all these drawbacks, these types of networks can only be used when the number of ports is small.

7.1.2 Benes Network

Because of the scalability problems associated with the crossbar, many large parallel machines and interconnects use multi-stage interconnect networks(MIN) to perform their permutations. One of the more popular MIN networks is the Benes network [11] whose topology is shown in Figure 7.2c. The Benes network traditionally consists of ($2Log_2N$-1) number of stages where each stage consists of N/2 2x2 switches which connect two input ports to two different output ports. These networks are more scalable because the number of MUXes increases $O(N^2 \log_2 N)$ as opposed to crossbars which increase in complexity by $O(N^3)$. Though Benes networks scale much better than crossbars they lack full connectivity. Benes networks can perform all permutations non-blocking but cannot perform all multicast swizzle operations without internally blocking. This forces the network to buffer either internally or externally the data within the network and incur multiple cycle penalties to resolve swizzle operations. This limits the flexibility of these types of networks. Though flexibility is a problem with Benes networks, the largest problem is with the control generation of the Benes network. Unlike crossbars which have simple decoders, Benes networks require complex control logic to control the multiple switches within the network.

Much research has been done to alleviate this problem but all solutions still require complex logic or a large number of lookup tables to generate the control [61][50]. Because of this, if the permutation patterns change frequently the reconfiguration cost will become the bottleneck. Though the benefit of the Benes network is realized when the network acts like a circuit switched network, transferring large amounts of data, the multiple stages can be pipelined allowing for very fast throughput at the cost of increased total power consumption.

7.1.3 Banyan Network

Though Benes networks are very popular among routing networks, the control overhead and the number of stages can overwhelm the performance of the network as the number of ports increases. Because of this, another MIN network commonly used is the Banyan network shown in Figure 7.2d. The Banyan network consists of $\log_2 N$ number of stages where each stage consists of N/2 2x2 switches. This provides a network that is almost half the complexity of the Benes network, allowing it to scale to larger number of ports. This also simplifies the control generation complexity of the network because the number of control wires that needs to be generated is reduced. The tradeoff of complexity in the Banyan network comes at a price of flexibility. Because it uses less stages than the Benes network, the Banyan network is an internally blocking network. This means that only limited multicasting and not all permutations can be realized in this network without a contention problem within the network. Some architectures like [91] are able to cope with this issue

Network Type	Crossbar	Benes	Banyan	XRAM
Scalability	*Low* $O(N^3)$	*Medium* $O(N^2 \log_2 N)$	*Medium-High* $O(N^2 \log_2 N)$	*Very Scalable* $O(N^2)$
Speed	*Medium* $(\log_2 N)$ 2:1 MUX	*Slow* $(2 \log_2 N) - 1$ stages	*Fast* $\log_2 N$ stages	*Fastest* SRAM like Sense Amps
Energy	*High*	*Medium*	*Low*	*Very Low*
Flexibility	*High* Swizzle and Permutations	*Medium* Non-Blocking Permutations	*Low* Blocking Permutations	*Medium-High* Swizzle and Permutations Programmed Configurations

Figure 7.3: Comparison between Crossbar, Benes, Banyan and XRAM networks with respect to Scalability, Speed, Energy, and Flexibility where N is the number of ports in an NxN network

because only a limited set of permutations are supported by the ISA. For general purpose interconnect networks that require all permutations and swizzle operations, the drawbacks will dominate the total delay and power consumption of the network. This is due to the need for internal buffers within the network and input buffers connected to simple arbitration logic to resolve the blocking conflict which start to dominate the total power consumption of the network.

7.2 XRAM Fundamentals

XRAM is a matrix crossbar that leverages some of the circuit techniques used in SRAM arrays for improving area and performance. Figure 7.4 shows a system level diagram of XRAM. The input buses span the width while the output buses run perpendicular to them, creating an array of cross points. Each cross point contains a 6T SRAM bit cell. The state of the SRAM bitcell at a cross point determines whether or not input data is passed onto the output bus at the cross point. Along a column,

only one bitcell is programmed to store a logic high and create a connection to an input. Matrix type crossbars incur a huge area overhead because of quadratically increasing number of control signals that are required to set the connectivity at the cross points. To mitigate this, XRAM uses techniques similar to what is employed in SRAM arrays. In an SRAM array, the same bit line is used to read as well as write a bit cell. Until the XRAM is programmed the output buses do not carry any useful data. Hence, these can be used to configure the SRAM cells at the cross points without affecting functionality. Along a channel (output bus), each SRAM cell is connected to a unique bit line comprising the bus. This allows for the programming of multiple SRAM cells (as many bit lines available in the channel) simultaneously.

Swizzle networks have traditionally been highly interconnect dominated, rendering a significant amount of logic space under-utilized. This gets aggravated in sub 100nm technology nodes because of poor scalability of interconnect wires in comparison to transistors. In a 128x128 port swizzle network with 16 bit channels fabricated using industrial standard libraries in an IBM 65nm technology, the silicon utilization is only 18%. XRAM mitigates this to some extent by re-using output channels for programming, resulting in improvement of silicon utilization to 45%. To further improve silicon utilization, multiple SRAM cells can be embedded at each cross point to cache more than one shuffle configuration. In 65nm, a 16-bit bus width allows six configurations to be stored without incurring any area penalty. Any one of these configurations can be selectively programmed or used to shuffle data. We find that many applications, especially in the signal processig domain, only utilize a small

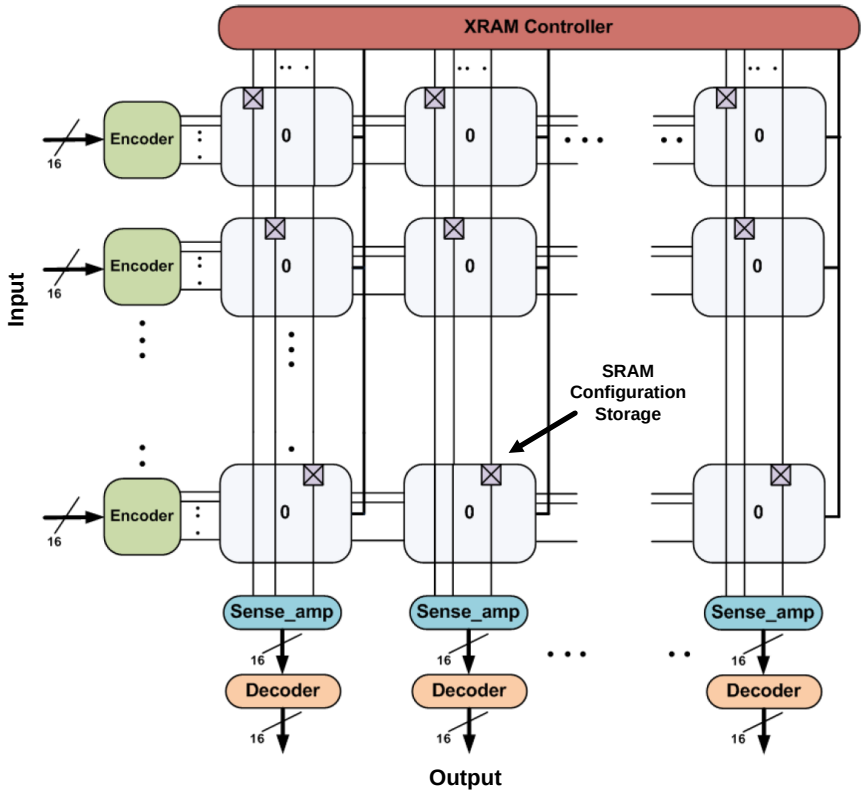


Figure 7.4: XRAM is a low power high performance matrix style crossbar that reuses output buses for control programming and stores multiple swizzle configurations at the cross points using SRAM cells. The XRAM controller sends the control line to pick which configuration SRAM cell to use. The encoder and decoder are the transition encoding/decoding logic used to minimize the switching power when the input and output bits do not change values.

number of permutations over and over again. By caching some of those patterns that are most frequently used, XRAM saves power and latency due to configuration signaling and generation that would have otherwise been incurred.

XRAM operates in two modes: programming mode and transmission mode. In programming mode, the controller sends a one-hot signal onto each output bus. A global wordline is then raised high to program the XRAM. With 16-bit buses, a 16x16 XRAM can be programmed in a single clock cycle. Larger XRAMs are divided into

multiple sections with independent wordlines and one section is programmed at a time. For example a 128x128 XRAM with 16 bit buses is divided into 8 sections with independent wordlines. To program a channel, in the first cycle all wordlines are raised high while sending an all zero code(16'b0) on the channel. This writes a logic low in all the bitcells in that channel. In the next cycle a one hot signal is sent while selectively raising only one wordline high to allocate the channel. For instance, to allocate the channel to input 43, the third wordline is raised high while sending 16'b0000100000000000 on the channel.

In transmission mode, the wordline stays low and incoming data is passed onto the output bus at the cross point storing a 1 using a precharge followed by conditional discharge technique. During the positive phase of clock, input data is launched and the bit lines are precharged to logic high. During the negative phase of clock, the bit lines are selectively pulled down if the data is high and the cross-point stores a 1. A bank of sense amplifiers evaluates the bit lines to retrieve data. The bit lines need not be pulled down all the way thereby saving power and improving performance. However, this technique results in power dissipation even with a non-switching input that stays high because the bit lines get precharged and discharged every cycle. To mitigate this, the incoming data is transition encoded at the input of XRAM. The original data is retrieved back at the output using transition decoders.

7.2.1 XRAM configurations

7.2.1.1 Embedded Configuration in XRAM

The fully programmable implementation of XRAM discussed above is most suited for generic SIMD processors that run a variety of algorithms, each requiring a small subset of vector permutations. This will allow the host processor to program the XRAM before running the application, thereby improving runtime. If larger numbers of permutation operations are required, a one cycle programming penalty will be incurred to reprogram one of the configurations stored locally within the SRAM cell. However, this requires shuffle patterns to be stored in SRAM cells beneath the crossbar wiring. Hence, only a few (6 in 65nm implementation with 16 bit buses) shuffle patterns can be supported; technology scaling will increase the number of configurations stored. But for application specific engines (like an FFT Processor) or a generic SIMD processor like Ardbeg [91] that supports only a limited number of shuffle patterns in its ISA, we propose an alternate version where SRAM cells are replaced by ROM based logic to encode shuffle patterns. In 65nm a 10 input logic gate can replace 6 SRAM cells without incurring any extra area while allowing 1024 shuffle patterns to be encoded. This is achieved by giving away the flexibility of having a user defined way to program the XRAM. However, a small number of programmable configurations (using SRAM cells) could be leveraged against fixed shuffle patterns to provide flexibility.

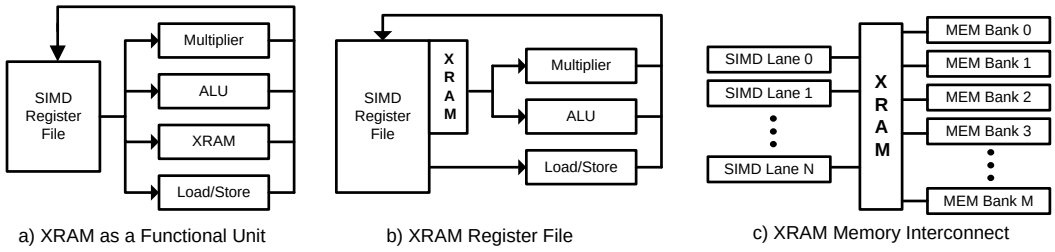


Figure 7.5: Different Placements of XRAM within the Processor Architecture. For the memory interconnect benefits occur when M $\geq$ 2N.

7.2.1.2 External Cache Configuration for XRAM

In this version, a large number of shuffle patterns are stored external to the XRAM and every cycle a pattern (control vector) is fed into the XRAM to permute data. The control vector is decoded locally at the cross points and stored in a memory element (Flip Flop) for use in subsequent cycles. For applications that require frequent programming, a separate bus (that is $\log_2$(Number of XRAM inputs)) can be used to enable simultaneous programming and data routing through XRAM. Fetching the control vector from the external cache and programming the XRAM needs to be done one cycle prior to the data routing cycle when the shuffle pattern is intended to be used.

7.2.1.3 Tradeoffs with Placement of XRAM in Processor Architectures

Figure 7.5 shows multiple placements of XRAM within a SIMD processor: as a separate functional unit, connected to the register file, and between the memory system and processor. Each placement has its benefits and drawbacks. In Figure 7.5a, the XRAM is implemented as another SIMD functional unit, in Figure 7.5b all SIMD

register file reads pass though the XRAM. Implementation (a) minimizes delay and power consumption when the re-arrangement functions carried out by the XRAM are implemented in RTL. Implementation (b) improves the CPI performance of the whole machine because swizzle can occur every time a register is accessed but it increases the total power consumption because swizzle is not always performed but the cost is the same.

The benefit of placing an XRAM between the memory blocks and the processing elements, as shown in Figure 7.5c, is that the system can cope with a larger number of memory banks than the number of processing elements. This is an interesting possibility since, as shown by Lawrie [48], access to rows, columns, diagonals, reverse diagonals, and blocks of a matrix cannot all be implemented if the number of memory banks is equal to the number of PEs, but can if the number of memory banks is two times greater than the number of PEs. Because one of the major applications that SIMD processors can accelerate are signal processing algorithms, optimizing matrix data alignment for SIMD can improve performance dramatically.

7.2.2 Summary of XRAM

This section presents a comparison between the multiple networks as shown in Figure 7.6. As shown, for sizes of ports, the XRAM has the fastest delay because it uses sense amplifiers to help evaluate signal. The Benes is the worst in terms of delay. This is because the critical path for the Benes network (is 2 $\log_2$N - 1) switch stages while the MUX based crossbar has only $\log_2$N. For small number of

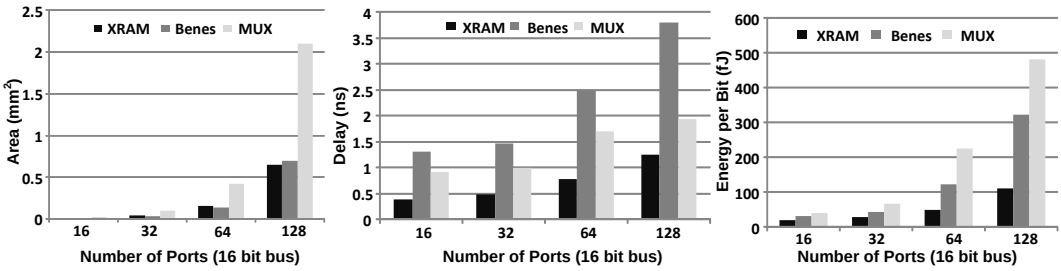


Figure 7.6: Comparison between the XRAM, Benes and MUX crossbar network with respect to Area, Delay and Energy per bit (total bits = number ports * 16-bit).

ports where N<8, the Benes network is the most power efficient because it uses less switches than the MUX based crossbar and internal node switching dominates the power consumption. For the XRAM, the initial cost of the sense amplifiers and the extra logic dominates the power consumption which are implemented to help drive long wires but do nothing if the interconnect driving power is low. We see that as the number of ports increases, the benefits of both the Benes and MUX based crossbar networks start to decrease rapidly. This is directly related to the increase in area and complexity of these networks. As the number of ports increases, the area and thus the wire interconnect power dominates the total power. The Benes network's gate complexity is $O(N \log_2 N)$ as opposed to the $O(N^2)$ complexity of the MUX based crossbar which is why the increase is less dramatic. But because the XRAM uses SRAM like sense amplifiers to evaluate the output buses, the power increases linearly. In all aspects the XRAM network is more efficient as the number of ports increases which enables us to build architectures with large number of input ports. Though the XRAM scales much better than the other networks, after 256 ports

the wire length of the output buses are too long for the sense amplifiers to resolve causing a drastic increase in power and multistaged or banked configurations become necessary. Another big advantage of the XRAM is in area and layout. The area is the smallest for all sizes of the XRAM compared to the other networks. One main reason for this is because we are able to minimize the control wires needed by storing the configuration at the cross points and also because we program the configurations using the output bus. Though the XRAM has the smallest area, the Benes network has properties that because each switch output only connects to one other output, the layout and routing itself can be minimized. Finally because the XRAM is built like a matrix crossbar, the input and output wires themselves can be located in any order. This simplifies the wiring and routing which allows the XRAM to be laid out with the only constraint being the wires for the input and output ports and bit width and not the switching elements which are the limiting factors in the MUX based crossbar.

7.3 Case Studies

Case studies for the XRAM were done across multiple applications in the signal processing and graphics domain. These applications benefit from not only increase in processing elements but also take advantage of the shuffle and multicasting operation of the XRAM network. These applications are compared against MUX based implementations. Each of the case studies were synthesized using IBM65nm technology with Synopsys design compiler and place and routed using Cadence encounter. Energy dissipation were generated from the synthesis results. The FFT results were

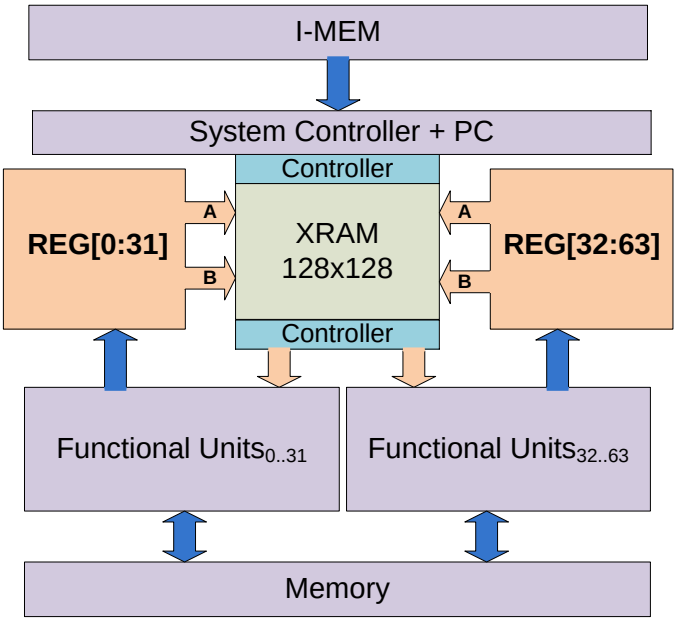


Figure 7.7: FFT Engine

verified with a test chip fabricated in TSMC 65nm and tested.

7.3.1 FFT Engine

Signal processing applications have traditionally been shown to benefit from scaling of processing elements. These types of algorithms are able to utilize massive amounts of data level parallelism (DLP) especially in SIMD style machines. In order to fully utilize the SIMD computational units and provide constant data, there needs to be sufficient support for permutations between the SIMD lanes. A SIMD engine was synthesized with a swizzle network implemented using MUX based crossbar and XRAM network. We pick the most classic signal processing algorithm, FFT, to show how the XRAM helps us build higher performance FFT accelerators by efficiently utilizing more processing elements. Figure 7.7 shows configurations for the FFT ac-

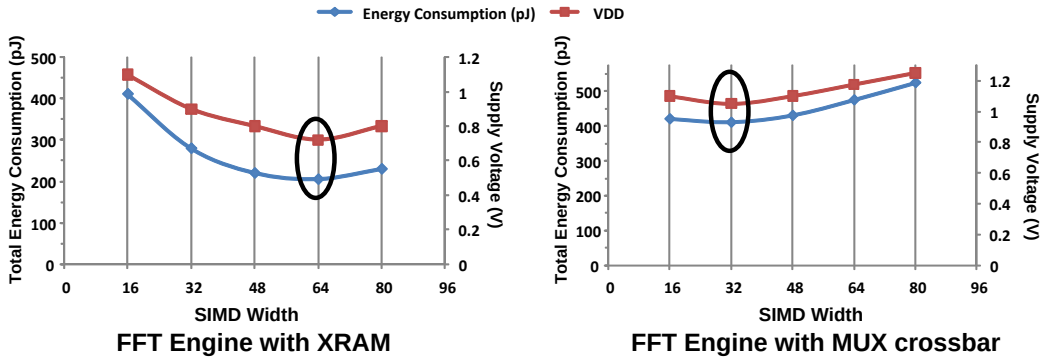


Figure 7.8: FFT results

celerator with SIMD width of 64. One reason we showcase the FFT is because the number of permutations required can fit within the embedded SRAM configuration at the cross points for a 128 port XRAM. XRAM's scalability comes at the cost of flexibility in that it can only store a certain number of configurations at a time and FFT is a perfect candidate for how this flexibility problem can be managed.

Figure 7.8 shows the results of implementing a SIMD processor that accelerates FFT with an XRAM and also with the MUX. The XRAM enables us to build accelerators which contain many more processing elements running at a lower voltages and thus lower frequency and perform at iso-throughput compared to narrower designs while consuming less total energy. At small numbers of processing elements, less than 32 wide, the crossbar is not a major energy dissipator compared to the execution units. As we start scaling the SIMD width, we start seeing the MUX based crossbar slowly dominate the energy dissipation, where at a SIMD width of 64, the energy of the MUX is equal to the energy of the multiplier. At SIMD widths greater than 64, the MUX based networks are consuming 3 times more energy than the multipliers.

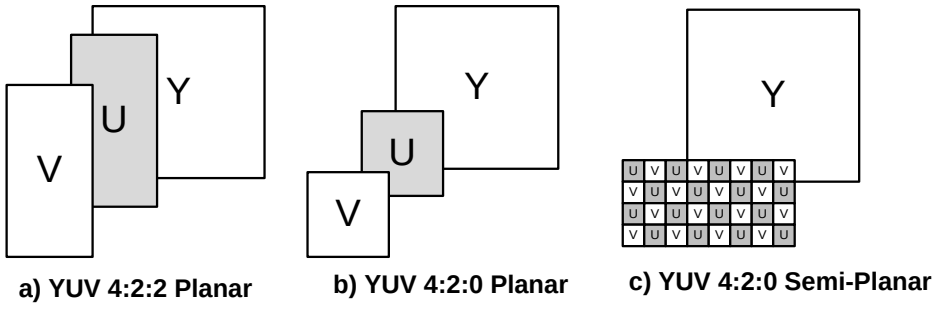


Figure 7.9: Examples of different YUV modes. In each YUV mode the Y plane is always written out to memory then the U plane followed by the Y plane. In YUV 4:2:0 semi-planar, the U and V data are interleaved together. The typical size of the Y plane is 16x16 pixels.

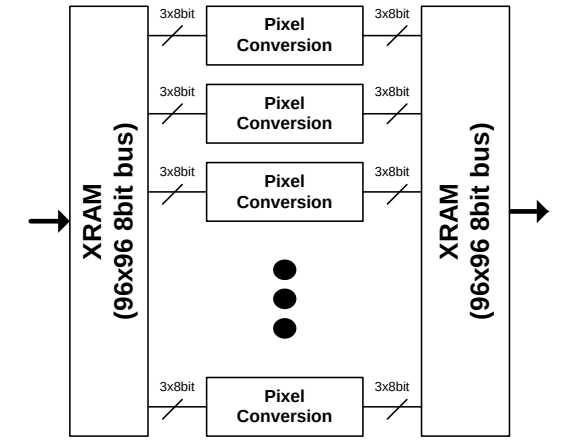
At larger SIMD widths, the MUX based crossbar not only dominate the energy dissipation, but the delay of the MUX itself becomes equal to the delay of the multiplier which reduces performance. On the other end, the XRAM based crossbar scales extremely well with increasing SIMD width. Because the XRAM uses sense amplifiers to speed up its operation, the XRAM does not contribute significant amount to the overall delay. The XRAM enables us to design accelerators with massive SIMD lanes. This allows us not only to design high performance FFT accelerators but also allows us to voltage scale and lower the frequency in order to reduce the total power consumption providing the same throughput as narrower SIMD based accelerators while consuming almost 50% less energy than the MUX based counterpart.

7.3.2 Color-space Conversion Hardware

Many smartphones and portable internet devices are based around systems like the Qualcomm Snapdragon [71] and ARM Mali [6] which integrate together a Mobile

GPU and high definition video engine (HDVE). These systems typically are design where the HDVE decodes the video stream data and the mobile GPU performs video overlay of the menu system and other graphical interactions and post processing. Problems arise between the communication between the HDVE and GPU when we start going to high definition standards like 1080p. This is because in these systems, interconnect bandwidth and memory storage are scarce and expensive commodities, so the HDVE and GPU in these systems read and write out YUV data which is a form of compressed pixel data. The problem is that current mobile GPUs operate on RGB data only [39]. In order to use YUV data, the GPUs have to do color-space conversion on the YUV output of the HDVE and convert the data into RGB for subsequent processes. At this point the GPU can output the image directly to the display or perform another color-space conversion to compress the RGB data back into YUV data. Typically in desktop GPUs from Nvidia and ATI, power consumption is in the hundreds of watts, so the number of streaming processors is extremely large, typically in the hundreds. Performing color-space conversions for post processing on these GPU utilizes very little of the total resources. In mobile GPUs like Imagination Technologies' PowerVR SGX [85] or ARM Mali there are less than tens of stream processors because the power budget is in the hundreds of mW. Performing color-space conversion on mobile GPUs can saturate the whole GPUs processing power and for video modes like 1080p can not even be completed in real time, utilizing more than 100% of the GPU.

In the literature, accelerators for color conversion have been designed to deal with



	XRAM	MUX Based
Delay	1.001ns	1.727ns
Energy Consumption	64.7pJ	261pJ
Power	64.6mW	151.1mW

```
Y = ( (  66 * R + 129 * G +  25 * B + 128) >> 8) +  16
U = ( ( -38 * R -  74 * G + 112 * B + 128) >> 8) + 128
V = ( ( 112 * R -  94 * G -  18 * B + 128) >> 8) + 128
```

RGB to YUV conversion

```
C = Y - 16     R = clip(( 298 * C           + 409 * E + 128) >> 8)
D = U - 128    G = clip(( 298 * C - 100 * D - 208 * E + 128) >> 8)
E = V - 128    B = clip(( 298 * C + 516 * D           + 128) >> 8)
```

YUV to RGB conversion

Operation of the Pixel Conversion Unit

Figure 7.10: Color Space Conversion Hardware

this issue but these systems only support a specific set of YUV modes like YUV420 and YUV422 [38]. One reason that they do this is because of how the pixel data in the different formats is laid out. Figure 7.9 shows a few different color-space modes. As you can see the pixel data itself can be in many different places but for a given input color-space and output color-space there is only one configuration needed in the XRAM. This can be programmed in once and performed multiple times making the one cycle to program the configuration negligible. This causes the color-space conversion accelerators to add increased amounts of contention into the interconnect system, requesting different memory locations. Figure 7.10 shows our implementation of a color space conversion accelerator that can reduce the number of memory request to the bus while also increasing the performance. The main enabler to build this type

of color space conversion is the XRAM. This is because we need to buffer a larger number of pixels and then perform multicasting and permutations based on the color compression scheme. One example is we first convert YUV 4:2:0 to YUV 4:4:4 then perform the YUV to RGB conversion. After we buffer pixel data, we can efficiently process the pixels and convert them to the corresponding color-space. If we used a MUX based system, the power consumption, area and delay of the MUX would dominate the accelerator. The increase incurred would lead us to an alternative solutions where we would just add more stream processors to the GPU rather than use a color space conversion accelerator. Figure 7.10 shows that compared to MUX based implementation of the color space conversion accelerator, we are able to perform 1.7x faster, and consume almost 2.5x less power.

7.4 Usage Models for XRAM

7.4.1 Multi-core Interconnect

On-chip Networks (NoCs) are becoming the bottleneck and limiting factor in the number of cores that can be scaled in CMP systems. One of the most critical problems with scaling NoCs is the power consumption. The majority of power of these multi-core NoCs is in input and internal buffers between links and the physical crossbar link itself [92]. One way to decrease the buffer power, which is more than 50% in some cases, is by reducing the number of stages. Building larger crossbars between the stages will help solve this problem. Currently, the stages consists of multiple

bidirectional switches connecting to each of its neighbors and itself. We can scale up the number of ports in each stage using XRAM while still achieving the same link speed. Internal buffering power will be reduced and also the physical crossbar link power will be reduced using XRAM as compared to MUXes. This requires an XRAM with conflict detection and avoidance. Such a design is left as future work. For a large number of ports, arbitration needs to be resolved quickly or it will become the latency bottleneck.

7.4.2 Memory Interconnect

The number of memory banks and the number of outstanding memory requests are increasing in order to increase memory bandwidth. There are many more memory banks than there are processing elements. The latency of current networks is too large, leading to performance degradation when they are used to access on-chip memory. XRAM can provide a fast interconnect between the large number of memory banks and less number of processing elements. Again, an XRAM with a conflict detection and avoidance is needed.

7.4.3 Reliability

The XRAM can help us design more reliable multi-core systems using techniques like those used in StageWeb [33]. Traditionally reliability issues have been are addressed in a coarse granularity by disabling faulty cores in multicore systems. These permanent hardware failures were caused by wearout and process variation. Because

failure rates are increasing as we scale technology nodes, coarse grain reliability would cause you to potentially throw away the whole die because certain components of different cores were broken. StageWeb solves this by having an interconnect connecting the previous stage of a core to the next stage of all neighboring cores. The limiting factor in StageWeb is the number of cores that can be connected. After 6 cores, the interconnect MUXes start to impact the performance and the network adds 2 cycles between each stage just for the interconnect. Using XRAM would allow StageWeb to connect more cores, improving reliability and actually improving performance because of the better delay performance of the XRAM.

Another technique that XRAM can provide is reliability of SIMD processors. As we stated because Moore's Law is providing us more transistor density. Even though transistor density is increasing the power density also increasing limiting us from actually utilizing all the cores. To solve this problem Near Threshold Computing has been proposed. This technique scales the voltage giving a quadratic reduction in power but at the same time decreasing the frequency. In order to make up the performance loss increasing the number of SIMD lanes is needed. The problem is that process variation will start to become a big problem because of the increasing unreliability of future process technologies, increase timing variation due to voltage fluctuations, and general process variation with increase in SIMD lanes. XRAM can help solve this problem by allowing a cheap way to allow redundant lanes.

7.5 Conclusion

In this chapter, we introduced a low power and scalable crossbar design, XRAM, which provides a solution to the bandwidth and scaling problem seen in traditional interconnect implementations. XRAM implements novel circuit techniques to solve the scaling problem while providing high bandwidth. Unlike other interconnect networks like Benes and Banyan Networks, the XRAM is able to perform all permutations and swizzle operations including multicasting and broadcasting in a non-blocking fashion. One of the circuit techniques solves the control complexity by embedding the cross point control within the cross points of a matrix style crossbar using SRAM cells rather than having it driven by an external controller. Other circuit techniques like using the input bus to program the control helps reduce the number of control wires needed within the XRAM. Finally borrowing techniques that are currently used in SRAM arrays helps improve performance and lower the energy used in driving the wires of the XRAM. Although these techniques helps solve the performance and scaling problem of traditional interconnects, one drawback is with flexibility; the XRAM is only able to store a certain number of swizzle configurations at a given time. These configurations are not static and can be changed and reprogrammed at run-time. Though this seems like a major drawback, we present case studies for many applications where this is not a bottleneck and also architectural techniques that mitigate this flexibility problem. We showed that compared to conventionally implemented crossbars, the area scales linearly with the number of input*output ports while consuming almost 50% less energy. We present two different application case

studies using XRAM and show that the XRAM enabled gain of 1.4x on color-space conversion and between 1.5-2.5x less power for applications like FFT and color-space conversion.

CHAPTER 8

AnySP: Anytime Anywhere Anyway Signal Processing

Previous chapters have characterized a 4G wireless protocol and high definition mobile video, and provided architectural insights into achieving the computational efficiency needed in mobile devices. After understanding the workings of mobile signal processing systems and their algorithms, an architecture can be designed. This chapter proposes *AnySP*—an example of an advanced signal processor architecture that targets the next generation mobile computing. Our objective is to create a fully programmable architecture that supports 4G wireless communication and high-definition video coding at efficiency levels of 1000 Mops/mW. Programmability is recognized as a first class design constraint, thus no hardwired ASICs are employed. To address the issues of computational efficiency and adaptivity, we introduce a configurable single-instruction multiple-data (SIMD) datapath as the core execution engine of the system. The datapath supports three execution scenarios: wide vector computation

(64 lanes), multiple independent narrow vector computation threads (8 threads × 8 lanes), and 2-deep subgraphs on moderate wide vector computation (32 lanes × depth 2 computations). This inherent flexibility allows the datapath to be customized to the application, but still retain high execution efficiency. *AnySP* also attacks the traditional inefficiencies of SIMD computation: register file power, data shuffling, and reduction operators.

The key contributions of *AnySP* are as follows:

- An in-depth analysis of the computational characteristics of 4G wireless communication and high-definition video algorithms.
- The design, implementation, and evaluation of *AnySP*—an example programmable processor that targets next generation mobile computing.
- The design and organization of a configurable SIMD datapath for sustaining high throughput computing with both wide and narrow vector widths.

8.1 Mobile Signal Processing

8.1.1 Overview of Benchmarks

8.1.1.1 4G Wireless Protocol

We model our 4G wireless system after the NTT DoCoMo test 4G wireless system [84], because there is no standard yet for 4G. The major components of the physical layer consists of three blocks: a modulator/demodulator, a MIMO encoder/decoder,

Algorithm	SIMD Workload (%)	Scalar Workload (%)	Overhead Workload (%)	SIMD Width (Elements)	Amount of TLP
FFT/IFFT	75	5	20	1024	Low
STBC	81	5	14	4	High
LDPC	49	18	33	96	Low
Deblocking Filter	72	13	15	8	Medium
Intra-Prediction	85	5	10	16	Medium
Inverse Transform	80	5	15	8	High
Motion Compensation	75	5	10	8	High

Table 8.1: Data level parallelism analysis for MSP algorithms. Overhead workload is the amount of instructions needed to aid the SIMD operations like data shuffle and SIMD load/store.

and a channel encoder/decoder. These blocks constitute the majority of the 4G computations [90].

The role of the modulator is to map data sequences into symbols with certain amplitudes and phases, onto multiple orthogonal frequencies. This is done using inverse FFT. The demodulator performs the operations in reverse order to reconstruct the original data sequences. The MIMO encoder multiplexes many data signals over multiple antennae. The MIMO decoder receives all the signals from the antennae and either decodes all the streams for increased data rates or combines all the signals in order to increase the signal strength. The algorithm used to increase data rate is the vertical Bell Laboratories layered space-time (V-BLAST), and the algorithm used to increase the signal quality is the space time block coding (STBC). Finally, the channel encoder and decoder perform forward error correction (FEC) that enables receivers to correct errors in the data sequence without retransmission. There are many FEC algorithms that are used in wireless systems. LDPC is used because it supports the highest data rates. Our target for 4G wireless is the maximum data rate for high mobility, which is 100Mbps. This 4G configuration utilizes the FFT, STBC, and

LDPC kernels.

8.1.1.2 H.264 Video Standard

Video compression standards, such as MPEG-4 and H.264, are being actively considered for mobile communication systems because of the increasing demand for multimedia content on handheld devices. H.264 is selected as the multimedia benchmark because it achieves better compression compared to previous standards and also contains most of the basic functional blocks (prediction, transform, quantization, and entropy decoding) of previous standards. Of the three profiles in H.264, we focused on the *Baseline* profile due to its potential application in videotelephony, and videoconferencing.

The H.264 decoder receives a compressed bitstream from the network abstract layer (NAL). The first block is the entropy decoder which is used to decode the bitstream. After reordering the stream, the quantized coefficients are scaled and their inverse transform is taken to generate the residual block data. Using header information in the NAL, the decoder selects prediction values for motion compensation in one of two ways: from a previously decoded frame or from the filtered current frame (intra-prediction). According to the power profile of H.264, about 75% of the decoder power consumption is attributed to three algorithms: deblocking filter (34%), motion compensation (29%), and intra-prediction (10%) [53]. Therefore, these three algorithms are selected as the H.264 kernel algorithms in this study.

8.1.2 Algorithm Analysis

To build a unified architecture for mobile signal processing (MSP), we performed a detailed analysis of the key kernel algorithms. These are FFT, STBC, and LDPC for the wireless workload, and deblocking, intra-prediction, inverse transform, and motion compensation for the video processing workload. Our goal was to explore the characteristics of each algorithm in order to find their similarities and differences. This section will discuss the studies that helped define our example architecture for MSP.

8.1.2.1 Multiple SIMD Widths

Many previous studies have analyzed and determined the best SIMD width that should be implemented for general-purpose and application-specific workloads. The optimal SIMD width is different in each study because width depends on the algorithms that constitute the workload. Examples are the 2-8-wide Intel MMX extensions [68], the 16-wide NXP EVP [88], and the 32-wide SODA [54].

Table 8.1 presents our study analyzing the data level parallelism (DLP) of MSP algorithms. We calculate the available DLP within each of the algorithms and show the maximum natural vector width that is achievable. The instructions are broken down into 3 categories: SIMD, overhead, and scalar. The SIMD workload consists of all the raw SIMD computations that use traditional arithmetic and logical functional units. The overhead workload consists of all the instructions that assist SIMD computations, for example loads, stores and shuffle operations. The scalar workload

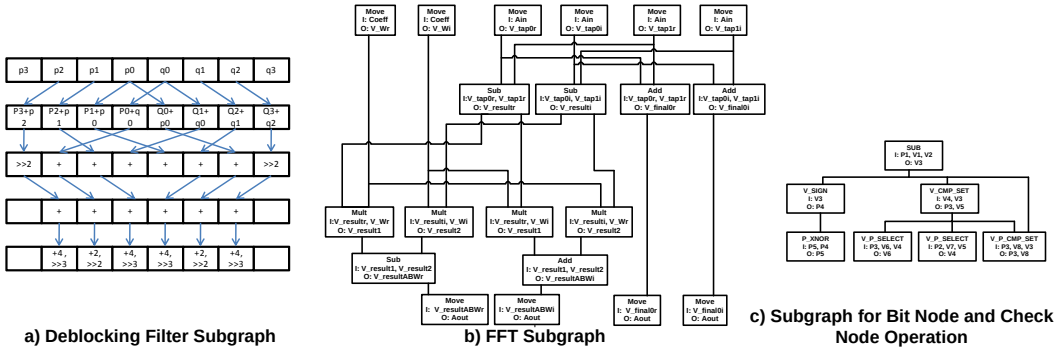


Figure 8.1: Select subgraphs for the inner loops of MSP algorithms.

consists of all the instructions that are not parallelizable and must be run on a scalar unit or on the address generation unit (AGU).

From Table 8.1, we see that many of the algorithms have different natural vector widths–4, 8, 16, 96, 1024. Also, the algorithms with smaller SIMD widths exhibit a high level of TLP, which means that we can process multiple threads that work on separate data on a wide SIMD machine. For instance, 8 instances of STBC that have SIMD width of 4 can be processed on a 32-wide machine. Unlike most SIMD architectures that are designed with a fixed SIMD width to process all the algorithms, this study suggests that the best solution would be to support multiple SIMD widths and to exploit the available thread-level parallelism (TLP) when the SIMD width is small. By supporting multiple SIMD widths, the SIMD lane utilization can be maximized as will be demonstrated in Section 8.2.1.1.

Though the scalar and overhead workloads are not the majority, they still contribute 20-30% of the total computation. In instances such as LDPC, comprised of mostly simple computations, data shuffling and memory operations dominate the

majority of the workload. This suggests that we cannot simply improve the SIMD performance, but also must reduce the overhead workload. This can be accomplished by introducing better support for data reorganization or by increasing the scalar and AGU performance. Table 8.1 also shows how much TLP is available; high TLP means that many independent but identical threads can be spawned and low means only one or a few can be spawned mainly due to data dependencies or absence of outer loops.

8.1.2.2 Register Value Lifetimes

For many processors, the register file (RF) contributes a large percentage of the power consumption. Some architectures have RFs that consume 10-30% of the total processing element (PE) power [31] [66]. This is due to the reading and writing of values for each instruction. Many architectures, like very large instruction word (VLIW), compound this problem by having multi-ported RFs that increase the energy per access. Recent works in [28] and [24] show that performance and power can be reduced by bypassing the RF for short-lived values and storing the value within a register bypass pipeline. The bypass network is a non-critical path structure for holding values a fixed number of cycles before they are consumed. By bypassing the RF, access power and register pressure are reduced allowing for more effective register allocation. Another technique used to reduce register file power is register file partitioning. Previous research found that for some embedded applications, a small set of registers accounted for the majority of the register file accesses [31]. By splitting the register file into a small and large region, the high access registers can

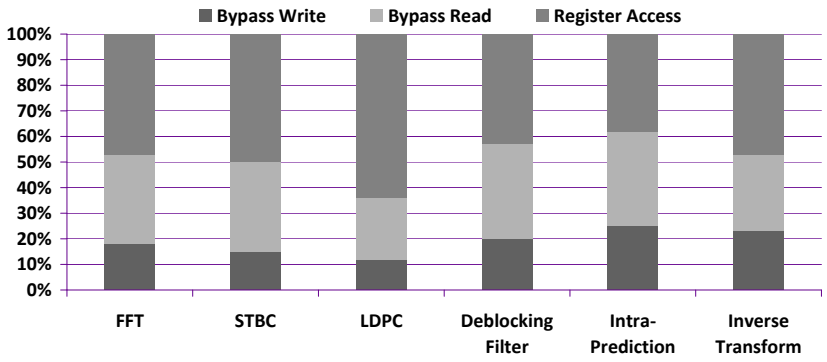


Figure 8.2: Register file access for the inner loops of MSP algorithms. Bypass write and read are the RF read and write accesses that do *not* need to use the main RF. Register accesses are the read and write accesses that *do* need to use the main RF.

be stored in the small partition resulting in less energy consumed per access.

We studied the potential for eliminating RF accesses in MSP applications. Figure 8.1 shows the subgraphs for several of the MSP inner loops. We see that there exists large amounts of data locality in these subgraphs, which is illustrated by the number of single lines that go from one source to one destination. The MSP applications fit the streaming dataflow model, where data is produced and then read once or twice. The values are usually consumed by the instructions directly succeeding the producer. The number of temporary registers needed are small but are accessed often suggesting a register bypass network or a partitioned RF can reduce power and increase performance.

We further analyze the MSP algorithms to determine the extent to which RF accesses can be reduced. We use the SODA architecture [54] with a small four entry RF partition modification creating a total of 20 registers: 16 main registers and 4 temporary registers. Figure 8.2 shows the breakdown of accesses that can and cannot

bypass the RF. *Bypass write* and *bypass read* correspond to the RF read and write accesses that do not need to use the main RF and can be eliminated with either data forwarding or by storing the data in a small RF partition. *Register accesses* are the read and write accesses that cannot be bypassed and need to use the main RF. For most of the algorithms a large portion of register accesses can be bypassed. The exception to this is LDPC where many values are calculated and stored for a several cycles before one of them is chosen. Based on our analysis, we conclude that architectures that run MSP algorithms should support bypass mechanisms to help reduce the accesses to the main RF.

8.1.2.3 Instruction Pair Frequency

In typical digital signal processing (DSP) codes, there are pairs of instructions that are frequently executed back to back. A prime example of this is the multiply followed by accumulate instruction. To improve performance and decrease energy consumption, DSP processors such as the Analog Devices ADSP and the TI C series introduced the multiply-accumulate (MAC) operation. The work in [18] has exploited this further by building specialized hardware to exploit long chains of instructions.

We performed a study on the MSP algorithms to find the most common producer-consumer instruction pairs. In each case the producer instruction's value, once consumed, is not referenced again. This allows the producer and consumer instructions to be fused. Figure 8.3 shows the breakdown of the most frequent instruction pairs that were found in MSP algorithms programmed for SODA [54]. Among all the algo-

	Instruction Pair	Frequency	Total
1	multiply-add	26.71%	26.71%
2	add-add	13.74%	40.45%
3	shuffle-add	8.54%	48.99%
4	shift right-add	6.90%	55.89%
5	subtract-add	6.94%	62.83%
6	add-shift right	5.76%	68.59%
7	multiply-subtract	4.14%	72.73%
8	shift right-subtract	3.75%	76.48%
9	add-subtract	3.07%	79.55%
10	Others	20.45%	100.00%

a) Intra-prediction and Deblocking Filter Combined

	Instruction Pair	Frequency	Total
1	shuffle-move	32.07%	32.07%
2	abs-subtract	8.54%	40.61%
3	move-subtract	8.54%	49.15%
4	shuffle-subtract	3.54%	52.69%
5	add-shuffle	3.54%	56.23%
6	Others	43.77%	100.00%

b) LDPC

	Instruction Pair	Frequency	Total
1	shuffle-shuffle	16.67%	16.67%
2	add-multiply	16.67%	33.33%
3	multiply-subtract	16.67%	50.00%
4	multiply-add	16.67%	66.67%
5	subtract-mult	16.67%	83.33%
6	shuffle-add	16.67%	100.00%

c) FFT

Figure 8.3: Instruction pair frequencies for different MSP algorithms.

rithms, the MAC instruction was the most frequent, because a large number of these algorithms have a dot product as their inner loop. Other fused instruction pairs that were common were permute-add, add-add, and shift-add. The permute-add pair occurs in many of the algorithms because data must first be aligned to effectively utilize the SIMD lanes. The add-add and shift-add pairs also occur quite frequently in the video algorithms, because values are accumulated together and then normalized by a power of 2 division—a shift.

Based on this analysis, we conclude that the fused instruction pairs with high frequency should be supported. This increases performance and decreases power because register access between the producer and consumer instructions would no longer be needed.

8.1.2.4 Algorithm Data Reordering Patterns

Most commercial DSP and general purpose processor(GPP) multimedia extensions support some form of data reordering operations. By data reordering we refer to both permutation and replication of data. In graphics terminology this is referred

to as swizzling. Swizzle operations are particularly important in SIMD machines where they are essential to keep the functional units utilized. For example, Intel's Larrabee [82], which supports 16-wide SIMD operations, has a hardware "gather and scatter" instruction that allows 16 distinct memory addresses to build one SIMD vector. This is one extreme case where the gather and scatter operations can take multiple cycles and consume significant power if a number of distinct cache lines have to be accessed. At the other end of the spectrum is the data "shuffle" network in SODA [54] where the programmer needs to align the data within 32-wide SIMD blocks often requiring multiple shuffle instructions. This consumes less power than Intel's Larrabee but can take multiple cycles to complete.

We performed a study to enumerate the number of swizzle operations required for the MSP algorithms. Figure 8.4 shows a few of the common examples. The key finding was that all of the algorithms had a predefined set of swizzle operations, typically below 10 that were known beforehand. Because all the swizzle operations were known beforehand, sophisticated gather and scatter hardware support like that of Larrabee is not needed. However, a shuffle network like that of SODA, is not complex enough to support the needed swizzle operations. An example is LDPC where the number of predetermined swizzle operations(in this case permutations) were large. In LDPC we need a mechanism to change the swizzle operations based on a parity check matrix. In SODA, the data shuffle network may require many cycles to complete these task becoming a bottleneck in the performance of LDPC. A network which can be easily configured and also perform the operations in fewer cycles is desirable. Either a

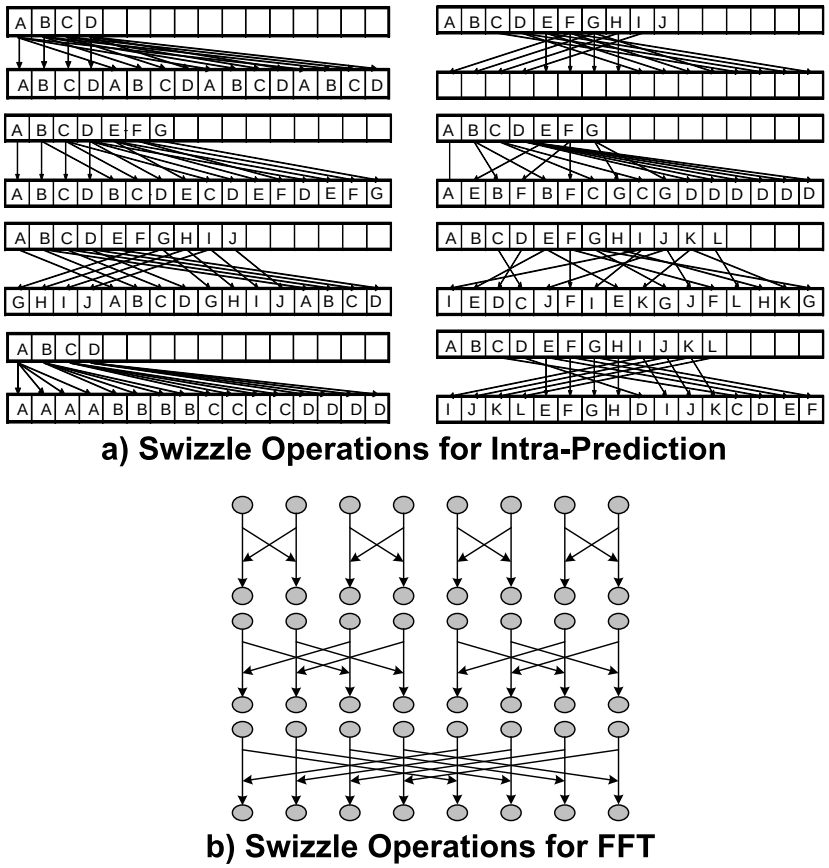


Figure 8.4: A set of swizzle operations that are required for a subset of MSP algorithms.

swizzle network or a more complex programmable fixed pattern network would be the best solution.

8.1.2.5 Design Study Summary

The result of these studies provided us with four key insights that will be exploited in the next section to design an efficient high-performance architecture for MSP applications:

- Most algorithms have small natural vector widths and large amounts of TLP

- A large percentage of register values are short-lived and many do not need to be written to the register file
- A small set of instruction pairs are used a large percentage of the time
- Each algorithm uses a small set of predetermined swizzle patterns

8.2 AnySP Architecture

In this section, we describe the AnySP architecture which exploits the MSP algorithm features described in the previous section.

8.2.1 AnySP PE Design

The PE architecture is shown in Figure 8.5. It consists of SIMD and scalar datapaths. The SIMD datapath in turn consists of 8-groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32 and 64. Each of the 8-wide SIMD units are composed of groups of Flexible Functional Units (FFUs). The FFUs contain the functional units of two lanes which are connected together through a simple crossbar. The SIMD datapath is fed by eight SIMD register files (RFs). Each RF is 8 wide and has 16 entries. The swizzle network aligns data for the FFUs. It can support a fixed number of swizzle patterns of 8, 16, 32, 64 and 128 wide elements. Finally, there is a multiple output adder tree that can sum of groups of 4, 8, or 16 elements, and store the results into a temporary buffer.

The local memory consists of 16 memory banks; each bank is an 8-wide SIMD containing 256 16-bit entries, totalling 32KB of storage. Each 8-wide SIMD group has a dedicated AGU unit. When not in use, the AGU unit can run sequential code to

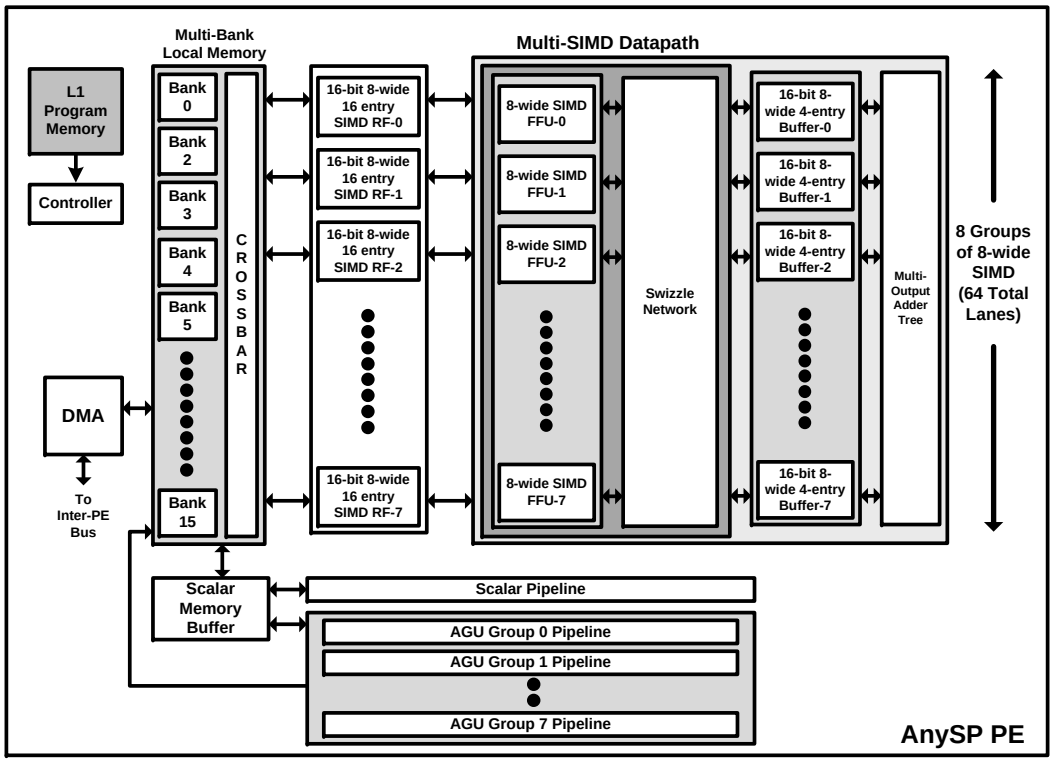


Figure 8.5: AnySP PE. It consists of SIMD and scalar datapaths. The SIMD datapath consists of 8-groups of 8-wide SIMD units, which can be configured to create SIMD widths of 16, 32 and 64. Each of the 8-wide SIMD units are composed of groups of Flexible Functional Units (FFUs). The local memory consists of 16 memory banks; each bank is an 8-wide SIMD containing 256 16-bit entries, totalling 32KB of storage.

assist the dedicated scalar pipeline. The AGU and scalar unit share the same memory space as the SIMD datapath. To accomplish this, a scalar memory buffer that can store 8-wide SIMD locations is used. Because many of the algorithms access data sequentially, the buffer acts as a small cache which helps to avoid multiple accesses to the vector banks. Details about each of these architectural features is discussed in the rest of this section.

8.2.1.1 Configurable Multi-SIMD Width Support

We analyzed the performance of MSP algorithms on existing SIMD architectures like [54][88] and found that the full SIMD widths could not always be utilized. In many of the algorithms the natural vector width is smaller than the processors's SIMD width, leading to wasted lanes, and power or performance penalties. In order to improve SIMD utilization, code transform work, such as [55], tried to transform the algorithms to fit the SIMD width. This was achieved at the cost of increased total computation and complex code transformations. These techniques should be avoided because of their complexity and increase in power consumption from additional pre-processing requirements.

In Section 8.1.2.1, we showed that many of the algorithms had different natural vector widths. While the small SIMD width kernels had large amounts of TLP, the ones with higher DLP showed little TLP. An interesting insight was that independent threads were not the only contributors to TLP. Specifically, in H.264, the same task would be run many times for different macroblocks. Each task was independent of each other and ran the exact same code and followed almost the same control path. The difference between each thread was that the data accesses from memory were different. This meant that for each thread, separate memory addresses would have to be computed. In SODA, the only way we were able to handle these threads was to execute them one at a time on the 32-wide SIMD. This is inefficient because these algorithms had widths smaller than the SIMD width. In order to support these types of kernel algorithms, we chose to design a multi-SIMD width architecture. Each group

of 8-wide SIMD units has its own AGU to access different data. The 8-wide groups can also be combined to create SIMD widths of 16, 32 or 64. Such a feature allows us to exploit the DLP and TLP together for large and small SIMD width algorithms. Small SIMD width algorithms like intra-prediction and motion compensation, can process multiple macroblocks at the same time while exploiting the 8-wide and 16-wide SIMD parallelism within the algorithms. Large SIMD width algorithms like FFT and LDPC, can use the full 64-wide SIMD width and run multiple iterations.

8.2.1.2 Temporary Buffer and Bypass Network

As discussed in Section 8.1.2.2, the RF is one of the major sources of power consumption in embedded processors. We implemented two different techniques in order to lower RF power: temporary register buffers and a bypass network.

The temporary register buffers are implemented as a partitioned RF. The main RF still contains 16 registers. A second partition containing 4 registers is added to the design, making the total number of registers 20. This small partitioned RF shields the main RF from accesses by storing values which have very short lifetimes. This saves power because the smaller, lower power RF is accessed multiple times rather than the main, higher power, RF. Another benefit is that by reducing accesses to the main RF, register file pressure is lessened, thereby decreasing memory accesses.

The bypass network is a modification to the writeback stage and forwarding logic. Typically in processors, data that is forwarded to another instruction to eliminate data hazards is also written back to the RF. In the bypass network, the forwarding

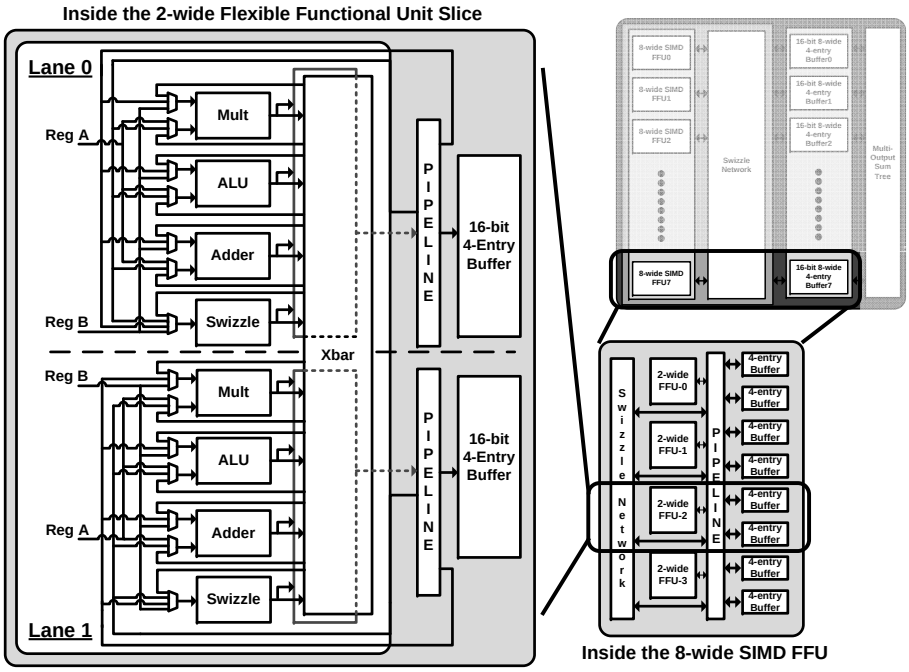


Figure 8.6: Flexible Functional Unit

and writing to the RF is explicitly managed. This is similar to TTA [20], which defines all input and output ports of all modules as registers and the programmer explicitly manages where the data should go. However, in our design we only allow the bypass network to be visible to the programmer. This means that the instruction dictates whether the data should be forwarded and whether the data should be written back to the RF. This eliminates RF writes for values that are immediately consumed thus reducing RF power.

8.2.1.3 Flexible Functional Units

In typical SIMD architectures, power and performance is lost when the vector size is smaller than the SIMD width due as a result of underutilized hardware. The

proposed flexible functional units (FFU) allow for reconfiguration and flexibility in the operation of different types of workloads. When SIMD utilization is low, the FFUs are able to combine the functional units between two lanes, which can speed up more sequential workloads that underutilize the SIMD width by exploiting pipeline parallelism. This effectively turns two lanes of the SIMD into a 2-deep execution pipeline. Two different instructions can be chained through the pipeline, and data can be passed between them without writing back to the RF. Ultimately, this allows two instructions per chained FFU to be in flight per cycle. Using instruction chaining we are able to extract different levels of parallelism like pipeline parallelism and DLP within the same FFU unit.

As shown in Figure 8.6, each 8-wide SIMD group is built from four 2-wide FFUs. Each functional unit consists of a multiplier, ALU, adder and swizzle network sub-block. Such a structure benefits algorithms with SIMD widths that are smaller than 64. In chained execution mode, the functional units among two internal lanes can be connected together through a crossbar network. Overall, FFUs improve performance and reduces power by adding more flexibility and exploiting both TLP and DLP.

8.2.1.4 Swizzle Network

Our analysis in Section 8.1.2.4 showed that the number of distinct swizzle patterns that are needed for a specific algorithm is small, fixed, and known ahead of time. Previous research has discussed building application specific crossbars for SIMD processors [72]. Such designs hardwired only the swizzle operations needed to support

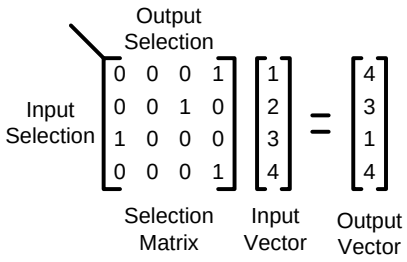


Figure 8.7: The swizzle operation in computer graphics. The rows of the selection matrix correspond to the input vector and the columns correspond to the output vector selected. Each row can only have a single 1, but columns can have multiple 1s which allow for selective multi-casting of the data.

the application, yielding a power efficient design. However, they lack flexibility as they are unable to support new swizzle operations for different applications post fabrication.

We propose using an SRAM-based swizzle network that adds flexiblity while maintaining the performance of a customized crossbar. Figure 8.7 illustrates how the swizzle operation works. If the *(i,j)*th entry of the selection matrix is 1, then input i is routed to output j.

The SRAM-based swizzle network is similar to the layout of [30], where the layout is an X-Y style crossbar, where the input buses are layed out horizontally and the outputs are layed out vertically. Each point of intersection between the input and output buses contains a pass transistor which is controlled by a flip flop. This allows for more flexiblity in that configurations can be changed post fabrication. However, this design has three drawbacks. First, each output is directly driven by the input. As the crossbar size increases, the capacitance of the output lines increase requiring input drivers to be larger, resulting in a higher power consumption. Second, each time a different

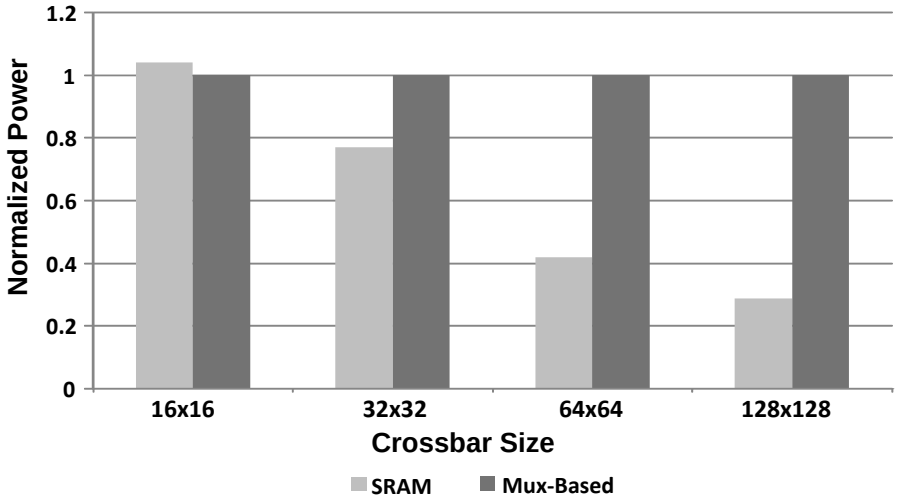


Figure 8.8: Comparison of power between the SRAM-based swizzle network and MUX-based crossbar. The values are normalized to the MUX-based crossbar for different crossbar widths with 16-bit data inputs. These results correspond to a switching activity of 0.50.

permutation is needed, the flip flops have to be reprogrammed resulting in additional cycles. Third, a large number of control wires is required for reprogramming the flip flops increasing power consumption and complicating the layout.

To solve these short comings, we leverage an SRAM-based swizzle network in which multiple SRAM cells replace the single flip flop of [30]. First, by using circuit techniques, we decouple the input and output buses reducing the needed input driver strength and decreasing the overall power consumption while providing more scalability. Second, multiple sets of swizzle configurations can be stored into the SRAM cells, allowing zero cycle delays in changing the swizzle pattern. Third, by storing multiple configurations we remove a large number of control wires necessary.

Using the SRAM-based swizzle network, the area and power consumption of the network can be reduced while still operating within a single clock cycle. Figure 8.8

shows the power difference between the SRAM-based swizzle network and MUX-based crossbar for different crossbar sizes using 16-bit inputs. As we can see, for crossbar sizes larger than 32x32, the power of the SRAM-based swizzle network is dramatically lower than the MUX-based one and is able to run at almost twice the frequency.

This is one of the enabling technologies for very wide SIMD machines to be power efficient and fast. Though only a certain number of swizzle patterns can be loaded without reconfiguration, it is a viable solution since only a limited set of swizzle patterns need to be supported for each algorithm. These swizzle patterns can be stored into the SRAM-based swizzle network configuration memory at initialization. Because it supports arbitrary swizzle patterns with multi-casting capabilities, it functions better than previous permutation networks found in [54][91].

8.2.1.5 Multiple Output Adder Tree Support

SIMD architectures such as [54] have special SIMD summation hardware to perform "reduction to scalar" operations. To compute this, adder trees sum up the values of all the lanes and store the result into the scalar RF. These techniques worked for 3G algorithms but do not support video applications where sums of less than the SIMD width are needed. Other architectures capable of processing video applications like [7][74] have sum trees, but their SIMD widths were only 4-8 elements wide. For wide-SIMD machines like AnySP, we designed the adder tree to allow for partial summations of different SIMD widths, which then writeback to the temporary buffer unit. Writing the resultant sum into the temporary buffer unit is key, because many times

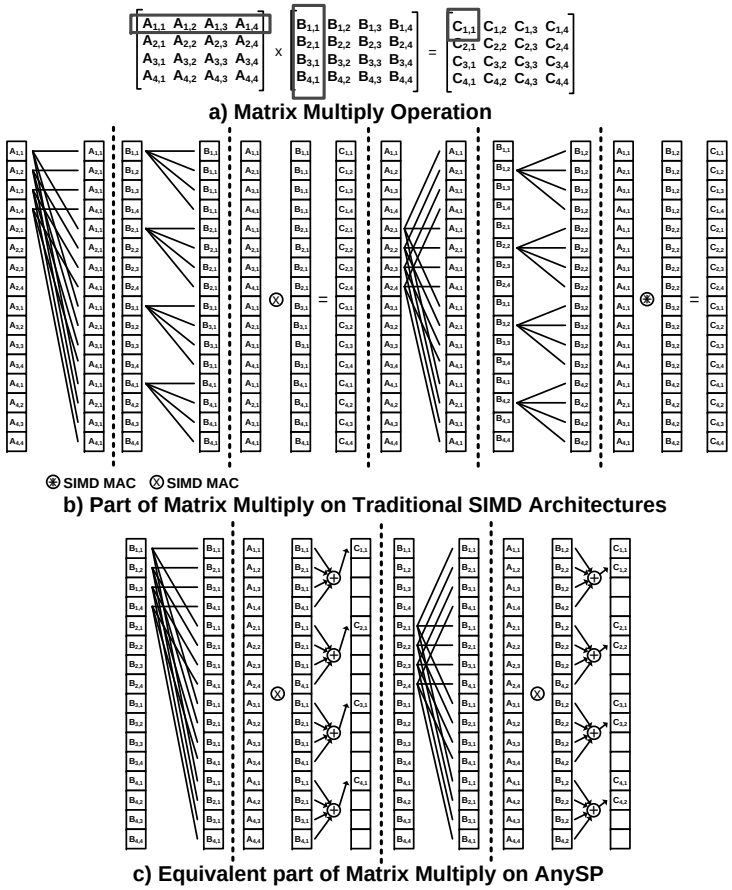


Figure 8.9: Demonstration of the output adder tree for matrix multiplication

the summed values have short lifetimes but are updated and used very frequently within that period.

In the 4G and video decoding algorithms, matrix calculations occur frequently. One of these calculations is matrix multiplication, shown in Figure 8.9a, and an example of doing a 4x4 matrix multiply on a typical SIMD architecture is shown in Figure 8.9b. This calculation requires many swizzle operations to align both matrices. Figure 8.9c shows how using an adder tree that adds only 4 numbers can reduce the total operations required. Here, we see that only one of the matrix values needs to

be swizzled. Since the adder tree only produces 4 of the 16 values per iteration, the adder tree uses the temporary register buffer to store the partial SIMD values in order to prevent wasted reading and writing to the RF. After 4 iterations, 16 sums are produced forming a complete SIMD vector that is then written back to the RF. AnySP supports sums of 4, 8, 16, 32 or 64 elements.

8.3 Results and Analysis

8.3.1 Methodology

The RTL Verilog model of the SODA processor [54] was synthesized in TSMC 180 nm technology, and the power and area results for 90 nm technology were estimated using a quadratic scaling factor based on Predictive Technology Model [40]. The main hardware components of the AnySP processor were implemented as RTL Verilog and synthesized in TSMC 90 nm using Synopsys physical compiler. The timing and power numbers were extracted from the synthesis and used by our in-house architecture emulator tool to calculate the timing and power values for each of the kernels. AnySP's PE area is 130% larger than SODA's estimated 90 nm PE area. Unlike SODA, where the target frequency was 400 MHz, AnySP was targeted at 300 MHz. The major kernels of the 100 Mbps high mobility 4G wireless protocol and high quality H.264 4CIF video are analyzed. The 4CIF video format is 704x576 pixels at 30 fps.

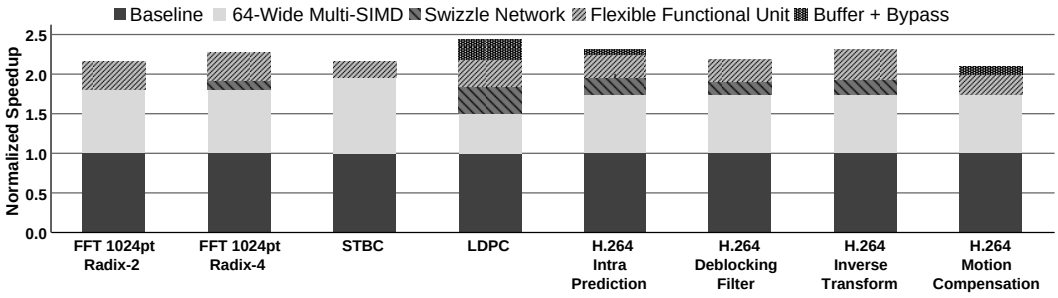


Figure 8.10: AnySP speedup over SODA for the key algorithms used in 4G and H.264 benchmarks. The speedup is broken down into the different architectural enhancements - wider SIMD width, single-cycle SRAM-based crossbar, fused-operations and buffer support

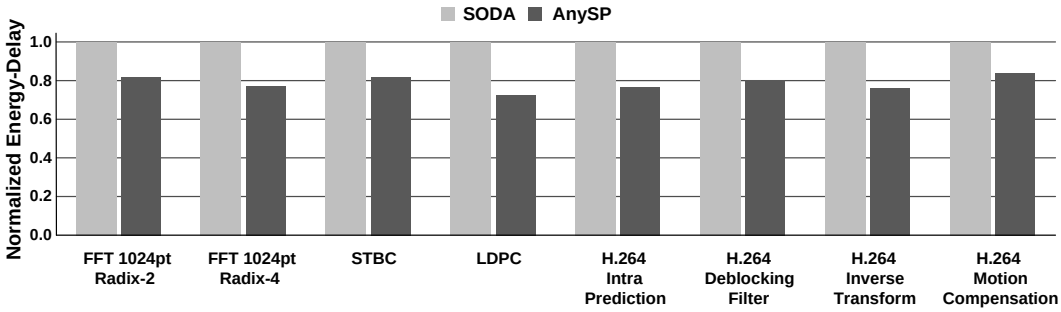


Figure 8.11: Normalized Energy-Delay product for each kernel algorithm

8.3.2 Algorithm-Level Results

In this section, we present the performance analysis of key algorithms in MSP, namely FFT, STBC and LDPC of 4G wireless protocols and intra-prediction, deblocking filter, inverse transform and motion compensation of an H.264 decoder.

The speedup of AnySP over SODA for each algorithm is shown in Figure 8.10. Each improvement is broken down by architectural enhancements: wider SIMD width (from 32 to 64), use of single-cycle SRAM-based swizzle network, fused operations, and temporary buffer with the bypass network. The energy-delay product for running

the kernel algorithms on SODA and AnySP are also presented in Figure 8.11. On average, AnySP achieves 20% energy-delay improvement over SODA with more than 2x speedup. A detailed analysis is presented next.

8.3.2.1 4G Algorithm Analysis

Fast Fourier Transform. Figure 8.10 provides results of two different FFT configurations: 1024 point Radix-2, and 1024 point Radix-4. On average, AnySP achieves a 2.21x speedup over SODA. A wider SIMD width is the main contribution of the speedup because FFT is a highly parallel algorithm. The radix-4 FFT algorithm has better performance compared to radix-2. 34% of the speedup is attributed to the fused-operations, such as shuffle/add. The radix-4 FFT algorithm takes more advantage of the single-cycle SRAM-based swizzle network compared to the radix-2 algorithms, because radix-4 algorithms require complex shuffle operations that cannot be done in a single cycle by other networks.

STBC. As shown in Table 8.1, STBC contains large amounts of DLP and TLP. Figure 8.10 shows that the wider SIMD width along with the multi-SIMD support helped speed up STBC by almost 100%. The rest of the speedup was obtained with the FFU, where the fused-operations contributed another 20% speedup. This was because fused pairs such as add-add, subtract-add, add-subtract, and subtract-subtract were used frequently in the algorithm. A combination of fused-operations, register buffers, and data forwarding bypass network added to an almost 20% decrease in the energy-delay product, making the algorithm run more efficiently compared to

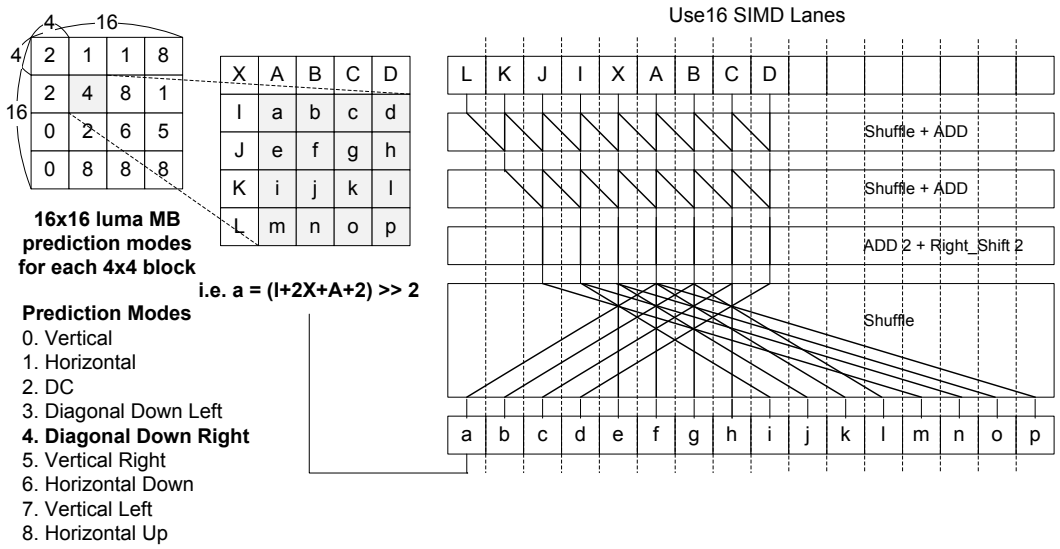


Figure 8.12: Mapping a luma 16x16 MB intra-prediction process on AnySP; Example of the Diagonal Down Right intra prediction for a 4x4 sub block (grey block) is presented with each cycle's operations listed.

the SODA architecture.

LDPC. Among the many decoding algorithms for LDPC, the min-sum algorithm was selected because of its simple operations and low memory requirements. As shown in Figure 8.10, AnySP's LDPC implementation has a speedup of 2.43x compared to SODA. This performance jump is because of the wider SIMD width and the versatile swizzle network which significantly reduced the number of register reads/writes and cyclic shuffle operations when dealing with large block codes. In addition, the support of temporary buffers gives the LDPC min-sum decoding function storage for the minimum and the second minimum values thereby accelerating the corresponding compare/update process [83].

	Components	Units	Area		4G + H.264 Decoder	
			Area mm^2	Area %	Power mW	Power %
PE	SIMD Data Mem (32KB)	4	9.76	38.78%	102.88	7.24%
	SIMD Register File (16x1024bit)	4	3.17	12.59%	299.00	21.05%
	SIMD ALUs, Multipliers, and SSN	4	4.50	17.88%	448.51	31.58%
	SIMD Pipeline+Clock+Routing	4	1.18	4.69%	233.60	16.45%
	SIMD Buffer (128B)	4	0.82	3.25%	84.09	5.92%
	SIMD Adder Tree	4	0.18	<1%	10.43	<1%
	Intra-processor Interconnect	4	0.94	3.73%	93.44	6.58%
	Scalar/AGU Pipeline & Misc.	4	1.22	4.85%	134.32	9.46%
System	ARM (Cortex-M3)	1	0.6	2.38%	2.5	<1%
	Global Scratchpad Memory (128KB)	1	1.8	7.15%	10	<1%
	Inter-processor Bus with DMA	1	1.0	3.97%	1.5	<1%
Total	90nm (1V @300MHz)		**25.17**	100%	**1347.03**	100%
Est.	65nm (0.9V @ 300MHz)		**13.14**		**1091.09**	
	45nm (0.8V @ 300MHz)		**6.86**		**862.09**	

Figure 8.13: PE Area and Power Summary for AnySP running 100Mbps high mobility 4G wireless and H.264 4CIF video at 30fps

8.3.2.2 H.264 Algorithm Analysis

Intra-Prediction. Figure 8.12 shows how the H.264 intra-prediction process is mapped onto AnySP. A 16x16 luma macroblock is broken into 16 4x4 sub-blocks, and each 4x4 sub block has a different intra-prediction mode: Vertical, Horizontal, DC, Diagonal Down Left, Diagonal Down Right, Vertical Right, Horizontal Down, Vertical Left, or Horizontal Up. As can be seen in Figure 8.12, 16 SIMD lanes are used to generate 16 prediction values (a,b, ..., p) with neighboring pixel values (capital letters). Independence between intra-prediction computations for 4x4 sub-blocks allows other SIMD units to execute on different 4x4 sub blocks simultaneously. A factor of 2 increase in the SIMD width almost doubles the processing performance. In addition, fused-operations (shuffle-add and add-shift) reduces unnecessary register read/write accesses, and the SRAM-based swizzle networks supports single cycle

complex swizzle operations, both results in a speedup of 30%.

Deblocking Filter. The H.264 deblocking filter smoothens block edges of decoded macroblocks to reduce blocking distortion. Based on data-dependent filtering conditions, the function of this filter varies dynamically (three-tap, four-tap, or five-tap filter). Figure 8.10 shows AnySP achieves about 2.18x performance increase compared to SODA. The wider SIMD width allows the deblocking filter to take twice as many pixel values to process, which results in 2x speedup over SODA. Like H.264 intra-prediction, fused operations such as shuffle-add, shuffle-shift, and add-shift and the swizzle network helps boost performance up to 27% and 16%, respectively.

Motion Compensation. H.264 adopts tree structured motion compensation (MC). The size of a MC block can be one of 16x16, 16x8, 8x16, 8x8, 4x8, or 4x4, which uses integer-pixel, half-pixel, quarter-pixel, or eighth-pixel resolution. Because sub-sample positions do not exist in the reference frames, the fractional pixel data is created by interpolation. For example, half-pixel values are generated by the 6-tap filtering operation, which fits into an 8-wide SIMD partition; therefore a wider SIMD engine supports processing of multiple filters at the same time. Also, buffer support helps store half-pixel values while sliding a filter window, which saves the number of SIMD-scalar data exchanges. Overall, a 2.1x speedup is achieved by these enhancements.

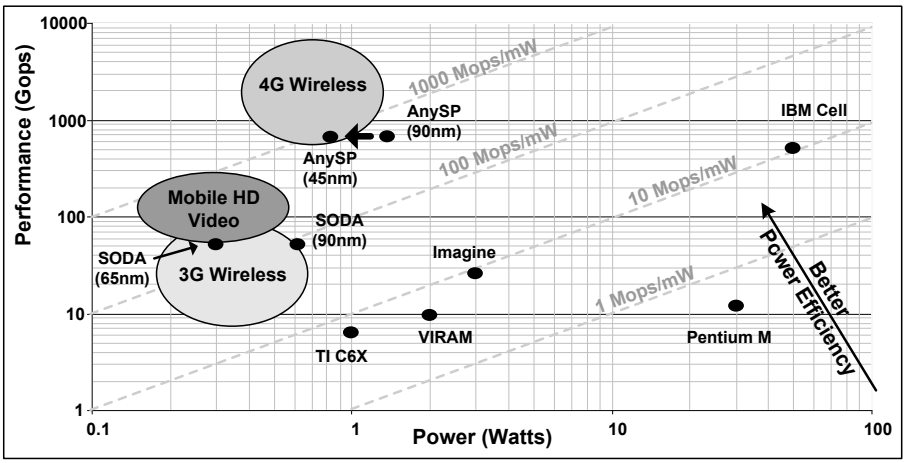


Figure 8.14: Performance verses power requirements for various mobile computing applications.

8.3.3 System-Level Results

Figure 8.13 shows the power and area breakdown of AnySP running both 100 Mbps high mobility 4G wireless and high quality H.264 4CIF video. AnySP was able to meet the throughput requirement of 100 Mbps 4G wireless while consuming 1.3 W at 90 nm. This is just below the 1000 Mops/mW target but close enough to meet that target in 45 nm process technology. We show this in Figure 8.14 where we replot AnySP on the performance verses power chart of Figure 1.1. It can also be seen that high quality H.264 4CIF video at 30 fps can be achieved with 60 mW at 90 nm, meeting the requirements for mobile HD video.

The power breakdown of AnySP shows that the SIMD functional units are the dominant power consumers. The RF and data memory are lower in comparison which shows that our design is efficient because the largest portion of the power is spent doing actual computations. In comparison, SODA's RF and data memory power

were the dominant consumers which suggested that a lot of power was being wasted reading and writing data from the register file and memory. Our proactive approach to reducing register and memory accesses has helped in improving the efficiency of our design.

8.4 Related Work

Many architectures have tried to exploit DLP and TLP in order to increase power efficiency. For instance, the Vector-Thread (VT) architecture [44] can execute in multiple modes: SIMD, MIMD, and hybrids of the two by controlling which processors fetch instructions. AnySP always executes in SIMD mode. When vector lengths are less than the SIMD width, neighboring lanes are combined to execute complex subgraphs or simultaneously operate on multiple vectors using the FFU. Maintaining the single-instruction mode of operation translates into gains in power efficiency compared VT. Although VT is more flexible, the workload characteristics of 4G and H.264 show that this level of flexibility is not required. Judiciously restricting the flexibility helps increase the power efficiency of AnySP.

The use of temporary buffer networks is not a new concept, many former architectures have used it [21][31][67]. The novelty here is the usage model of the temporary buffer and bypass network. Like ELM [21], we use it to store the temporary values that usually do not need to be written back to the register file. The temporary buffer units are connected to the swizzle network in the 8-wide SIMD FFU. This allows the temporary values to be exchanged within the lanes of the SIMD group, which ELM

cannot do. In order to communicate data across multiple ALU units in ELM, the data has to go through an ALU and then enter the distributed switch network. By allowing connection between ALU units, the performance and power is optimized in our SIMD design.

Current solutions that support 3G wireless protocols in SDR solutions can be broken into two categories: SIMD-based architectures and reconfigurable architectures. SIMD-based architectures typically consist of one or few high-performance SIMD DSP processors. The processors are usually connected together through a shared bus, and managed through a general purpose control processor. Some SIMD-based architectures also have a shared global memory connected to the bus. Processors that fall into this category are [54][91][27][42]. Reconfigurable architectures are usually made up of many simpler PEs. Depending on the particular design, these PEs range from fine-grain LUTs to coarser-grain ALU units and even ASICs. The PEs are usually connected together through a reconfigurable fabric. Processors that fall into this category are [59][56][10].

ARM's Ardbeg [91] is an example of a low power SIMD-based architecture that focused solely on 3G wireless communication. On top of the 3G requirements, AnySP was designed to deal with two more challenges: 1) higher bandwidths and more complex algorithms in 4G wireless; and 2) the support for video on the same substrate. The central challenge is simultaneously achieving both high performance and low power for mobile platforms. There is evidence to show that some companies are trying to reduce the number of distinct intellectual properties in their systems, because there

is significant cost and power reduction. They are looking for convergent architectures that can not only execute the communication workload, but also handle some of the other applications like video and graphics. AnySP can be viewed as an example of such a convergent architecture that can support applications with similar characteristics within a limited power and cost budget.

8.5 Summary

Future uses for mobile devices will require more connectivity at higher data rates, support of high quality audio and video, as well as interactive applications. This increase in application diversity can be addressed by combining different processor types each tailored to a specific application. Such a solution is costly in terms of time, silicon area, and power. In this chapter, we presented AnySP, a programmable and flexible SIMD architecture that is also low power. The major contributions of AnySP are the configurable SIMD datapath which supports wide and narrow vector lengths, flexible functional units which can fuse instructions together, temporary buffer and a bypass network which reduces register and memory accesses, SRAM-based swizzle network which reduces the power and latency of swizzle operations, and a multiple output adder tree with speeds up video applications. Results show AnySP's architectural design achieves 2-2.5x speedup over SODA for the set of MSP algorithms while also operating at an energy-delay profile 20-30% more efficient than SODA. AnySP meets the throughput requirement of 100 Mbps 4G wireless while consuming 1.3 W at 90 nm. This puts AnySP under the 1000 Mops/mW target but is close enough that

in 45 nm process technology it will be able reach this milestone. Also, high quality H.264 4CIF video at 30 fps can be achieved with power consumption of 60 mW at 90 nm making AnySP a suitable candidate for mobile HD video processing.

CHAPTER 9

Conclusion

From 4G wireless communications to high definition videos, mobile signal processing has already become an integral part of our everyday lives. Within the past twenty years, engineers have designed increasingly complex DSP systems in order to satisfy our growing appetites for faster and better multimedia content. The up-and-coming multimedia applications pose a new design challenge for computer engineers. They have computation requirements beyond existing desktop computers, while also requiring the power efficiency of hand-held devices.

9.1 Summary

This thesis presents a set of design proposals for realizing an advanced signal processing architecture that targets the next generation mobile computing. In order to design a solution that can meet the lofty requirements, this thesis takes an application-centric approach – evaluate and optimize all aspects of the design based

on the characteristics of the applications. We must first understand the workings of the different applications and their algorithms. With the insights from the applications, a mobile signal processing architecture is designed and optimized for these applications. This thesis makes the following contributions:

1. Algorithmic characterization of two mobile signal processing applications: a fourth generation wireless protocol and high definition mobile video.
2. Architectural insights derived from the characterizations of mobile signal processing applications.
3. An advanced signal processing architecture, *AnySP*, that targets multiple applications domains, specifically 4G wireless communication and high-definition video decoding within the power budget of a mobile device.

9.2 Future Work

The following topics are potential future research directions to extend this work.

High-Level Language for Mobile Signal Processing. Some of the key advantages of Software Defined Radio include flexibility and lower cost. These advantages are based on the assumption that software solutions are easier and more flexible than hardware solution. However, if the users are forced to program advanced mobile signal processors in machine code, then implementing a software solution is not easier nor more flexible than a hardware solution. Therefore, software tool support is a first-order design consideration in providing a viable solution.

Software development for uniprocessor DSPs is hard, and SoC DSP architectures, such as Ardbeg and AnySP processors, make this hard problem even harder. There is a clear need for better language support to help manage the complexity of mapping

DSP systems onto DSP hardware. A language extension designed to address these problems for embedded streaming DSP systems is required. The language should satisfy several design objectives: allowing programmers to express the inherent parallelism within streaming DSP systems, and providing an efficient interface for the compiler to generate code for embedded DSP hardware. It should be designed to support all aspects of embedded DSP computations. This includes dataflow constructs to describe streaming computations, vector arithmetic operations to describe DSP computations, and real-time constructs to describe real-time operations and deadlines.

Scientific Supercomputing with Embedded DSP Processors. Currently, scientific supercomputers are built from general purpose processors. With many high-end scientific applications using similar types of DSP computations as embedded multi-media processing, we can build supercomputers using embedded DSP processors. As previously mentioned, the *AnySP* architecture is able to achieve computation efficiency many orders of magnitude higher than general purpose processors. We can build power efficient supercomputers that consume much less power than the existing ones, which dramatically reduces the cost of maintaining a supercomputing center. With similar power budget as existing supercomputers, we can also build supercomputers that are an order of magnitude faster than the existing ones. There are many potential applications for both low-power supercomputing and high-end supercomputing, ranging from automotive image recognition to gene-folding. However, there are also many potential challenges and interesting questions in designing such embed-

ded DSP-based supercomputers. For example, these system must also provide fast general purpose processors for computations that are ill-suited for DSP processors.

Flexible Computing Fabrics for Multiple Classes of Devices. This thesis has demonstrated the feasibility of supporting multiple mobile applications with the *AnySP* architecture. The goal was to achieve supercomputing like performance at a mobile device power budget. One key idea that *AnySP* proposed was the design of a computing building block which we called the flexible function unit. Each of these units consists of an 8-wide SIMD unit. *AnySP* was composed of 8 of these units together. Though our target was the high-end mobile device market, using this building block we can potentially build smaller or larger *AnySP* processors. Different sizes can help accelerate simpler mobile applications like ZigBee or Bluetooth or even be a replacement for the graphics processor. Companies like Intel and AMD are already going down this path.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] http://public.itrs.net.

[2] http://www.ieee802.org/16/pubs/80216e.html.

[3] Global Cellphone Penetration Reaches 50 Pct. *Reuter*, Nov. 2007.

[4] J. H. Ahn, W. Dally, B. Khailany, U. Kapasi, and A. Das. Evaluating the imagine stream architecture. In *Proc. of the 31st Intl. Symposium on Computer Architecture*, pages 14–24, Jun. 2004.

[5] S. M. Alamouti. A simple transmit diversity technique for wireless communications. *IEEE J. on Select Areas in Communications*, 16(8):1451–1458, 1998.

[6] ARM. Mali gpu opengl es application development guide, 2010. http://infocenter.arm.com/help/topic/com.arm.doc.dui0363-/.

[7] ARM Ltd. *The ARM Architecture Version 6 (ARMv6)*, 2002. White Paper.

[8] ARM Ltd. *ARM Neon Technology*, Sep. 2004.

[9] ARM Ltd. *RealView Compilation Tools Assembler Guide*, Mar. 2007.

[10] R. Baines and D. Pulley. The picoArray and reconfigurable baseband processing for wireless basestations. In *Software Defined Radio*, Feb. 2004.

[11] V. E. Benes. Optimal rearrangeable multistage connecting networks. 43:1641–1656, 1964.

[12] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The M5 Simulator: Modeling Networked Systems. In *IEEE Micro*, volume 26, no. 4, pages 52–60, Jul/Aug 2006.

[13] H.-M. Bluethgen, C. Grassmann, W. Raab, and U. Ramacher. A programmable platform for software-defined radio. pages 15–, Nov. 2003.

[14] S. Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual Design Automation Conference*, pages 746–749, New York, NY, USA, 2007. ACM.

[15] A. Cappelli, A. Lodi, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, and R. Guerrieri. XiSystem: a XiRisc-based SoC with a reconfigurable IO module. In *2005 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 196–593, Feb. 2005.

[16] R. Chau, B. Doyle, M. Doczy, S. Datta, S. Hareland, B. Jin, J. Kavalieros, and M. Metz. Silicon nano-transistors and breaking the 10 nm physical gate length barrier. *Device Research Conference*, pages 23–25, 2003.

[17] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata. Cell broadband engine architecture and its first implementation: a performance view. *IBM J. Res. Dev.*, 51(5):559–572, 2007.

[18] N. Clark, J. Blome, M. Chu, S. Mahlke, S. Biles, and K. Flautner. An architecture framework for transparent instruction set customization in embedded processors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 272–283.

[19] N. Clark et al. OptimoDE: Programmable Accelerator Engines Through Retargetable Customization. In *Proc. Hot Chips 6*, "Aug." 2004.

[20] H. Corporaal and H. J. Mulder. Move: a framework for high-performance processor design. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 692–701, New York, NY, USA, 1991. ACM.

[21] W. Dally et al. Efficient embedded computing. *Computer*, 41(7):27–32, July 2008.

[22] ETSI EN 302 304 V1.1.1. *Digital Video Broadcasting(DVB); Transmission System for Handheld Terminals(DVB-H)*, Nov. 2004.

[23] ETSI TR 101 190 V1.2.1. *Digital Video Broadcasting(DVB); Implementation guidelines for DVB terrestrial services; Transmission aspects*, Apr. 2004.

[24] K. Fan, N. Clark, M. Chu, K. Manjunath, R. Ravindran, M. Smelyanskiy, and S. Mahlke. Systematic register bypass customization for application-specific processors. In *Proceedings. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 2003.*, pages 64 – 74, 2003.

[25] J. Fridman and Z. Greenfield. The TigerSharc DSP architecture. In *IEEE Micro*, pages 66–76, Jan. 2000.

[26] M. Gebhart, B. A. Maher, K. E. Coons, J. Diamond, P. Gratz, M. Marino, N. Ranganathan, B. Robatmili, A. Smith, J. Burrill, S. W. Keckler, D. Burger, and K. S. McKinley. An evaluation of the trips computer system. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 1–12, New York, NY, USA, 2009. ACM.

[27] J. Glossner, E. Hokenek, and M. Moudgill. The sandbridge sandblaster communications processor. In *3rd Workshop on Application Specific Processors*, pages 53–58, Sept. 2004.

[28] N. Goel, A. Kumar, and P. R. Panda. Power reduction in VLIW processor with compiler driven bypass network. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, pages 233–238, 2007.

[29] L. R. Goke and G. J. Lipovski. Banyan networks for partitioning multiprocessor systems. In *ISCA '73: Proceedings of the 1st annual symposium on Computer architecture*, pages 21–28, New York, NY, USA, 1973. ACM.

[30] R. Golshan and B. Haroun. A novel reduced swing CMOS bus interface circuit for high speed low power VLSI systems. volume 4, pages 351–354 vol.4, May-2 Jun 1994.

[31] X. Guan and Y. Fei. Reducing power consumption of embedded processors through register file partitioning and compiler support. In *Proceedings of the 2008 International Conference on Application-Specific Systems, Architectures and Processors*, pages 269–274, Washington, DC, USA, 2008. IEEE Computer Society.

[32] Z. Guo and P. Nilsson. A vlsi architecture of the square root algorithm for v-blast detection. *J. VLSI Signal Process. Syst.*, 44(3):219–230, 2006.

[33] S. Gupta, A. Ansari, S. Feng, and S. Mahlke. Stageweb: Interweaving pipeline stages into a wearout and variation tolerant cmp fabric. In *Proc. of the 2010 International Conference on Dependable Systems and Networks*, June 2010.

[34] W. Haensch, E. J. Nowak, R. H. Dennard, P. M. Solomon, A. Bryant, O. H. Dokumaci, A. Kumar, X. Wang, J. B. Johnson, and M. V. Fischetti. Silicon cmos devices beyond scaling. *IBM J. Res. Dev.*, 50:339–361, July 2006.

[35] P. H. Hofstee. All About the Cell Processor. In *IEEE Symposium on Low-Power and High-Speed Chips(COOL Chips VIII)*, April 2005.

[36] H. Holma and A. Toskala. *WCDMA for UMTS: Radio Access For Third Generation Mobile Communications.* John Wiley and Sons, LTD, New York, New York, 2001.

[37] IEEE Standard 802.11a-1999, Part 11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 5 GHz Band*, 1999.

[38] L. D. Inc. Lf3370 application note, 2001. http://www.logicdevices.com/support/appnotes/lf3370ycb.pdf.

[39] T. Instruments. Render to texture with opengl es, 2010. http://processors.wiki.ti.com/index.php/Render_to_Texture_with_OpenGL_ES.

[40] N. Integration and M. Group. *Predictive Technology Model*. http://www.eas.asu.edu/ ptm/.

[41] T. Karnik, S. Borkar, and V. De. Sub-90nm technologies: challenges and opportunities for cad. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 203–206, New York, NY, USA, 2002. ACM Press.

[42] S. Knowles. *The SoC Future is Soft*. IEE Cambridge Branch Seminar 2005, Dec. 2005. http://www.iee-cambridge.org.uk/arc/seminar05/slides/SimonKnowles.pdf.

[43] C. Kozyrakis and C. Patterson. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks. In *Proc. of the 35th Intl. Symposium on Microarchitecture*, pages 283–293, Nov. 2002.

[44] R. Krashinsky et al. The vector-thread architecture. In *Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004.*, pages 52–63, June 2004.

[45] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Trans. Comput.*, 32(12):1091–1098, 1983.

[46] M. Kudlur and S. Mahlke. Orchestrating the execution of stream programs on multicore platforms. In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, pages 114–124, New York, NY, USA, 2008. ACM.

[47] A. Kumar, P. Kundu, A. P. Singh, L. shiuan Peh, and N. K. Jha. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator. In *in 65nm CMOS, ICCD-2007*, 2007.

[48] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. Comput.*, 24(12):1145–1155, 1975.

[49] H. Lee, Y. Lin, Y. Harel, M. Woh, S. Mahlke, T. Mudge, and K. Flautner. Software defined radio - a high performance embedded challenge. In *HiPEAC. Volume 3793 of Lecture Notes in Computer Science*, pages 6–26. Springer, Nov 2005.

[50] K. Y. Lee. A new benes network control algorithm. *IEEE Trans. Comput.*, 36(6):768–772, 1987.

[51] S.-J. Lee, N. R. Shanbhag, and A. C. Singer. A low-power vlsi architecture for turbo decoding. In *ISLPED '03: Proceedings of the 2003 international symposium on Low power electronics and design*, pages 366–371, New York, NY, USA, 2003. ACM Press.

[52] T. Lestable and E. Zimmermann. Ldpc options for next generation wireless systems. In *Proceedings of the 14th Wireless World Research Forum (WWRF)*, 2005.

[53] T. A. Lin, T. M. Liu, and C. Y. Lee. A low-power H.264/AVC decoder. *International Symposium on VLSI Design, Automation and Test, 2005.*, pages 283–286, April 2005.

[54] Y. Lin et al. SODA: A low-power architecture for software radio. In *Proc. of the 33rd Annual International Symposium on Computer Architecture*, pages 89–101, 2006.

[55] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. Flautner. Design and implementation of Turbo decoders for software defined radio. *IEEE Workshop on Signal Processing Systems Design and Implementation, 2006. SIPS '06.*, pages 22–27, Oct. 2006.

[56] A. Lodi et al. Xisystem: A XiRisc-based SoC with reconfigurable IO module. *IEEE Journal of Solid-State Circuits*, 41(1):85–96, Jan. 2006.

[57] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proc. of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, June 1998.

[58] J. W. McPherson. Reliability challenges for 45nm and beyond. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 176–181, New York, NY, USA, 2006. ACM Press.

[59] B. Mei et al. ADRES: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. *13th International Conference on Field-Programmable Logic and Applications, 2003. FPL 2003*, pages 61–70, Sept. 2003.

[60] F. Naessens, B. Bougard, S. Bressinck, L. Hollevoet, P. Raghavan, L. Van der Perre, and F. Catthoor. A unified instruction set programmable architecture for multi-standard advanced forward error correction. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 31 –36, 2008.

[61] D. Nassimi and S. Sahni. A self-routing benes network and parallel permutation algorithms. *IEEE Trans. Comput.*, 30(5):332–340, 1981.

[62] Y. Neuvo. Cellular phones as embedded systems. pages 32–37 Vol.1, Feb. 2004.

[63] A. Nilsson and D. Liu. Area efficient fully programmable baseband processors. In *International Symposium on Systems, Architectures, Modeling and Simulation (SAMOS)*, pages 333–342, July 2007.

[64] NXP Semiconductors. *Samsung, NXP, and T3G Showcase World's First TD-SCDMA HSDPA/GSM Multi-mode Mobile Phone*, Nov. 2007.

[65] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, 2007.

[66] S. Park, A. Shrivastava, N. Dutt, A. Nicolau, Y. Paek, and E. Earlie. Bypass aware instruction scheduling for register file power reduction. In *LCTES '06: Proceedings of the 2006 ACM SIGPLAN/SIGBED conference on Language, compilers, and tool support for embedded systems*, pages 173–181, New York, NY, USA, 2006.

[67] S. Park, A. Shrivastava, N. Dutt, A. Nicolau, Y. Paek, and E. Earlie. Register file power reduction using bypass sensitive compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1155 –1159, 2008.

[68] A. Peleg, S. Wilkie, and U. Weiser. Intel MMX for multimedia PCs. *Commun. ACM*, 40(1):24–38, 1997.

[69] D. Pham et al. The design and implementation of a first generation CELL processor. In *IEEE Intl. Solid State Circuits Symposium*, Feb. 2005.

[70] B. Plunkett and J. Watson. Adapt2400 ACM Architecture Overview, QuickSilver Technology Inc. San Jose, Jan. 2004.

[71] Qualcomm Inc. *Qualcomm Snapdragon*, 2010. http://www.qualcomm.com/products_services/chipsets/snapdragon.html.

[72] P. Raghavan et al. A customized cross-bar for data-shuffling in domain-specific simd processors. In *ARCS*, volume 4415 of *Lecture Notes in Computer Science*, pages 57–68. Springer, 2007.

[73] U. Ramacher. Software-defined radio prospects for multistandard mobile phones. *Computer*, 40(10):62–69, Oct. 2007.

[74] S. Raman, V. Pentkovski, and J. Keshava. Implementing streaming simd extensions on the pentium III processor. *Micro, IEEE*, 20(4):47–57, Jul/Aug 2000.

[75] G. Rauwerda, P. Heysters, and G. Smit. Towards Software Defined Radios Using Coarse-Grained Reconfigurable Hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):3–13, Jan. 2008.

[76] J. P. Robelly, H. Seidel, K. C. Chen, and G. Fettweis. Energy efficiency vs. programmability trade-off: architectures and design principles. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 587–592, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[77] S. Rodriguez and B. Jacob. Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm). In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 25–30, New York, NY, USA, 2006. ACM Press.

[78] P. Ryan, T. Arivoli, L. De Souza, G. Foyster, R. Keaney, T. McDermott, A. Moini, S. Al-Sarawi, L. Parker, G. Smith, N. Weste, and G. Zyner. A single chip phy cofdm modem for ieee 802.11a with integrated adcs and dacs. *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, pages 338–339, 463, 2001.

[79] S. Satpathy, Z. Foo, B. Giridhar, D. Sylvester, T. Mudge, and D. Blaauw. A 1.07 tbit/s 128x128 swizzle network for simd processors. In *IEEE Symposium on VLSI Circuits*, 2010.

[80] M. Schneider, H. Blume, and T. G. Noll. Power estimation on functional level for programmable processors. volume 2, pages 215–219, 2004.

[81] M. Schulte, J. Glossner, S. Jinturkar, M. Moudgill, S. Mamidi, and S. Vassiliadis. A low-power multithreaded processor for software defined radio. *J. VLSI Signal Process. Syst.*, 43(2-3):143–159, 2006.

[82] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):1–15, 2008.

[83] S. Seo, T. Mudge, Y. Zhu, and C. Chakrabarti. Design and analysis of LDPC decoders for software defined radio. *IEEE Workshop on Signal Processing Systems, 2007*, pages 210–215, Oct. 2007.

[84] H. Taoka, K. Higuchi, and M. Sawahashi. Field experiments on real-time 1-Gbps high-speed packet transmission in MIMO-OFDM broadband packet radio access. *IEEE 63rd Vehicular Technology Conference, 2006. VTC 2006-Spring.*, 4:1812–1816, May 2006.

[85] I. Technologies. Sgx graphics ip core family, 2010. http://www.imgtec.com/powervr/sgx.asp.

[86] B. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: A language for streaming applications. In *In Proceedings of the International Conference on Compiler Construction*, pages 179–196, June 2002.

[87] C. van Berkel et al. Vector Processing as an Enabler for Software-Defined Radio in Handsets From 3G+WLAN Onwards. In *Proc. 2004 Software Defined Radio Technical Conference*, Nov. 2004.

[88] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss. Vector processing as an enabler for software-defined radio in handheld devices. *EURASIP J. Appl. Signal Process.*, 2005(1):2613–2625, 2005.

[89] H.-S. Wang, L.-S. Peh, and S. Malik. A power model for routers: Modeling alpha 21364 and infiniband routers. *IEEE Micro*, 23(1):26–35, 2003.

[90] M. Woh et al. The next generation challenge for software defined radio. In *Proc. 7th Intl. Conference on Systems, Architectures, Modelling, and Simulation*, pages 343–354, Jul. 2007.

[91] M. Woh et al. From SODA to Scotch: The evolution of a wireless baseband processor. *Proceedings. 41th Annual IEEE/ACM International Symposium on Microarchitecture, 2008. MICRO-41.*, pages 152–163, Nov. 2008.

[92] T. T. Ye, G. D. Micheli, and L. Benini. Analysis of power consumption on switch fabrics in network routers. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pages 524–529, New York, NY, USA, 2002. ACM.

[93] H. Zhu, Z. Lei, and F. Chin. An improved square-root algorithm for blast. *IEEE Signal Processing Letters*, 11(9):772 – 775, 2004.

[94] Y. Zhu and C. Chakrabarti. Architecture-aware ldpc code design for software defined radio. *IEEE Workshop on Signal Processing Systems*, 2006.

[95] V. Zyuban and P. Kogge. Split register file architectures for inherently lower power microprocessors. *Proc. Power-Driven Microarchitecture Workshop, in conjunction with ISCA '98*, pages 32–37, 1998.