ENERGY-EFFICIENT COMPUTING FOR MOBILE SIGNAL PROCESSING

by

Sangwon Seo

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, Arizona State University

© Sangwon Seo 2011
All Rights Reserved

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance and support of many people. First and foremost, I would like to thank my advisor, Trevor Mudge. His insight, expertise, enthusiasm, and encouragement played a large part in my success in graduate school. Without his guidance, this dissertation would not exist.

I would also like to thank my thesis committee, Professors Scott Mahlke, David Blaauw, William Martin, and Chaitali Chakrabarti. They donated their time, providing valuable comments and suggestions that helped me refine my thesis.

The research presented in this dissertation is not the work of one person; I was fortunate to have the assistance of a number of other students. In particular, Mark Woh gave me valuable help in virtually every aspect of my graduate school life. Yongjun Park also contributed significantly, helping me debug codes and perform hardware experiments.

As much as those who provided technical expertise, those who offered engaging conversation and moral support were crucial to my graduate school experience, namely: Hyunchul Park, Yuan Lin, Amin Ansari, Hyounkyu Cho, Ganesh Dasika, Shuguang Feng, Shantanu Gupta, Amir Hormati, Po-Chun Hsu. I have shared offices

with them, and my time in Ann Arbor would not have been the same without their friendship.

Finally, I would like to thank my family for their support, encouragement, and advice. My parents and brother provided their unconditional love and support throughout this whole process.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Energy-Efficient Computing for Mobile Signal Processing

by

Sangwon Seo

Chair: Trevor N. Mudge

Mobile devices have rapidly proliferated, and deployment of handheld devices continues to increase at a spectacular rate. As today's devices not only support advanced signal processing of wireless communication data but also provide rich sets of applications, contemporary mobile computing requires both demanding computation and efficiency. Most mobile processors combine general-purpose processors, digital signal processors, and hardwired application-specific integrated circuits to satisfy their high-performance and low-power requirements. However, such a heterogeneous platform is inefficient in area, power and programmability. Improving the efficiency of programmable mobile systems is a critical challenge and an active area of computer systems research.

SIMD (single instruction multiple data) architectures are very effective for data-

level-parallelism intense algorithms in mobile signal processing. However, new characteristics of advanced wireless/multimedia algorithms require architectural re-evaluation to achieve better energy efficiency. Therefore, fourth generation wireless protocol and high definition mobile video algorithms are analyzed to enhance a wide-SIMD architecture. The key enhancements include 1) programmable crossbar to support complex data alignment, 2) SIMD partitioning to support fine-grain SIMD computation, and 3) fused operation to support accelerating frequently used instruction pairs.

Near-threshold computation has been attractive in low-power architecture research because it balances performance and power. To further improve energy efficiency in mobile computing, near-threshold computation is applied to a wide SIMD architecture. This proposed near-threshold wide SIMD architecture—Diet SODA—presents interesting architectural design decisions such as 1) very wide SIMD datapath to compensate for degraded performance induced by near-threshold computation and 2) scatter-gather data prefetcher to exploit large latency gap between memory and the SIMD datapath. Although near-threshold computation provides excellent energy efficiency, it suffers from increased delay variations. A systematic study of delay variations in near-threshold computing is performed and simple techniques—structural duplication and voltage/frequency margining—are explored to tolerate and mitigate the delay variations in near-threshold wide SIMD architectures.

This dissertation analyzes representative wireless/multimedia mobile signal processing algorithms, proposes an energy-efficient programmable platform, and evaluates performance and power. A main theme of this dissertation is that the perfor-

mance and efficiency of programmable embedded systems can be significantly improved with a combination of parallel SIMD and near-threshold computations.

CHAPTER 1

Introduction

1.1 Background

Mobile computing has become ubiquitous. As the proliferation of mobile devices has increased at a spectacular rate, mobile devices have become one of the dominant computing platforms. This trend will continue as mobile devices cover broader application areas such as high-bandwidth internet access, high-quality video, biometric computations (voice and fingerprint recognition), and interactive conferencing. The advanced functionalities for next generation mobile computing require higher data rates, more sophisticated algorithms, and greater computational diversity with stringent power requirements.

The current mobile platforms are designed as heterogeneous system-on-a-chip (SoC) that employs a combination of general-purpose processors (GPPs), digital signal processors (DSPs), application-specific integrated circuits (ASICs), and hardwired accelerators to provide giga-operations-per-second on sub-watt power budget. How-

ever, such heterogeneous organizations are inefficient to build and maintain, and waste silicon area and power. As state-of-the-art applications are adopted, these mobile platforms need to be redesigned with additional ASICs and specialized functional accelerators because current ones are designed only for outdated specifications. Therefore, as more applications and features are introduced to the devices, development and material costs become more expensive.

To solve these problems, programmable mobile platforms that can support multiple standards and applications are being actively investigated. Software Defined Radio (SDR) is one of these mobile platforms that promises to deliver a cost effective and flexible solution by implementing various wireless applications in software. The key advantages of SDR are 1) Multi-mode operation—running multiple protocols, 2) Fast time-to-market—reusing the same hardware for new applications, 3) Easy prototyping and bug fixes—changing software without redesign, and 4) High chip volume.

Many baseband processing architectures for SDR have been proposed in the last few years. They can be broadly categorized into two classes: single instruction multiple data (SIMD)-based and reconfigurable architectures. SIMD-based architectures usually consist of several high performance SIMD processing elements (PEs) that are typically connected together through a shared bus, a shared global memory connected to the bus, and a general purpose control processor that manages these SIMD PEs. Many of SIMD-based architectures support VLIW execution by allowing concurrent memory and SIMD arithmetic operations. SODA [1], Ardbeg [2], EVP [8], Tiger-

SHARC [7], MuSIC-1 [9], Sandblaster [6], and SIMT [5] fall under the SIMD-based architecture category. Reconfigurable architectures, on the other hand, usually consist of many simpler PEs. Depending on the particular design, these PEs range from fine-grain LUTs (lookup tables) to coarse-grain ALUs (arithmetic logic units) or even ASICs. The PEs are usually connected through a reconfigurable fabric. Compared with SIMD-based designs, reconfigurable architectures are more flexible at the cost of higher power. ADRES [10], Montium [11], and XiRisc [13] are categorized in the reconfigurable architecture group.

1.2 Motivation

This dissertation takes a SIMD-based architecture, SODA, to explore the architectural impacts of emerging wireless protocols and advanced signal processing. As wireless signal processing contains vast amounts of vector parallelism, SIMD hardware is recognized as an effective strategy to achieve high efficiency in performance and energy due to its regular structure, ability to scale SIMD lanes, and low control cost. However, the next generation of mobile computing requires higher performance and/or lower power as shown in Figure 1.1.

Figure 1.1 presents the demands of the third generation (3G) and the fourth generation (4G) wireless technology protocols in terms of 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,

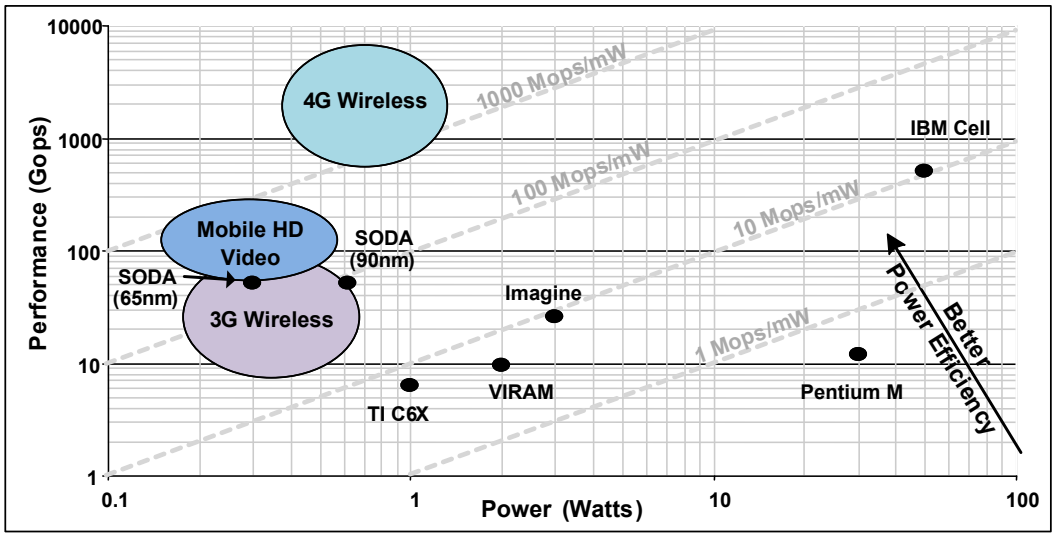

Figure 1.1: Throughput and power requirements for various mobile computing applications [18].

while digital signal processors, such as the TI C6x, operate around 10 Mops/mW. High performance systems such as the IBM Cell can provide excellent throughput, but its power consumption makes it infeasible for mobile devices [14]. Research solutions, such as VIRAM [16] and Imagine [50], 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 [1]. Companies such as Phillips [8], Infineon [9], ARM [2], and Sandbridge [6] have also proposed domain-specific systems that meet the requirements for 3G wireless.

However, 4G increases 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 3G with a power envelope that can only increase by 2-5x [15]. Mobile computing systems are not limited to wire-

less signal processing. High-definition video, audio, 3-D graphics, and other forms of media processing are high value applications for mobile devices. Media applications in mobile devices offer a number of challenges different from those in wireless signal processing. First, the complexity of media processing algorithms is typically higher than that of signal processing algorithms. Computation is no longer dominated by simple vectorizable loops. Instead, significant amounts of control flow exist to handle different operating modes and the inherent complexity of media coding. As a result, SIMD parallelism becomes less efficient in media algorithms. Second, the data access complexity in media processing is higher. Wireless signal processing algorithms typically operate on single dimension vectors, whereas video algorithms operate on two or three dimensional vectors. Thus, media processing push designs to have higher bandwidth and more flexible memory systems. In addition, the power budget is generally more constrained for media processing than for wireless signal processing because of higher usage times. As shown in Figure 1.1, high-definition video is 10-100x more compute intensive than 3G protocols.

Therefore, 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 dissertation focuses mostly on techniques for improving efficiency. The efficient mobile computing in Diet SODA exploits massively parallel systems and near-threshold voltage operations to provide efficiency and programmability as well.

1.3 Contributions

This dissertation presents a set of design proposals for an energy-efficient programmable wireless protocol implementation. In order to satisfy demanding performance and power requirements of next generation mobile computing, this dissertation takes a hardware-software co-design approach that optimizes and evaluates a mobile computing platform based on the characteristics of wireless signal processing algorithms. This dissertation makes the following contributions.

Design and Analysis of advanced signal processing algorithms This dissertation presents algorithmic characterization of two major mobile signal processing algorithms: a representative 4G protocol algorithm (Low Density Parity Check (LDPC)) and high definition mobile video (H.264). Based on insights from their characteristics, a wide-SIMD architecture for SDR, SODA, is revisited and optimized to

meet performance and power requirements. The key enhancements on SODA are 1) use of programmable crossbar to support complex shuffle operations, 2) SIMD partitioning to support fine-grain SIMD computation, 3) Bypass and temporary buffer to support efficient access for short-lived intermediate data, and 4) fused operation to support accelerating frequently used instruction pairs.

Design, implementation, and evaluation of an energy efficient signal processing architecture, Diet SODA This dissertation presents an energy efficient signal processing architecture, Diet SODA. The key design idea is to apply near-threshold operation on a wide-SIMD architecture to achieve both high energy efficiency and high throughput performance in a synergistic manner. A combination of near-threshold circuit techniques and parallel SIMD computations offer several new promising architectural design options: 1) very wide SIMD datapath to compensate for degraded throughput performance induced by near-threshold operations, 2) scatter-gather data prefetcher to exploit the large latency gap between memory operating at full voltage and the SIMD datapath operating at near-threshold voltage, and 3) dual operating mode to support both less stringent realtime-constrained tasks and high-throughput demanding tasks.

In-depth study of variations in near-threshold operations This dissertation presents a systematic study of delay variations induced by near-threshold operations at both circuit- and architecture-levels. The variation-induced timing errors in wide SIMD architectures are shown to be fairly small; therefore three simple techniques—1) structural duplication, 2) voltage margining and 3) frequency margining—are ex-

plored to tolerate and mitigate the timing variability problems. Through a case study based on Diet SODA in 90nm technology node, the variation-induced timing errors in wide SIMD architectures can be handled by the structural duplication scheme by increasing the number of SIMD functional units to replace underperforming ones and exploiting XRAM crossbars to build a new error-free datapath. However, for lower technology nodes, use of only structural duplication is not as efficient; rather a combination of structural duplication and voltage margining leads to a solution with the lowest power overhead.

1.4 Organization

The remainder of this dissertation is organized as follows. Chapter 2 introduces two types of contemporary baseband processing architectures—SIMD-based and Reconfigurable for SDR and presents a SIMD-based architecture, SODA, in detail. Chapter 3 and Chapter 4 present case studies of the implementation of LDPC decoders and H.264 codecs on wide-SIMD architectures. Chapter 5 proposes an energy-efficient signal processing architecture, Diet SODA, and presents the central themes and ideas of the dissertation. Chapter 6 addresses increased process variation issues in Diet SODA. Finally Chapter 7 concludes the dissertation, summarizing contributions and suggesting future research directions.

CHAPTER 2

Background

2.1 SDR solutions

There has been tremendous industrial interest in SDR from universities and many leading semiconductor companies. The proposed SDR solutions can be categorized into two types: SIMD-based architectures and reconfigurable architectures.

SIMD-based architectures usually consist of several high performance SIMD PEs and a shared global memory; these components are connected through a shared bus and managed by a general purpose control processor. Many SIMD-based architectures support VLIW execution by allowing concurrent memory and SIMD arithmetic operations. Reconfigurable architectures, on the other hand, usually consist of many simple PEs that are connected through a interconnection fabric. These PEs range from fine-grain LUTs to coarse-grain ALUs or even ASICs. Appendix A and B introduces existing SIMD-based and reconfigurable architectures for SDR baseband processing. In this chapter, we present a SIMD-based architecture, SODA, which

serves as a baseline architecture throughout this dissertation.

2.1.1 SIMD-based Architectures

SIMD-based architectures usually consist of one or more high-performance SIMD DSP processors that are connected through a shared bus and managed by a general purpose control processor. These types of architectures usually use software-managed scratchpad data memories to meet real-time constraints. Most SIMD-based SDR processors support VLIW execution by allowing concurrent memory and SIMD arithmetic operations. Some commercial solutions choose to incorporate accelerators for error correction algorithms, including Viterbi and Turbo decoders.

2.1.1.1 University of Michigan, SODA

SODA (Signal Processing On Demand) is an academic research prototype for mobile SDR [1]. It is a SIMD-based DSP architecture designed to meet both performance and power requirements for two representative protocols, WCDMA and IEEE 802.11a.

The SODA multiprocessor architecture is shown in Fig. 2.1. It consists of multiple data processing elements (PEs), one control processor and a global scratchpad memory, all of which are connected through a shared bus. Each SODA PE consists of five major components: 1) a 32-way, 16-bit datapath SIMD pipeline for supporting vector operations. Each datapath includes one 16-bit ALU with multiplier and a 2 read-port, 1 write-port 16 entry register file. Intra-processor data movements

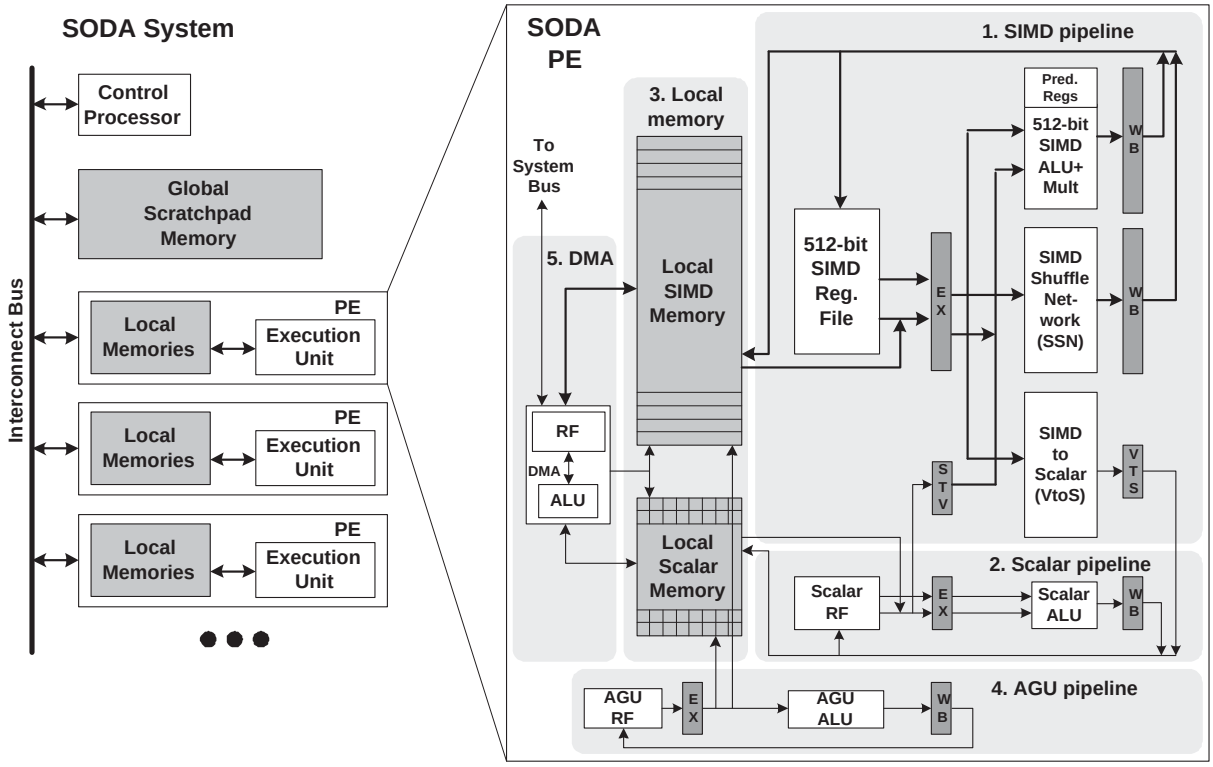

Figure 2.1: A SDR architecture: SODA [1]

are supported through a SIMD Shuffle Network (SSN); 2) a 16-bit datapath scalar pipeline for sequential operations. The scalar pipeline executes in lock-step with SIMD pipeline; SIMD-to-scalar and scalar-to-SIMD operations exchange data between two pipelines through the STV (Scalar-To-Vector) and VTS (Vector-To-Scalar) registers; 3) two local scratchpad memories for the SIMD pipeline and the scalar pipeline; 4) an AGU (Address-Generation-Unit) pipeline for providing the addresses for local memory accesses; and 5) a programmable DMA (Direct-Memory-Access) unit for transferring data between scratchpad memories and interface with the outside system (inter-processor data transfer). The SIMD pipeline, the scalar pipeline

and the AGU pipeline execute in VLIW-styled lock-step manner, controlled with one program counter (PC) [1].

Arithmetic Data Precision. SODA PE only provides support for 8- and 16-bit fixed-point operations because many DSP algorithms in both WCDMA and 802.11a wireless protocols operate on 8- or 16-bit fixed point data. Each lane in the SIMD pipeline and the scalar pipeline support 16-bit fixed-point arithmetic operations. The AGU pipeline supports 8-bit addition and subtraction because 8 bits are sufficient for software-managed data buffers.

Vector Permutation Operations. SODA's SSN consists of a shuffle exchange (SE) and an inverse shuffle exchange (ISE) networks to support any 32-wide vector permutation. By including both the SE and ISE networks, the number of iterations can be reduced to a maximum of 9 clock cycles and a straight-through connection is also provided. Combining with predicated move operations, the SSN can support any vector length permutation.

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 400MHz, SODA consumes 2.95W for W-CDMA 2Mbps system and 3.2W for 802.11a 24Mbps system. However, with technol-

ogy scaling, the power numbers are expected to reduce to acceptable levels such as 450mW for 90nm technology and 250mW for 65nm technology.

2.1.1.2 ARM, Ardbeg

The Ardbeg system architecture consists of two PEs, an ARM general purpose controller, and a turbo coprocessor, all of which are connected through a 64-bit AMBA 3 AXI interconnect bus. The overall architecture of the Ardbeg PE is very similar to the SODA PE, with a 512-bit SIMD pipeline, a scalar pipeline, an AGU pipeline, and local memory. In addition to SODA's 16-fixed point operations, Ardbeg also supports 8-bit and 32-bit fixed operation as well as 16-bit block floating point operations. To implement SIMD shuffle network, Ardbeg adopts a 7-stage single-cycle Banyon network, which allows faster data alignment operations that are more important tasks in upcoming wireless algorithms. Ardbeg PE uses one unified scratchpad memory because DLP-dominant DSP algorithms make it more efficient to share the memory space between the SIMD datapath and the scalar datapath. Turbo decoding, one of the widely used error correction algorithms in wireless communications, is very computationally intensive and hard to vectorize. Therefore, Ardbeg offloads the task to a turbo coprocessor to increase performance and power-efficiency. Ardbeg is designed using the OptimoDE framework [3], which allows the generation of custom VLIW-style architecture and faster evaluation of the architecture. The instruction set for Ardbeg is derived from the ARM NEON extensions [17].

2.1.1.3 Icera, DXP

The Icera's Deep eXecution Processor (DXP) [4] is a 2-LIW 4-way 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 less flexible SIMD datapath. Icera's processors do not use any hardware accelerators.

2.1.1.4 Linkoping University, SIMT

SIMT [5] architecture consists of Complex MAC (CMAC) SIMD units, Complex ALU (CALU) SIMD units, memory banks, on-chip network, accelerators, and a controller unit. The controller core efficiently manages the two SIMD units and the matching memory bank system so that several threads can be run simultaneously. 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. To provide required memory bandwidth to the SIMD units and accelerators, SIMT architecture uses a number of memory banks. Each memory bank contains its own address generation unit to minimize memory access conflicts. The programmable reconfigurable crossbar switch is used as the on-chip network.

2.1.1.5 Sandbridge Technology, Sandblaster

Sandblaster [6] 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 (T3) which consumes much less power than simultaneous multithreading (SMT), because T3 issues instructions in round-robin fashion. The Sandblaster architecture supports up to eight threads.

2.1.1.6 Analog Devices, TigerSHARC

The TigerSHARC [7] implementation, ADSP-TS001, adopts several mechanisms that are found in general-purpose processors such as 1) a register-based load-store architecture with a static super-scalar dispatch mechanism, 2) highly parallel short-vector-oriented memory architecture, 3) support for multiple data types including 32-bit single-precision floating point and 8-bit/16-bit fixed point, 4) parallel arithmetic instructions for two floating-point multiply-accumulate (MAC) operation or eight 16-bit MACs, 5) 128-entry four-way set associative branch target buffer (BTB), and 6) 128 architecturally visible, fully interlocked registers in four orthogonal register files. 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.

2.1.1.7 NXP, EVP

The Embedded Vector Processor, EVP [8], consists of 16-wide 16-bit SIMD datapath, one scalar datapath, programmable memory, and VLIW controller. The SIMD datapath comprises 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. The EVP also supports multiple data types such as complex numbers. This architecture allows maximum parallelism using VLIW execution: five vector operations, four scalar operations, three address operations and one loop control can be executed at once. In the SIMD datapath, the shuffle unit rearrange the elements of a single vector according to any pattern; intravector unit supports summation, maximum, and split operations, and the code generation unit supports various CDMA-code generations for different systems and Cyclic Redundancy Checks (CRC) as well.

2.1.1.8 Infineon Technologies, MuSIC-1

Infineon baseband processor, MuSIC-1[9], 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 in parallel. 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; therefore, all of the baseband processor's components are shared through on-chip memory, which consists of multiple banks to support simultaneous accesses.

2.1.2 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.

2.1.2.1 IMEC, ADRES

ADRES [10], Architecture for Dynamically Reconfigurable Embedded System, is an example of a coarse-grain reconfigurable tile architecture that tightly couples a VLIW processor and a 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; this approach is more compiler-friendly than the message-passing method. In addition, ADRES relies on modulo scheduling and traditional VLIW compiler support to exploit both DLP and ILP to maximize PE utilization.

2.1.2.2 Delft, Montium

Montium [11] 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 based on a particular algorithm. The TP consists of five Montium ALUs and ten local memories that are vertically segmented into a 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. In the Montium implementation, 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.

2.1.2.3 QuickSilver Technology, Adapt2400 ACM

Adapt2400 ACM (Adaptive Computing Machine) [12] 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 consists of its own controller, memory, and computational resources so that it independently executes algorithms that are downloaded in the form of SilverWare binary files. A node is capable of implementing a first come, first served, non preemptive multitasking system with the support of hardware task manager. The MIN ties the heterogeneous nodes together carrying data, SilverWare, and control information between nodes as well as outside the system. This 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 an integrated scalable hardware/software platform.

2.1.2.4 XiSystem, XiRisc

XiRisc [13] 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 of rows that 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 exploits the synergy of the different execution units, ranging from a 32-bit dedicated MAC unit to bit-level processing blocks on PiGoGA., which increases execution efficiency and saves energy.

2.2 Near-threshold computing

Near-threshold computation has been attractive in low-power architecture research due to its characteristics of balancing energy savings and performance loss. Near-threshold operation, as described by Zhai et al. [54], defines three regions of operation, pictured in Figure 2.2. In the superthreshold regime ($V_{dd} > V_{th}$), energy is highly sensitive to V_{dd} due to the quadratic scaling of switching energy with V_{dd}. Hence, voltage scaling down to the near-threshold regime ($V_{dd} \sim V_{th}$) yields an 10x energy reduction at the expense of approximately 10x performance degradation. However, the dependence of energy on V_{dd} becomes more complex as voltage is scaled below V_{th}. In subthreshold regime ($V_{dd} < V_{th}$), circuit delay increases exponentially with V_{dd}, causing leakage energy (the product of leakage current, V_{dd}, and delay) to increase in a near-exponential fashion. This rise in leakage energy eventually dominates any reduction in switching energy, creating an energy minimum.

The identification of an energy minimum led to interest in processors that operate at this energy optimal supply voltage [58]. However, the energy minimum is relatively

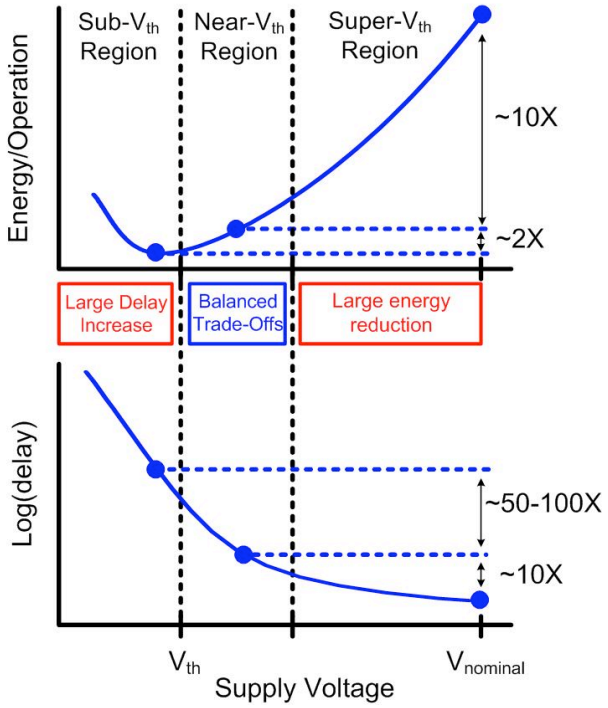

Figure 2.2: Supply voltage operating regions and the energy and delay associated at each point. The near-threshold region provides considerable energy savings for non-timing critical low power applications such as DSCs.

shallow. Energy typically reduces by only ∼2x when V_{dd} is scaled from the near-threshold regime to the subthreshold regime, though delay rises by 50-100x over the same region. While acceptable in ultra-low energy sensor-based systems, this delay penalty is not tolerable for a broader set of applications.

The identification of an energy minimum led to interest in processors that operate at this energy optimal supply voltage [58]. However, the energy minimum is relatively shallow. Energy typically reduces by only ∼2x when V_{dd} is scaled from the near-threshold regime to the subthreshold regime, though delay rises by 50-100x over the same region. While acceptable in ultra-low energy sensor-based systems, this delay penalty is not tolerable for a broader set of applications.

The near-threshold region offers an opportunity for many applications to reduce energy further. In order to do so, the design must overcome one hurdle, the 10x increase in delay. This delay impacts the ability of designs to meet more stringent real time constraints without scaling the voltage higher and losing energy efficiency. However, in cases where the application can be parallelized, simply using more near-threshold processing elements can meet the timing constraint with greater efficiency. Near-threshold operation, therefore, has a natural synergy with data parallel environments like SIMD. In a SIMD architecture, the number of functional units can be increased to help meet a timing critical code, provided the application has sufficient DLP.

However, Near-threshold designs are impacted greater by process variations than traditional designs, because the on-current (I_{on}) in the near-threshold voltage region is highly sensitive to variations in V_{th}. Increased process variations in advanced technology nodes further exacerbates the problem, providing many challenges for process engineers and circuit designers [62]. These variation-induced timing errors are much more critical in wide SIMD architectures for two reasons. First, the probability that all SIMD datapaths are error-free decreases when variations are severe, because the number of critical paths are multiplied by the SIMD width. Recent work also shows that there is a significant performance drop in SIMD architectures as single-stage-error probabilities increase [63]. Second, commonly used error-tolerating methods such as pipeline stalling or re-execution result in greater performance and power penalties due to problems in one lane impacting all other lanes. To tolerate variation-induced timing

errors in near-threshold operations, complex architectural enhancements have been considered. For example, Synctium [63] proposed decoupled parallel SIMD pipelines and pipeline weaving using decoupling queues and micro-barriers.

The details about how near-threshold computing affects wide SIMD architectures and how the increased delay variation affects the architectures will be discussed in Chapter 5 and Chapter 6 respectively.

CHAPTER 3

Design and Analysis of LDPC Decoders for Software Defined Radio

Wireless communication has grown at a spectacular rate. As the number of users and the demand for high quality contents increase, the current bandwidth that 3G wireless technology provides becomes insufficient. To address these issues, International Telecommunications Union (ITU) proposes 4G wireless technology that increases the bandwidth to maximum data rates of 100Mbps for high mobility and 1Gbps for stationary/low mobility. This increase in bandwidth requires significant computations to process 4G wireless standard.

Low Density Parity Check (LDPC) codes are one of the most promising error correction codes that are being adopted by many 4G wireless standards. This chapter presents a case study for a scalable LDPC decoder supporting multiple code rates and multiple block sizes on a software defined radio (SDR) platform. Since technology scaling alone is not sufficient for current SDR architectures to meet the requirements

of the next generation wireless standards, this chapter presents three techniques to improve the throughput performance. The techniques are use of data path accelerators, addition of a few assembly instructions and addition of a memory interface. The proposed LDPC decoder implementation on an SDR platform achieves 30.01 Mbps decoding throughput for n=2304 and R=5/6 LDPC code outlined in the IEEE 802.16e standard.

3.1 Introduction

Low density parity check (LPDC) codes have excellent error correction performance that approaches the Shannon capacity limit [20], [21]. As a result, they have been adopted in many current and next generation wireless protocols such as DVB-S2 and the IEEE 802.16e standard (WiMAX). Decoders used for LDPC codes have high throughput requirements and have been successfully implemented using ASICs and FPGAs [22]. However, the emergence of a wide variety of wireless protocols that are rapidly changing makes custom hardware for these decoders relatively time consuming and expensive to develop.

This chapter presents a case study for a LDPC decoder implementation that supports multiple code rates and multiple block sizes on a SDR platform, SODA. When the LDPC matrix is represented by structured submatrices, the data-level parallelism can be efficiently handled by the SIMD pipeline. However the current SODA architecture is unable to meet the high decoding throughput and the scalability requirements (multiple block sizes and multiple code rates) of the IEEE 802.16e standard.

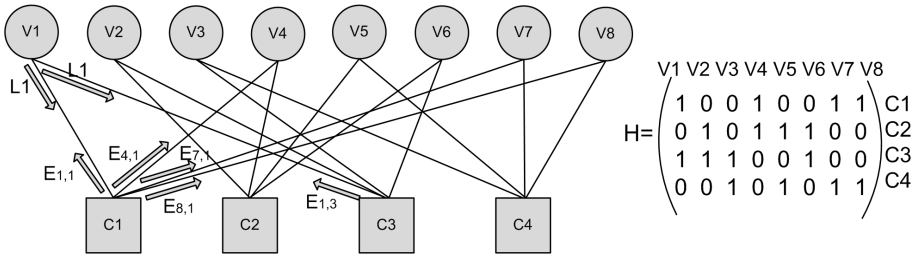

Figure 3.1: LDPC matrix H and the corresponding bipartite graph

In this chapter we present use of data path accelerators, addition of memory units and addition of a few assembly instructions to address the throughput and scalability requirements. The proposed LDPC decoder implementation achieves 30.4 Mbps decoding throughput for the n=2304 and R=5/6 LDPC code outlined in the IEEE 802.16e standard.

The rest of the chapter is organized as follows. Section 3.2 gives a brief overview of LDPC codes. Section 3.3 introduces the mapping of the LDPC decoder onto SODA. Section 3.4 describes LDPC accelerators, memory controller/buffer organization and assembly support required for the high throughput scalable LDPC decoder implementation. Section 3.5 presents memory and throughput analysis of the augmented architecture. Section 3.6 concludes the chapter.

3.2 LDPC Basics

3.2.1 Introduction

A LDPC code is a class of linear block codes whose codewords satisfy a set of linear parity-check constraints [20]. These constraints are typically defined by an m-by-n parity-check matrix H, whose m rows specify each of the m constraints (the number of parity checks), and n represents the length of a codeword. H is also characterized by W_r and W_c, which represent the number of 1's in the rows and columns, respectively. A LDPC code can be represented by a bipartite graph, which consists of two types of nodes, Variable Nodes (VN) and Check Nodes (CN). Check node i is connected to variable node j whenever h_{ij} of H is non-zero. Fig. 3.1 describes the matrix H and the corresponding bipartite graph of a simple LDPC code.

3.2.2 LDPC Decoding Process

LDPC codes are decoded iteratively using a message passing algorithm [20]. This algorithm involves exchanging the belief information among the variable nodes and check nodes that are connected by edges in the bipartite graph. Let I_n be the intrinsic information from the received signal, L_n be the reliable information for variable node n, $L_{n,m}$ be the information conveyed from variable node n to check node m, and $E_{n,m}$ be the extrinsic information generated in check node m that is passed to variable node n. The belief information is updated in an iterative manner and implemented in two phases. In the first phase, the variable nodes send their belief information,

$L_{n,m}$, to check nodes connected to them; in the second phase, the check nodes send the updated belief information (new $E_{n,m}$) to the variable nodes connected to them for updating L_n (See Fig. 3.1). The iteration steps are summarized in Algorithm 1.

Algorithm 1: Min-sum LDPC Decoding Algorithm

---

1. Initialization: $E_{n,m} = 0$, $L_n = I_n$
2. VN to CN: $L_{n,m} = L_n$ - $E_{n,m}$
3. Update $E_{n,m}$: $E_{n,m}^{new} = f(L_{n',m} | n' \in S \subset N(m))$
4. Update L_n: $L_n^{new} = L_{n,m} + E_{n,m}^{new}$
5. Repeat the steps 2,3,4 for NUM iteration times
6. Make a decision of bit n based on the corresponding L_n value

---

Here, $N(m)$ is the set of variable nodes which are connected with check node m in the bipartite graph. Similarly, $M(n)$ is the set of check nodes which are connected with variable node n. The decoding algorithms differ in how the function f in Step 3 of Algorithm 1 is evaluated.

There are three options for the LDPC iterative decoding algorithm: Belief Propagation (BP), λ-min and min-sum algorithms [23]. Although BP and λ-min algorithms show better error correction performance compared to min-sum algorithm, these algorithms require a look-up table for hyperbolic function values, which requires additional memory space. The min-sum algorithm is selected here because of the limited

memory size and easy computation patterns. The min-sum algorithm f is shown as follows. Here, $n' \in N(m)$, $n' \neq n$.

$$E_{n,m}^{new} = - \left(\prod_{n'} sign(L_{n,m})\right) \times \min_{n'} |L_{n,m}|$$

As can be seen, the operations in the min-sum LDPC decoding algorithm are limited to addition, subtraction and finding a minimum value, all of which can be supported by our SDR architecture described in Section 3.3.

Theoretically, the LDPC decoding process finishes when all parity-check equations are satisfied. In reality, a predefined number of iterations (NUM) based on SNR is generally used.

3.2.3 LDPC Matrix Partition

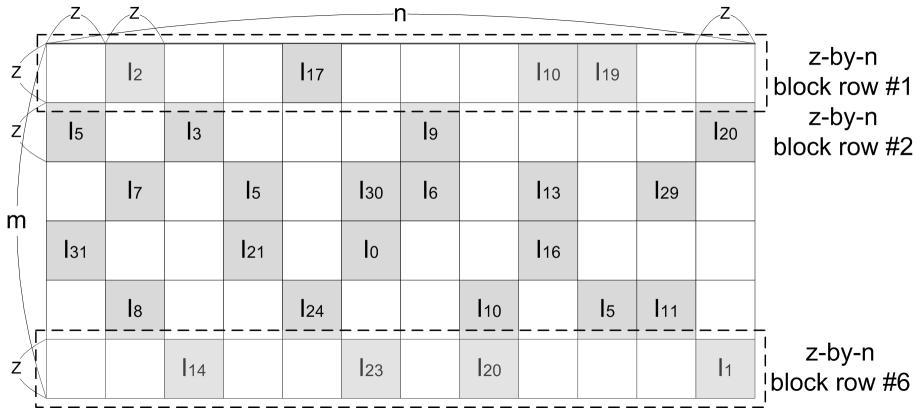

Figure 3.2: Partitioning of H into z-by-z cyclic identity matrices

A LDPC matrix H has randomly distributed 1's which results in complex data routing and is a major challenge for building a high-performance and low-power LDPC decoder. [22] and [24] show that introduction of some structural regularity in the

matrix does not degrade its error correction performance. Moreover the regularity enables partially parallel implementation of LDPC decoders and has been utilized in the IEEE 802.16e standard. Fig. 3.2 shows the partitioning of H into z-by-z cyclic identity submatrices. Here, I_x represents a cyclic identity matrix with rows shifted cyclically to the right by x positions. This characteristic reduces the routing overhead and has been exploited efficiently in our architecture. Fig. 3.2 also shows how the $\frac{n}{z}$ of the identity matrices along a row can be grouped to form a block row. So, in essence, the H matrix can also be partitioned into $\frac{m}{z}$ block rows each of size z-by-n.

3.3 LDPC on SODA

The min-sum LDPC decoding algorithm (Algorithm 1) is mapped onto SODA (See Figure 2.1) in the following way. Step 2 of Algorithm 1 is applied to non-zero z-by-z submatrices. However, because Step 3 uses the $L_{n,m}$ values related with check node m, the SIMD pipeline loads z values of type L_n and aligns the data in check node order by using SSN before executing Step 2. The shuffled $L_{n,m}$ values for all non-zero z-by-z submatrices in one z-by-n block row are calculated in the SIMD datapath. After that, the SIMD-to-Scalar unit is used for finding the minimum $E^{new}_{n,m}$ among W_r of $L_{n,m}$ values for the same check node m. Next, $E^{new}_{n,m}$ and the corresponding sign indicator are used to update a L_n value (Step 4). This procedure implies that some SIMD slices execute additions and others execute subtractions based on sign values – a feature that is supported by predicated instructions in SODA. After updating the L_n values, the data is inversely shuffled and stored in variable node order. This

process is repeated for every z-by-n block row in every iteration.

3.4 Scalable LDPC Implementation

In this section, we study a scalable LDPC decoder implementation for block size n, code rate R=k/n, and (W_c, W_r)-LDPC code as specified by the IEEE 802.16e standard on a SODA PE. We describe the enhancements that had to be made in terms of accelerators, memory units, and new assembly instructions to support multiple code rates and multiple block sizes. Fig. 3.3 shows the modified SIMD pipeline – the additional units have been shown using shaded blocks.

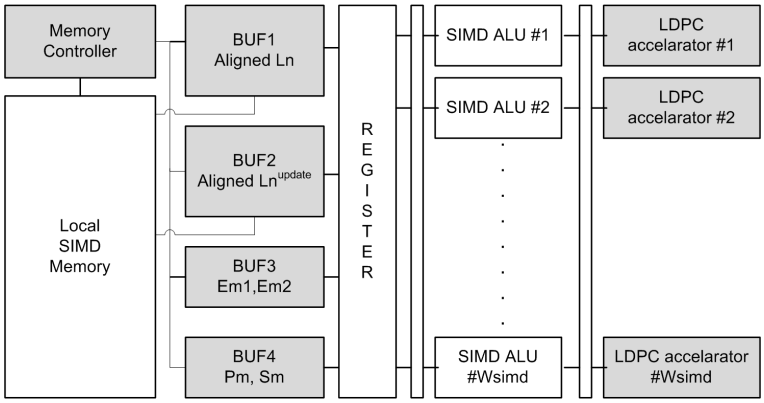

Figure 3.3: Modified SIMD pipeline in a SODA PE

3.4.1 LDPC Accelerator

In order to meet the high decoding throughput requirements, we introduce a LDPC accelerator in every SIMD slice as shown in the Fig. 3.3. There are only two possible $E_{n,m}^{new}$ values for check node m in Step 3 of Algorithm 1 (which are selected from W_r

values of type $L_{n,m}$): the minimum E_{m1} and the second minimum E_{m2}. Each LDPC accelerator expedites finding the minimum values using two compare/store units with two W_r-bit special registers, a selection register P_m and a sign register S_m, as can be seen in Fig. 3.4. The operation of the LDPC accelerator is summarized below.

The Algorithm of LDPC Accelerator

```
if (Ln,m <= Em1) \\ operations in Cmp&Store 1
{
  Em1 <= Ln,m; Em2 <= Em1;
  if (Ln,m < Em1) Pm = 1 << i; else Pm = 0;
}
else if (Ln,m < Em2) \\ operations in Cmp&Store 2
{
  Em2 <= Ln,m;
}
Sm = (Sm | sign(Ln,m)) << 1;
```

E_{m1}, E_{m2}, P_m and S_m are extracted using a flush signal and these values are used to compute $E_{m,n}$ using the following operation (Step 7 and 14 of Algorithm 2).

if $(P_m[i] == 1)$ $E_{m,n}[i] = (S_m[i])$ E_{m1}, else $E_{m,n}[i] = (S_m[i])$ E_{m2}

3.4.2 Memory Units

A major challenge in decoding LDPC codes is the large number of data alignment operations required for every z-by-z permutation matrix. z values of type L_n need to

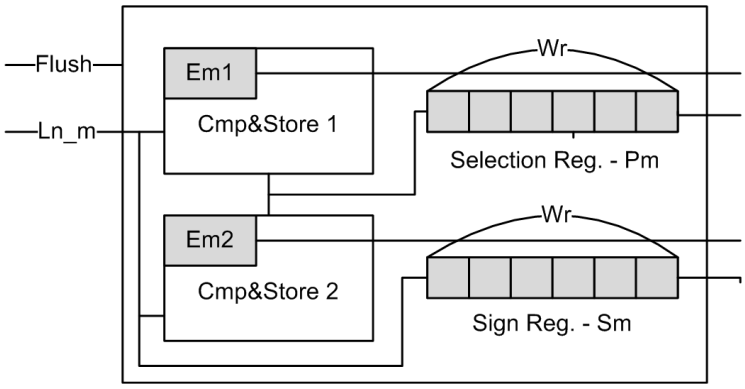

Figure 3.4: LDPC accelerator

be shuffled so that they can be correctly aligned for check node processing. If z is less than the SIMD width (W_{simd}), the data alignment can be executed in one clock cycle using SSN. However, the IEEE 802.16e standard uses different z values (24, 28, 32, ..., 96) for different block sizes [25]. If z is larger than W_{simd}, many clock cycles are required for data alignment operation when SSN is used. This causes a degradation in the LDPC decoding throughput performance.

To solve the alignment issue, we propose a memory controller and buffer organization (instead of using the shuffle network) as shown in Fig. 3.3. BUF1 and BUF2 contain aligned L_n and L_n^{update} (to be described in Section. 3.4.3) respectively; BUF3 contains E_{m1} and E_{m2}; and BUF4 contains P_m and S_m.

The memory controller handles movement of L_n data between the SIMD memory and BUF1. Since the z-by-z permutation matrices in the LDPC codes used in the IEEE 802.16e standard are circular right-shifted identity matrices, each permutation matrix can be defined by a single right-shifted amount s. The alignment operation can now be achieved by two memory copy operations described below. If the shifted

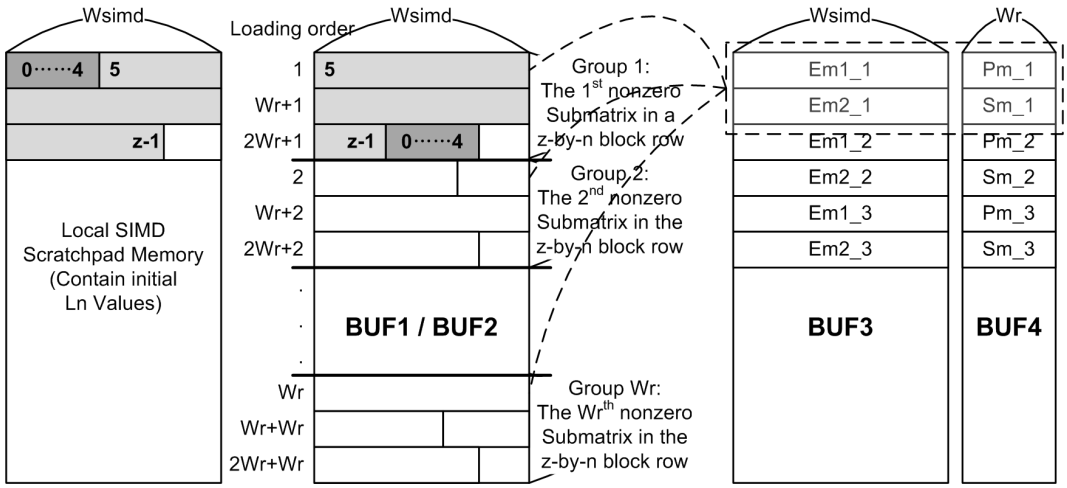

Figure 3.5: Data alignment in buffers

amount is s and the start memory address is m_{start}, the memory controller first copies MEM[$m_{start} + s$ ... $m_{start} + z - 1$] to BUF1, and then copies MEM[m_{start} ... $m_{start} + s - 1$] to BUF1. This is shown in Fig. 3.5 for an example where s=5, m_{start}=0. This is done for all non-zero W_r submatrices in a z-by-n block row. At the end of this process, BUF1 contains W_r groups of aligned L_n data (see Fig. 3.5). In a similar way, the memory controller fills BUF2 for L_n^{update} data with another shift amount $((s - s^{update})$ mod $W_{simd})$ (to be described in Section. 3.4.3). Note that the width of BUF1 and BUF2 is W_{simd}.

3.4.3 Modified Decoding Algorithm

Algorithm 2 shows the LDPC decoding algorithm on the modified SODA architecture. The L_n and L_n^{update} values are aligned and stored in BUF1 and BUF2 (Steps 1 and 2 of Algorithm 2). The aligned values of L_n and L_n^{update} (Step 5) along with

E_{m1}, E_{m2} (Step 4), P_m and S_m (Step 6) of the first row of the first group (see for example Group 1 in Fig. 3.5) are fed to the ALU unit and LDPC accelerator in each SIMD slice. These values are updated in Steps 7, 8, 9 of Algorithm 2. The process is repeated for the first row of the next group (see for example Group 2 in Fig.6), and so on. After completing processing of all the first rows of all the W_r groups (Step 10), the updated values of E_{m1}, E_{m2}, P_m and S_m are stored in their respective buffers (Steps 11, 12). The updated values are used to compute $E_{m,n}^{new}$ and L_n^{update} (Step 15, 16) of the first row of each W_r group (Step 17). The process is repeated for the second row of each W_r group, and so on (Step 18). The above schedule results in high decoding throughput performance; it reduces the number of data switches and also speeds up the operation of finding the minimum values in the min-sum decoding algorithm. After processing all the data for one z-by-n block row, the data for the next z-by-n block row is loaded into BUF1 and BUF2, and the process repeats the number of z-by-n block rows($=\frac{n(1-R)}{z}$) times.

Algorithm 2: LDPC decoding algorithm in the modified SODA PE

---

1. load aligned L_n to BUF1
2. load aligned L_n^{update} to BUF2
3. load W_r for the current z-by-n block row
4. load E_{m1}, E_{m2} from BUF3
5. load L_n, L_n^{update} from BUF1, BUF2
6. load P_m, S_m from BUF4

7. compute $E_{m,n}^{curr}$ using E_{m1}, E_{m2}, P_m, and S_m.

8. update $L_{n,m} = L_n + L_n^{update}$ - $E_{m,n}^{curr}$

9. update E_{m1}, E_{m2}, P_m, and S_m using $L_{n,m}$

10. repeat step 5 to step 9 W_r times

11. store updated E_{m1}, E_{m2} (E_{m1}^{new},E_{m2}^{new}) in BUF3

12. store updated P_m, S_m (P_m^{new}, S_m^{new}) in BUF4

13. load L_n^{update} from BUF2 again

14. compute $E_{m,n}^{new}$ using E_{m1}^{new}, E_{m2}^{new}, P_m^{new}, and S_m^{new}

15. update L_n^{update} += $E_{m,n}^{new}$

16. store updated L_n^{update} in MEM

17. repeat step 12 to step 16 W_r times

18. repeat step 4 to step 17 $\lceil \frac{z}{W_{simd}} \rceil$ times

19. repeat step 1 to step 18 $\frac{n(1-R)}{z}$ times.

20. repeat step 1 to step 19 NUM times.

In order to reduce the memory for storing $L_{n,m}$, we introduce the parameter L_n^{update}, which is (-$E_{n,m}$ + $E_{n,m}^{new}$). In fact, the memory space is reduced by a factor of m by keeping one L_n^{update} value for each check node n instead of storing all $L_{n,m}$ values for every n and m combination.

Since updated L_n^{update} values are processed in check node order, inverse alignment operation is required to store the data in variable node order in memory. After L_n^{update} is stored back in memory, for the next z-by-n block row computation, the data is realigned with a different shift amount. However, these two alignment operations can be reduced to one alignment operation using another shift amount s^{update}; instead of inverse alignment operation, L_n^{update} is stored with the current shifted amount s^{update} and then, in the next iteration, the memory controller use $((s - s^{update}) \bmod W_{simd})$ as a shift amount to align L_n^{update}.

3.4.4 Assembly Support

New assembly instructions are required for the proposed architecture to improve the decoding throughput performance. Steps 1 and 2 of Algorithm 2 are independent and can be executed in parallel. These are combined to form instruction $ldpc\_mem2buf$. Similarly steps 5 and 6 of Algorithm 2 can be executed in parallel and combined to form instruction $ldbufs$. Steps 8 and 9 of Algorithm 2 can be executed in a pipelined manner through the ALU unit and the LDPC accelerator unit. We combine these two instructions and introduce a macro-operation instruction, $ldpc\_in$. To implement steps 11 and 12 of Algorithm 2, the new instruction, $ldpc\_out.(vp)$, is introduced to flush E_{m1}, E_{m2}, P_m, and S_m from LDPC accelerators and store them in BUF3 and BUF4. The additional new assembly instructions are listed below.

The New Assembly Instructions

1. *ldpc_mem2buf* Addr[Mem],Addr[BUF1],Addr[BUF2],S1,S2

   : send a control signal to the memory controller

   : the controller loads L_n,L_n^{update} from a memory and aligns the data with shift amounts (S1, S2) in BUF1 and BUF2

2. *ldbuf3* V3,V4,Addr[BUF3]

   : load V3=E_{m1}, V4=E_{m2} from BUF3

3. *ldbufs* V1,V2,P1,P2,Addr[BUF1],Addr[BUF2],Addr[BUF4]

   : load V1=L_n, V2=L_n^{update}, P1=P_m, P2=S_m from BUF1, BUF2, BUF4

4. *ldpc_in* V1,V6

   : 1) calculate $L_{n,m}$ with V1=L_n and V6=$L_n^{update} - E_{m,n}^{curr}$

   : 2) update E_{m1},E_{m2},P_m,S_m in LDPC accelerators with $L_{n,m}$.

5. *ldpc_out.v* V7,V8,Addr[BUF3]

   : extract V7=E_{m1}, V8=E_{m2} from LDPC accelerators and store them in BUF3

6. *ldpc_out.p* P3,P4,Addr[BUF4]

   : extract P3=P_m, P4=S_m from LDPC accelerators and store them in BUF4

The overhead of adding these new instructions is the increased instruction bit width and the instruction decoder complexity.

3.4.5 Scalability Issues

The proposed architecture supports different values of z and W_r corresponding to the different code sizes and code rates mandated by the IEEE 802.16e standard.

The memory configuration described in Section 4.2 handles the more difficult case of when $z > W_{simd}$. Larger z results in more computations and so a larger W_{simd} would help in achieving higher decoding throughput. The penalty is the larger area, both is terms of datapath and memory, and larger power. The parameter W_r affects the decoding throughput (number of iterations in Algorithm 2). Since it also affects the buffer size and P_m, S_m registers in the LDPC accelerators, the architecture has to be designed for the largest value of W_r.

3.5 Analysis

In this section, we study the required memory and buffer size, and also analyze the improvement in the decoding throughput due to the memory organization, datapath accelerators and assembly instruction support.

3.5.1 Memory Size Analysis

LDPC decoding process consists of computationally simple operations and multiple memory operations. As a result, if the memory is not organized properly, then it is highly likely that the SIMD pipeline would have to wait for the data to arrive. In a typical implementation, there are four main values that are to be stored: L_n, $L_{n,m}$, $E_{n,m}$, and shuffle information. For n=2304 and R=5/6 LDPC codes outlined in the IEEE 802.16e standard, a brute-force decoding method needs 3.456GB for storing the $L_{n,m}$ and $E_{n,m}$ values. Even if we consider only non-zero elements, the storage still requires 30KB (15KB+15KB), which is a still large memory space for an SDR

platform. Therefore, a new scheme to reduce memory space should be considered. There is no way to reduce the storage of L_n because the data is used to decide the final decoded bit value. However, the storage for $L_{n,m}$ and $E_{n,m}$ can be significantly reduced.

To reduce $E_{n,m}$ storage size, we exploited the fact that there are only two possible $E_{n,m}^{new}$ values for check node m: E_{m1} and E_{m2}. This two-minimum method reduces the required memory space by a factor of $W_r/2$. For the case mentioned above, the storage requirement for $E_{n,m}$ values is reduced to 1.5KB. Also, instead of storing all $L_{n,m}$ values, we store L_n^{update} values, thereby reducing the storage by a factor of $m(=4)$ to 3.75KB.

Storage	Size(B)	Ex.(KB)
MEM: L_n, L_n^{update}	$4n$	9
BUF1: L_n	$2W_{simd}W_r\lceil\frac{z}{W_{simd}}\rceil$	3.75
BUF2: L_n^{update}	$2W_{simd}W_r\lceil\frac{z}{W_{simd}}\rceil$	3.75
BUF3: E_{m1}, E_{m2}	$4W_{simd}\lceil\frac{z}{W_{simd}}\rceil\frac{n(1-R)}{z}$	1.5
BUF4: P_m, S_m	$2W_r\lceil\frac{z}{W_{simd}}\rceil\frac{n(1-R)}{z}$	0.94

Table 3.1: Memory/Buffer requirements for n=2304 and R=5/6 LDPC code in the IEEE 802.16e standard

Table 3.1 summarizes the memory and buffer requirements for a block size n, code rate R=k/n, and (W_c,W_r)-LDPC code. We list the memory requirements for n=2304 and R=5/6 LDPC code (the IEEE 802.16e standard) when $W_{simd} = 32$, $W_r = 20$, and $z = 96$ under the column 'Ex.' in the table.

3.5.2 Throughput Analysis

The data path accelerators, the memory units, and the new instructions all help in increasing the decoding throughput. For the n=2304 and R=5/6 LDPC code in the IEEE 802.16e standard and for NUM=10, the achievable clock cycle reductions for each of the enhancements are shown in Table 3.2. Here 40000 is the number of cycles in the original SODA implementation.

	Reduction in Cycles	Reduction Percentage
LDPC Accelerators	5760(40000)	14.4 %
Memory Units	6912(40000)	17.3 %
New Instructions	4608(40000)	11.5 %

Table 3.2: Cycle reductions due to enhancements

The proposed SODA PE is implemented in 0.18um technology and is clocked at 400MHz. The LDPC decoding throughput for n=2304 and R=5/6 LDPC code can be boosted from 18.3 Mbps to 30.4 Mbps using the proposed enhancements. With technology scaling, the decoding throughput is expected to increase to around 62.2 Mbps in 90nm technology.

The area and power overhead in the datapath and memory is quite small. For instance the area of the memory controller and LDPC accelerators is negligible (5.37%) compared to the original design. However the complexity of adding CISC-type instructions requires careful evaluation.

3.6 Summary

In this chapter, we presented a software-hardware co-design case study of LDPC decoder for SDR. We first provided an overview of LDPC codes and then showed how LDPC decoding can be done by the SDR architecture. Next we showed how use of datapath accelerators, memory buffers and additional instructions can be used to improve the decoding throughput performance. We implemented a scalable LDPC decoder for the IEEE 802.16e standard. Our results show that we can achieve 30.4 Mbps decoding throughput for n=2304 and R=5/6 LDPC code.

CHAPTER 4

Customizing Wide-SIMD Architectures for H.264

In recent years, the mobile phone industry has become one of the most dynamic technology sectors. Mobile computing systems are not limited only to wireless signal processing. The increasing demands of multimedia services such as high-definition video, audio, and 3-D graphics on the cellular networks have accelerated this trend.

This chapter presents a low power SIMD architecture that has been tailored for efficient implementation of H.264 encoder/decoder kernel algorithms. Several customized features have been added to improve the processing performance and lower the power consumption. These include support for different SIMD widths to increase the SIMD utilization efficiency, diagonal memory organization to support both column and row access, temporary buffer and bypass support to reduce the register file power consumption, fused operation support to increase the processing performance, and a fast programmable crossbar to support complex data permutation patterns. The proposed architecture increases the throughput of H.264 encoder/decoder kernel algorithms by a factor of 2.13 while achieving 29% of energy-delay improvement on

average compared to our previous SIMD architecture, SODA.

4.1 Introduction

In the past decade, mobile devices have rapidly proliferated. Today's devices not only support advanced signal processing of wireless communication data, but also multimedia services such as video encoding/decoding, interactive video conferencing and image manipulation. All of this requires a powerful processor which has to be very power-efficient.

H.264 is a state-of-the art video compression standard of ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). This standard provides higher quality video with lower bit rates than earlier standards and has been adopted in many of current and next generation video applications. For instance, both the Bluray Disc and HD-DVD format ratified H.264 as one of three mandatory video compression codecs for High Definition DVD, and the Digital Video Broadcast (DVB) also selected the use of H.264 for broadcast television.

In this chapter, we present a programmable wide SIMD architecture that has been optimized for H.264. The wide-SIMD architecture like SODA [1] is customized to exploit the characteristics of the H.264 kernel algorithms with following features: 1) support of multiple SIMD widths to increase the SIMD utilization efficiency, 2) diagonal memory organization to avoid memory access conflict, 3) bypass and buffer support to reduce the register file (RF) power consumption, 4) fused operation support to speed up the processing, and 5) a fast programmable crossbar to support

complex data shuffle operations. The proposed architecture is similar to AnySP [18], but customized more for video codecs.

The rest of the chapter is organized as follows. Section 4.2 gives a brief overview of H.264 encoder/decoder. Section 4.3 introduces the new architectural features incurred by H.264 algorithms and Section 4.4 describes the modified processing element (PE) architecture in SODA architecture. Section 4.5 shows how H.264 kernel algorithms are mapped on the modified SIMD architecture. Section 4.6 presents the throughput and power analysis of the augmented architecture. Section 4.7 introduces the related work and Section 4.8 concludes the chapter.

4.2 H.264 CODEC

Video compression is being actively considered for mobile communication systems because of the increasing demand of multimedia services on mobile devices. In this chapter, we focus on H.264 because it is representative of contemporary video coding standards and achieves better performance than earlier standards such as MPEG-1, MPEG-2, MPEG-4, and H.263.

Fig. 4.1 shows the block diagram of H.264 encoder and decoder. The encoder includes two dataflow paths: a forward path (left to right) and a reconstruction path (right to left) [26]. The dataflow of the decoder contains the reconstruction path (shown in shaded blocks).

The H.264 encoder processes an input frame or field Fn in macroblock units. Each macroblock is encoded using inter-prediction or intra-prediction. In the inter-

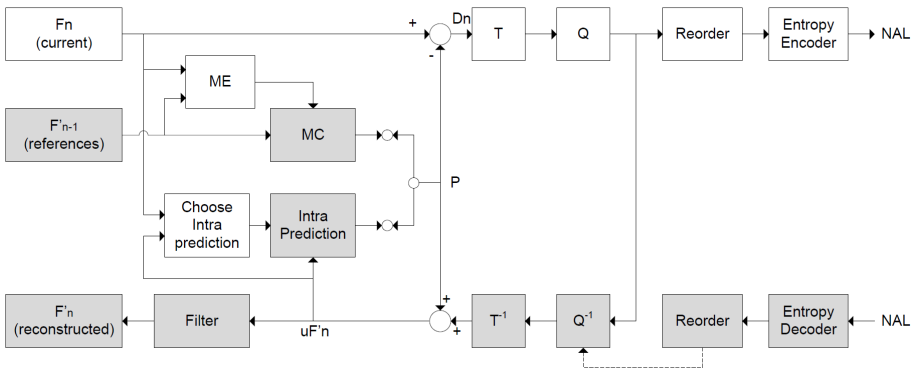

Figure 4.1: H.264 encoder/decoder reference design. ME: Motion Estimation, MC: Motion Compensation, T: Transformation, Q: Quantization, NAL: Network Abstract Layer. Grey area represents functional blocks of the H.264 decoder, which is the subset of the H.264 encoder [26].

prediction mode, the predicted P macroblock is formed by motion-compensated prediction from previously encoded frames, and in the intra-prediction mode, P is predicted by the current frame. The P macroblock is subtracted from the current macroblock to produce a residual block Dn that is transformed, quantized, reordered, and entropy encoded. The entropy-encoded coefficients with header information that includes prediction modes, quantizer parameter, motion vector information, etc. form the network abstract layer (NAL) bitstream.

The H.264 decoder receives the compressed bitstream from the NAL. The entropy decoder decodes the bitstream, and after reordering it, the quantized coefficients are scaled and inverse transformed to generate residual block data Dn. Using the header information in NAL, the decoder selects prediction values using either motion compensation or intra-prediction. The predicted block is added to the residual block to generate unfiltered block data uFn which is filtered by a deblocking filter and stored as reconstructed frame or field.

The computational requirements of H.264 video codec depends on video resolution, frame rate, and compression level. For mobile phone applications, the videos are encoded in the QCIF format (176 x 144) at 15 frames per second (fps). On the other hand, Bluray videos are encoded in 1080p (1920 x 1080) at 60 fps interlaced. The H.264 standard also defines several profiles, which use different compression algorithms. In this chapter, we focus on the baseline profile. We study the following algorithms: intra-prediction, deblocking filter, motion compensation - interpolation, and motion estimation because these algorithms contribute the most to the processing time and power consumption.

4.3 H.264 Algorithm Analysis and Design Decisions

In this section, we analyze key algorithms in H.264 and propose several architectural design decisions to improve the processing performance and power efficiency. This analysis led to the introduction of the following customizing features: 1) multiple SIMD widths 2) diagonal memory organization, 3) bypass and temporary buffer support (partitioned RF), 4) fused operation, and 5) programmable crossbar.

4.3.1 Multiple SIMD Widths

Table 4.1 shows the workload profiling for the key H.264 kernel algorithms. The other important computational kernels such as transform, quantization, and entropy

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

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 4.1. 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 address 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 4.1, 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.

4.3.2 Diagonal Memory Organization

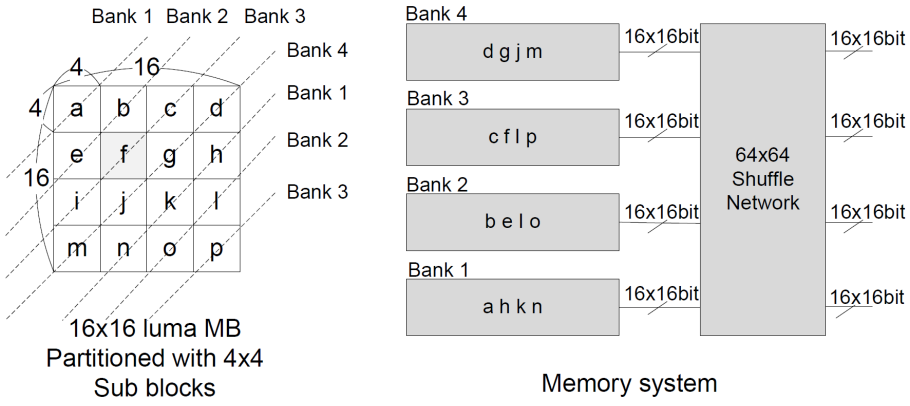

Figure 4.2: Diagonal memory organization and shuffle network, which allows the horizontal and vertical memory access without conflict. The 64x64 shuffle network realigns 64 16-bit data.

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.

Fig. 4.2 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.

4.3.3 Bypass and Temporary Buffer Support

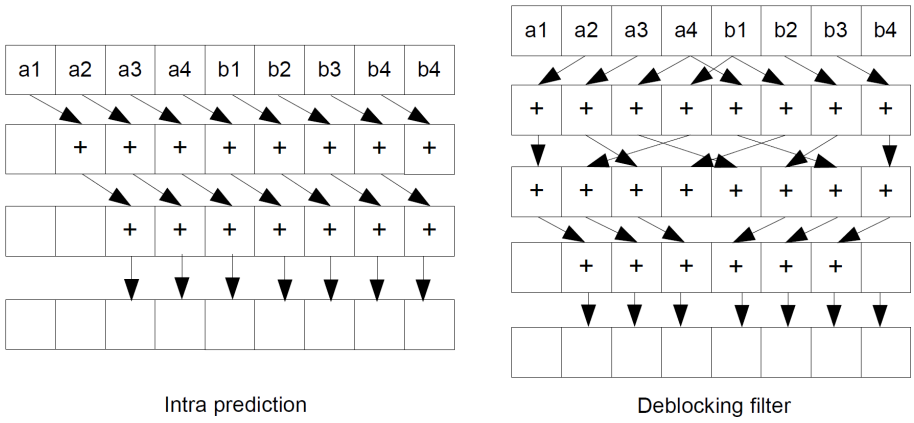

Figure 4.3: Subgraphs for the inner loops for two H.264 kernels; The bypass path is not shown for simplicity.

Fig. 4.3 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 [27] and [28], which show that storing short-lived data and bypassing

RF reduce the power consumption and increase the performance.

4.3.4 Fused Operation

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 4.2: Instruction pair frequency for H.264 kernel algorithms

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 4.2 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 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.

4.3.5 Programmable Crossbar

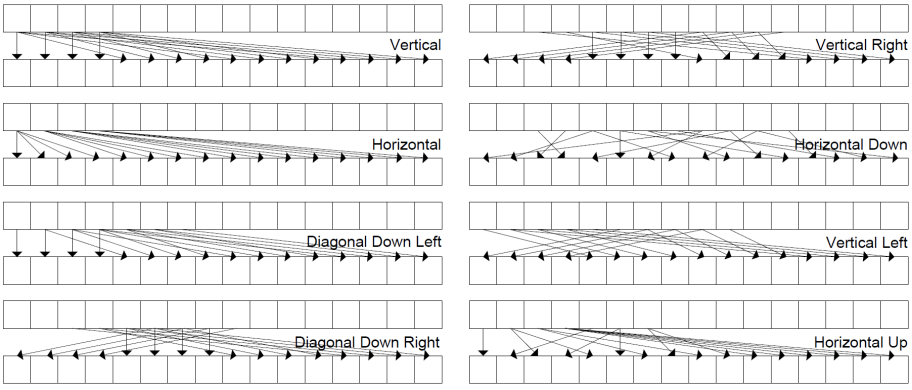

Figure 4.4: Permutation Patterns for H.264 Intra-prediction Modes

Fig. 4.4 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 [1] 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.

4.4 Proposed Architecture

In this section, we describe the customized wide-SIMD architecture which includes the features proposed in Section 4.3. Features such as configurable SIMD datapath, temporary buffer, bypass network and SRAM-based crossbar have also been incorporated in our recent architecture, AnySP [18]. The design of the functional unit and the multibank memory structure is, however, special to the proposed architecture.

4.4.1 PE Architecture

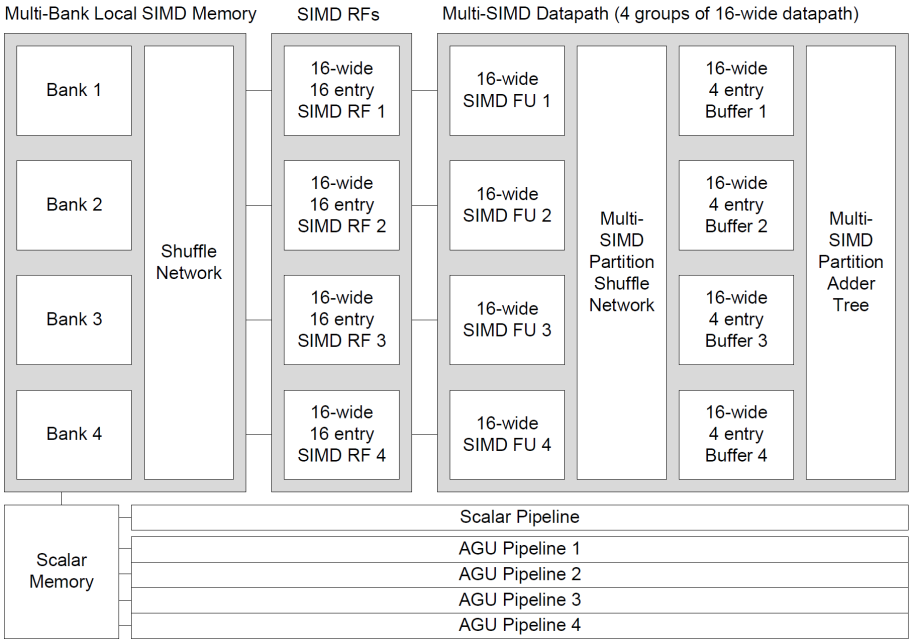

Figure 4.5: PE architecture consists of multi-bank local SIMD memory, SIMD RFs, multi-SIMD datapath, scalar pipeline, four AGU pipelines dedicated to four 16-wide SIMD partitions, and DMA (not shown here)

Fig. 4.5 shows the proposed PE architecture. It is similar to SODA shown in Figure 2.1 in that it consists of a SIMD pipeline, a scalar pipeline, and an AGU pipeline. The SIMD datapath consists of four groups of 16-wide SIMD units that can be functioned as eight groups of 8-wide, two groups of 32-wide or one 64-wide SIMD datapath. Each 16-wide 16-bit SIMD datapath consists of 16-wide 16-entry RF, 16 functional units (FUs) supporting fused instructions, partitioned 16-wide 4-entry RF (temporary buffer) and an adder tree that supports the summation of 2,4,8, and 16 elements. The 16-wide SIMD partitions are glued by multi-SIMD partition shuffle network and data within each 16-wide SIMD units can be shuffled using predefined shuffle patterns by a programmable crossbar. Also, multi-SIMD partition adder tree supports the function of the summation of 32 and 64 elements.

The local memory consists of four memory banks; each bank is 16-wide 16-bit 256-entries (8KB). The four AGU pipelines work for four local memory banks. The scalar and AGU pipeline share the same SIMD local memory using a scalar memory buffer which can be accessed sequentially. AGU pipeline also functions as scalar pipeline for each SIMD datapath. Details of these architectural features are described in the rest of this section.

4.4.2 SIMD Partitioning

As described in Section 4.3.1, H.264 kernel algorithms have different natural vector widths. When the processor's SIMD width is smaller than the natural vector width, the performance drops because the natural vector has to be split into many small

vectors and handling these vectors requires additional work. On the other hand, if the processor's SIMD width is larger than the natural vector width, some of the SIMD lanes are idle, thereby wasting power. Therefore, multiple SIMD partitioning is chosen to support both small SIMD-width algorithms having a large amount of TLP and large SIMD-width algorithms having little TLP.

As can be seen in Fig. 4.5, a 64-wide SIMD datapath is broken into four groups of 16-wide SIMD datapath units. This can be further broken into eight groups of 8-wide SIMD units. Each 16-wide SIMD datapath can be combined to exploit more data parallelism such as 32-wide and 64-wide with the support of the multi-lane shuffle network.

4.4.3 SIMD Functional Units

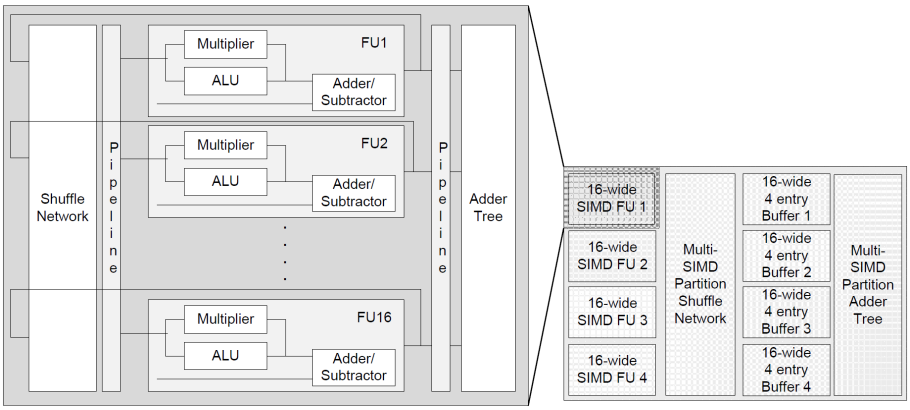

Figure 4.6: 16-wide SIMD Functional Unit

Fig. 4.6 shows the 16-wide SIMD functional unit, which consists of a 32x32 shuffle network, a functional unit (multiplier, ALU, simple adder/subtractor) and a 16-wide

adder tree. The shuffle network supports any permutation pattern using two 16-wide vectors. This shuffle network also stores a small number of shuffle patterns in the module to support fast permutation between 16 functional units. The functional units support instruction pairs such as multiplier-add and add-shift described in Section 4.3.4. A 16-wide adder supports the sum of 2, 4, 8, and 16 elements. The other characteristic of the functional unit is support of saturation arithmetic. For 2's complement signed 8-bit data, the results of the arithmetic units are saturated to +127 and -128, and for unsigned data, to 255 and 0. This saturation feature is very important for operations in the deblocking filter kernel.

4.4.4 Temporary Buffer and Bypass Support

To alleviate the problem of high power consumption of register files (RFs), two techniques are applied: temporary buffer (partitioned RF) and data bypass network support. Each SIMD lane has a 4-entry temporary buffer that stores intermediate data (short-lived values) to decrease the amount of main RF accesses. This small RF consumes less power than the main RF and also helps to reduce register pressure of the main RF. Typical writeback stage is modified to support data forwarding bypass by explicitly directed instructions. Instructions dictate functional units where to fetch data (from main RFs, from temporary buffers or from bypassed data).

4.4.5 Multi SIMD Partition Shuffle Network

Due to data access complexity in H.264 algorithms and proposed memory system, data needs to be shuffled within a SIMD partition or between SIMD partitions. The multi-SIMD partition shuffle network is placed next to four groups of 16-wide SIMD functional units to support data transfer between SIMD partitions. This large shuffle network also allows the processor to function as four SIMD pipelines connected in serial. This feature is useful when a signal processing algorithm have little TLP.

4.4.6 Multiple Output Adder Tree Support

In some H.264 algorithms, the operation of wide vector inner sum ($s = v[0] + v[1] + ... + v[N-1]$) occurs frequently. Examples of this operation are matrix multiplication operation of DCT and SAD calculation for motion estimation. Though H.264 algorithms usually require the sum of 2, 4, 8, and 16 pixel values, the 64-wide multiple SIMD partition adder tree supports other output possibilities such as 32 and 64. The multiple outputs are stored back into temporary buffers and written back to the main RFs if necessary.

4.5 Mapping of H.264 Kernels

In this section, we describe how the main H.264 kernels are mapped onto the proposed architecture.

4.5.1 Intra Prediction

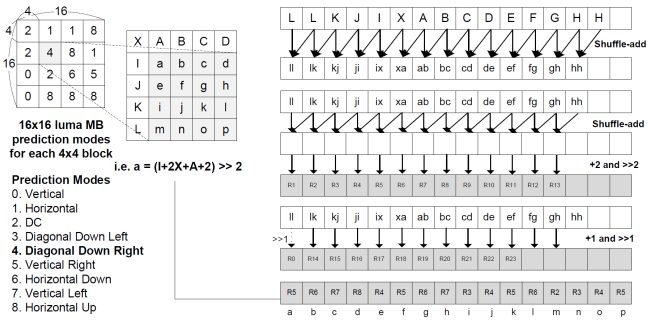

Figure 4.7: Mapping a 16x16 luma macroblock intra-prediction process on the proposed architecture. Example of the Diagonal Down Right intra-prediction for a 4x4 sub block (grey block) is presented with fused operations.

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 $(A, B, C, D, I, J, K, L, X)$ using 16 SIMD lanes. 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 crossbar and an adder tree. Fig. 4.7 shows how to compute the partial intermediate values that are reused for the six prediction modes. 16 SIMD lanes are used to generate two sets of partial sums for a 4x4 sub block with fused operations such

Prediction Mode	Shuffle 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 4.3: Shuffle patterns for six intra prediction modes for 4x4 luma

as shuffle-add and add-shift. After generating R0 to R23, these intermediate values are distributed to the 16 SIMD lanes by a shuffle network. Table 4.3 shows how to shuffle the partial sums for each prediction mode. The use of partial sums results in significant reduction in the number of instruction cycles in the encoder. The intra-prediction calculations in the encoder are very parallel and four groups of 16-wide SIMD datapath can be utilized in parallel. However, in the decoder, there are dependencies in the processing order. For example, in Fig. 4.8, the $A6$ macroblock requires $A1$, $A2$, $A3$, and $A5$ macroblocks to be predicted first. Fig. 4.8 shows a processing order in which four macroblocks are processed at the same time, thereby utilizing all SIMD lanes.

4.5.2 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. 4.9 shows how deblocking filter

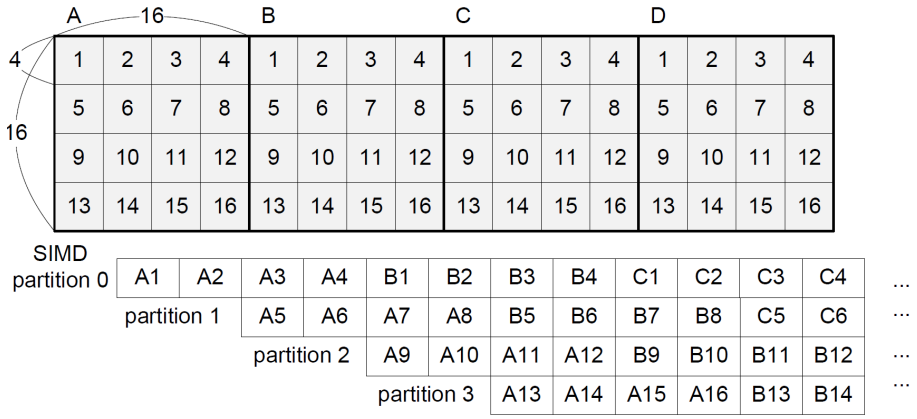

Figure 4.8: Mapping macroblocks into SIMD partitions such that all SIMD lanes are utilized

process is mapped on the SIMD pipeline. To utilize all SIMD lanes, 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 by a shuffle network in the memory system to generates eight groups of horizontally aligned eight pixel values. Each SIMD partition exploits fused shuffle-add operations followed by round operations to produce filtered pixel values.

4.5.3 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. 4.10. The equation is modified to reduce

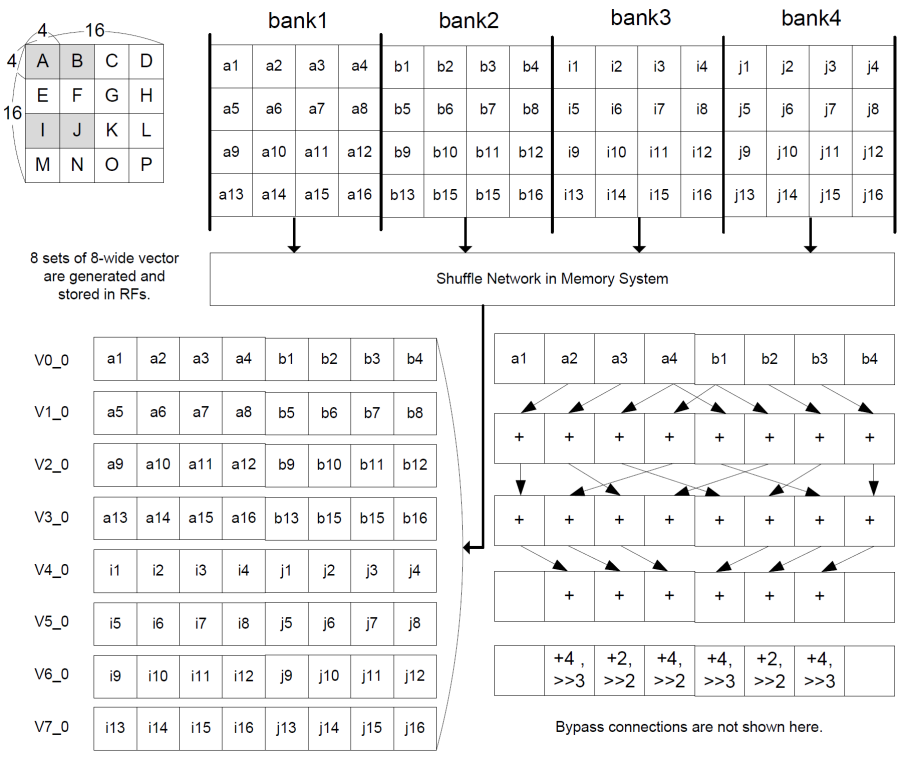

Figure 4.9: Mapping a deblocking filter process when BS (Block Strength)=4.

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. 4.10. As can be seen in Fig. 4.10, the first row of a 16x16 block is loaded to SIMD RFs by using a shuffle network, and the partial sums and differences are stored in temporary registers. A subset of these values are shuffled and summed with an adder tree to obtain the half-pixel estimate. Eight groups of 8-wide SIMD datapath handle the interpolation process for each row. Once the half pixel estimates have been calculated for a particular row, we can use them to compute the quarter pixel values.

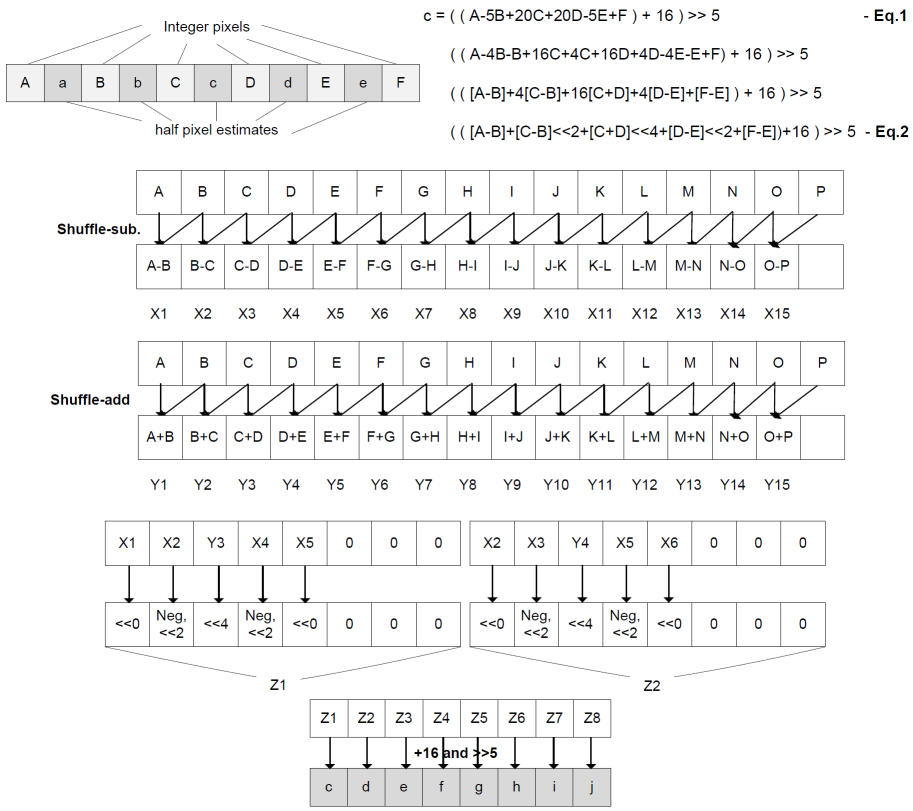

Figure 4.10: Example of interpolation of motion compensation (half-pel).

4.5.4 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. 4.11 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 to a SIMD register, and the pixels in the shaded 4x4 block $(f1, g1, h1, e2, j1, ..., a4)$ in the search area are obtained using memory loads and shuffles. The SAD value is

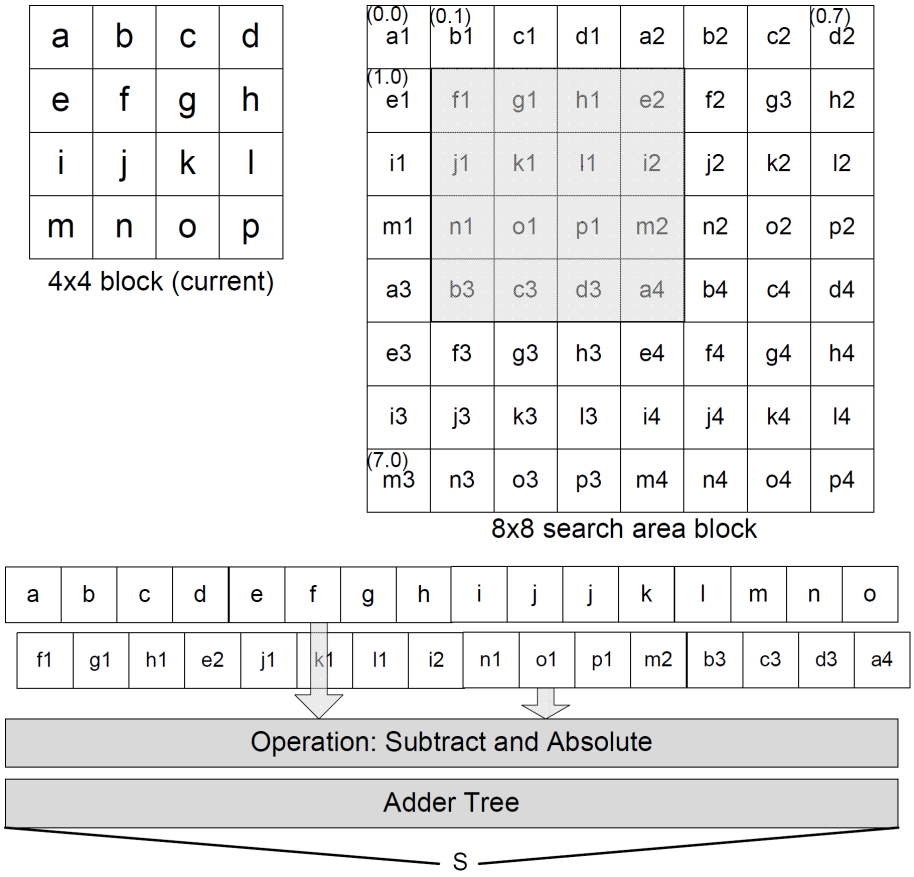

Figure 4.11: Mapping a motion estimation process for a 4x4 block on the proposed architecture; The search area is 8x8.

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. The motion estimation process is highly parallel and four groups of 16 SIMD lanes are utilized to generate four SAD values at a time.

4.6 Results and Analysis

4.6.1 Methodology

			Area		H.264 Decoder	
	Components	Units	Area mm^2	Area %	Power mW	Power %
PE	SIMD Data Mem (32KB)	2	4.88	34.15%	5.14	6.29%
	SIMD Register File (16x1024bit)	2	1.59	11.13%	14.95	18.27%
	SIMD ALUs, Multipliers, and SSN	2	2.25	15.75%	22.43	27.41%
	SIMD Pipeline+Clock+Routing	2	0.59	4.13%	11.68	14.28%
	SIMD Buffer (128B)	2	0.41	2.87%	1.70	2.08%
	SIMD Adder Tree	2	0.09	0.63%	0.52	0.64%
	Intra-processor Interconnect	2	0.47	3.29%	4.67	5.71%
	Scalar/AGU Pipeline & Misc.	2	0.61	4.27%	6.72	8.21%
System	ARM (Cortex-M3)	1	0.6	4.20%	2.5	3.05%
	Global Scratchpad Memory (128KB)	1	1.8	12.5%	10	12.22%
	Inter-processor Bus with DMA	1	1.0	7.00%	1.5	1.83%
Total	90nm (1V @300MHz)		**14.29**	100%	**81.81**	100%
Est.	65nm (1V @ 300MHz)		**7.46**		**25.76**	
	45nm (1V @ 300MHz)		**3.89**		**20.36**	

Table 4.4: Summary of Area and Power Running H.264 CIF video at 30fps

The RTL Verilog model of SODA processor [1] was synthesized in TSMC 180nm technology, and the power and area results for 90nm technology were estimated using a quadratic scaling factor based on Predictive Technology Model [49]. The proposed architecture was implemented in the RTL Verilog model and synthesized in TSMC 90nm using Synopsys physical compiler. The PE area is 25% larger than SODA's estimated 90nm PE area. The clock frequency is targeted for 300MHz, while SODA was targeted for 400MHz. The area and power breakdown of this architecture running H.264 CIF video at 30fps are presented in Table 4.4. This video encoder consumes about 81mW at 90nm, which is within the requirements for mobile video.

4.6.2 Results

Fig. 4.12 shows the speedup of proposed architecture over SODA for the H.264 kernel algorithms. The improvement is broken into several architectural enhancements: wider SIMD width (from 32 to 64), fused operation, buffer+bypass support, and single cycle programmable crossbar. The wider SIMD width allows H.264 algorithms to operate on twice as many pixels and results in 72% of performance improvement over SODA. The fast programmable crossbars expedites the data alignment process, and accounts for another 25% improvement. The fused operations and buffer+bypass support also helps to boost the speed by about 16%. The energy-delay product for the H.264 kernel algorithms are presented in Fig. 4.13. On average, there is a 29% of energy-delay improvement due to lower clock frequency, reduced memory and register file access supported by crossbar, fused operation, and buffer+bypass support.

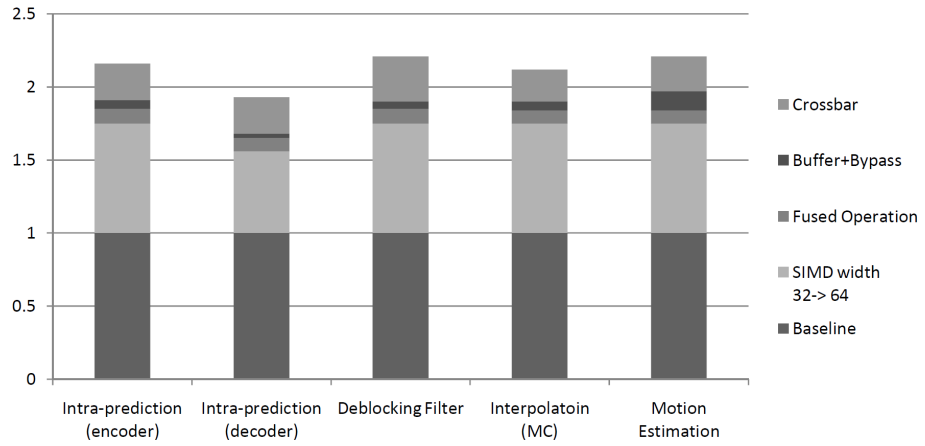

Figure 4.12: Speedup over SODA for the key H.264 algorithms. The improvements are broken down into several architectural enhancements - wider SIMD width, fused operation, buffer+bypass support and fast programmable crossbar.

Table 4.5 compares the power performance of our architecture with state-of-the-

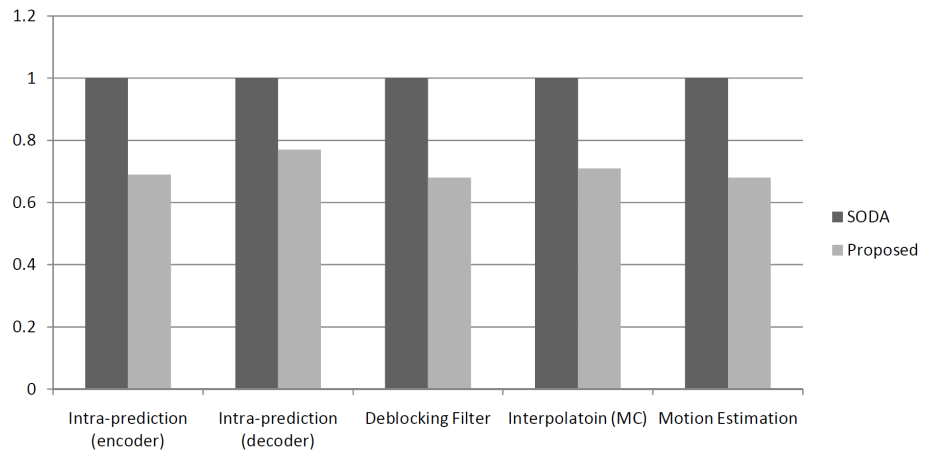

Figure 4.13: Normalized Energy-Delay Product for H.264 kernel algorithms compared to SODA.

art designs for H.264 baseline encoding. We use power consumption per pixel/sec (mW/(Mpix/s)) as the metric. Although the ASIC solution [40] outperforms the programmable solutions, our proposed architecture has programmable flexibility and consumes significantly less power compared to TI's DSP solution.

	ISSCC2007 [40]	**TMS320DM6446** C64x+ DSP [41]	**This work** 2 PEs
Resolution	720x480	720x480	352x288
Technology	130nm	90nm	90nm
Supply Voltage	0.9V	1.2V	1.0V
Clock Freq.	30MHz	594MHz	300MHz
Power consumption	27mW	415mW	68mW
Power efficiency (mW/(Mpixel/sec))	2.6	40	22

Table 4.5: Comparison with state-of-the art H.264 encoders.

4.7 Related Work

There have been several architectural solutions for H.264/AVC. Many of them are specialized architectures for key kernels such as motion estimation, motion compensation [30], [32], interpolation [31] and deblocking [33], [34]. An important consideration in all these architectures is efficient memory access. For instance the deblocking filter architectures reduce the number of memory accesses by manipulating data stored in shift registers in [33] and using vector registers and VLIW processing in [34]. Reducing the overhead in memory accesses and data alignment in multimedia processing has been addressed in systems such as MediaBreeze [29] by adding hardware support for address generation, looping etc.

Efficient techniques for mapping H.264 onto multiprocessor platforms have been proposed in [35], [36], [37], [38]. While [35] focused on efficient partitioning of data, [36] proposed a high speed multithreading implementation of the H.264 video encoder. The implementation in [37] focused on efficient scheduling and memory hierarchy for the H.264 video encoder for HDTV applications. A hybrid task pipelining scheme which greatly reduced the internal memory size and bandwidth was presented in [38].

Recently a FGPA based architecture, Video Specific Instruction Set Processor, has been proposed in [39]. The architecture consists of hardware accelerators for inter prediction and entropy coding, and specialized instructions for a programmable processor for the rest of the kernels.

4.8 Summary

The mobile multimedia processor requires high-performance low-power solutions for high quality video and wireless protocols. General purpose processors, digital signal processors and ASICs are typically combined to meet this requirement. Such a heterogeneous solution is inefficient in terms of area, power, and flexibility. In this chapter, we presented a software-hardware co-design case study of H.264 codec for a wide-SIMD architecture. Based on the characteristics of H.264 kernel algorithms, we proposed several key architectural enhancements including SIMD partitioning, diagonal memory organization system, bypass and temporary buffer support, fused operation support, and area and energy efficient programmable crossbar use. Our results show that we can achieve 2.13x speedup and 29% energy-delay improvement for the H.264 codec over a wide-SIMD architecture, SODA.

CHAPTER 5

Diet SODA: A Power-Efficient Processor for Digital Cameras

Power has become the most critical design constraint for embedded handheld devices. This chapter proposes a power-efficient SIMD architecture, referred to as Diet SODA, for DSP applications. The key design idea is to apply near-threshold operation on a single instruction and multiple data (SIMD) architecture to significantly lower the power consumption. The major features of Diet SODA are very wide SIMD width, scatter/gather data prefetcher, and dual mode operation. A case study was performed on digital still camera (DSC) applications; the results show that Diet SODA achieves ∼130x better performance and ∼340x better energy efficiency than a DSP solution.

5.1 Introduction

As today's devices not only support advanced signal processing of wireless communication data but also provide for richer sets of various applications, power dissipation

has become a more important design constraint. Increasing power consumption leads to increasing energy costs as well as impacts chip reliabilities. Therefore, more power-efficient processors for embedded DSP applications are highly required. Among many DSP applications, high resolution cameras have become an integral part of most cell phone designs. As a result, the market for these cameras has mirrored the spectacular growth in mobile phones [44]. Furthermore, the expectation is that these mobile cameras produce an image whose quality should approach that of high quality digital still cameras (DSCs). Therefore, a DSC processor needs to be of high-performance to support a large amount of image data and perform the DSC image processing tasks in a highly energy-efficient manner in order to conserve critical battery life for other phone applications.

Traditionally, the DSC image signal processing pipeline is implemented in digital signal processors (DSPs) or application specific integrated circuits (ASICs). DSP-based solutions [43] support high flexibility and handle various DSC algorithms, but they suffer from lower performance and higher energy consumption than ASIC solutions. ASIC-based solutions [45, 46] are highly specialized and optimized for the DSC image signal processing pipeline, but such designs lack flexibility and require longer design time. Therefore, to achieve high processing performance efficiency at low cost while maintaining programmability, hybrid architectures [47] are employed. In these designs, ASICs are typically used for a preview mode, where high processing capabilities are desired, and DSPs are adopted for picture-taking and post-processing modes, where flexibility is more important. However, such a heterogeneous solution

is inefficient to build and maintain.

To address these challenges, this chapter presents a power-efficient programmable architecture, Diet SODA, that has been optimized for DSC image signal processing. Diet SODA exploits near-threshold operation [54] on a wide-SIMD architecture — SODA [1]. Near-threshold operation offers a new opportunity for mobile applications such as DSCs to reduce power consumption. However, the reduction in power consumption comes at a cost of a ∼10x performance degradation. Diet SODA overcomes these hurdles by exploiting architectural features specific to near threshold operation. The key features of Diet SODA are 1) very wide SIMD width to exploit the significant amount of data level parallelism (DLP) inherent in DSC applications, which helps overcome the frequency loss from operating in the near-threshold region; 2) scatter-gather data prefetcher to support 2D memory access enabled by the latency difference between the full voltage SIMD memory and SIMD data engine operating at near-threshold voltage; and 3) dual voltage modes where the SIMD data engine operates at either full or near-threshold voltage based on processing demands.

The rest of the chapter is organized as follows. Section 5.2 gives a brief overview of near-threshold operation. Section 5.3 analyzes the computational characteristics of DSC signal processing algorithms for preview mode, picture-taking mode and post-processing mode. Section 5.4 introduces Diet SODA, the low-power DSP processor for DSCs with an analysis of design choices created by using near-threshold operation. Section 5.5 presents the performance, power, and comparison analysis of Diet SODA. Section 5.6 discusses the related work and Section 5.7 concludes the chapter.

5.2 Near Threshold Operation

The original SODA architecture was targeted for applications that require substantial processing to meet time-critical tasks in software defined radio applications on a limited energy budget. There are a considerable number of applications, such as DSC, that operate on an even tighter energy budget, but where timing is less critical. A paradigm shift is necessary for these applications to further reduce energy consumption.

The near-threshold computing offers an opportunity for applications, such as DSC, to reduce energy further. In order to do so, the design must overcome one hurdle, the 10x increase in delay. This delay impacts the ability of designs to meet more stringent real time constraints without scaling the voltage higher and losing energy efficiency. However, in cases where the application can be parallelized, simply using more near-threshold processing elements can meet the timing constraint with greater efficiency. Near-threshold operation, therefore, has a natural synergy with data parallel environments like SIMD. In a SIMD architecture, the number of functional units can be increased to help meet a timing critical code, provided the application has sufficient DLP.

In addition, the fact that near-threshold operation decreases frequency offers several new and interesting design choices related to the memory system. First, memory devices that are slower and more energy efficient can be used to replace previously timing critical memories. This will help to reduce the overall energy consumption of the chip. Second, multiple accesses to memory can be performed in one near-

threshold clock cycle. This means that data patterns that were impossible at full speed could be achieved using new hardware that scatter-gathers memory requests. And, third, the slower memory allows for elements originally designed to hide long memory latency, i.e. caches and register files, to be turned off or eliminated.

5.3 DSC Algorithm Analysis

5.3.1 DSC Signal Processing Pipeline

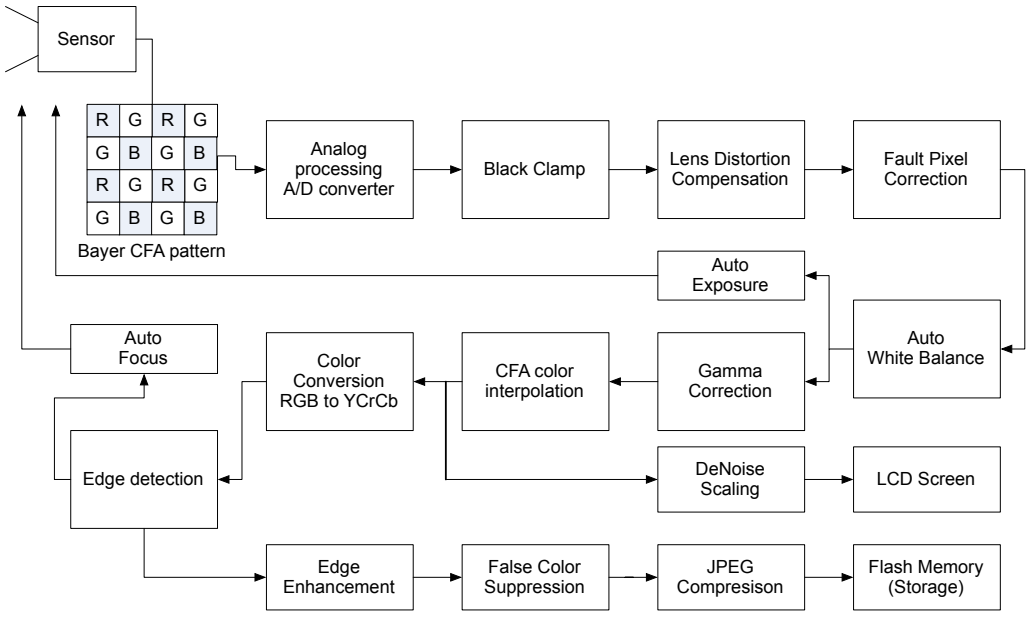

Figure 5.1: A typical DSC image signal processing pipeline [42], [43]

Figure 5.1 shows a typical DSC image signal processing pipeline [43], which performs multiple processing steps to generate a high-quality image. The image is first captured by a CCD or CMOS sensor using a Bayer-pattern [55] color filter array (CFA). Then, the image is digitized with a 10- or 12-bit A/D converter. The *Black*

Clamp adjusts the pixel values by subtracting a black offset value from all pixel values. The *Lens Distortion Compensation* adjusts the brightness of pixels depending on the spatial locations and the *Fault Pixel Correction* interpolates defective pixels with neighboring pixels. After all these pre-processing steps, *Auto White Balance* computes the average brightness of each color component and balances the energy of the colors. Based on the brightness information, *Auto Exposure* appropriately adjusts the CCD or CMOS exposure time and gain. After the white balanced image pixels are compensated by *Gamma Correction*, *CFA color interpolation* uses the one-color-per-pixel Bayer-pattern image to interpolate and generate the full color (R, G, and B) resolution for each pixel. The RGB color pixels are filtered by *De-Noise* and scaled down and sent to the LCD screen in preview mode. In picture-taking mode, the noise-filtered images are transformed to the YCrCb color domain. *Edge Detection* detects edges to help *Auto Focus*, and *Edge Enhancement* is performed. Next, *False Color Suppression* occurs, and finally the image is compressed by using *JPEG Compression* and stored in flash memory. Later, post-processing tasks such as *Histogram Calculation*, *Histogram Equalization*, and *Spatial Frequency Filtering* are used to enhance the quality of the stored images.

5.3.2 Characteristics of DSC Algorithms

In this section, we analyze the key algorithms in the two modes (preview and picture-taking) of DSC signal processing pipeline and post-processing tasks to find opportunities for improving the processing performance and energy efficiency.

5.3.2.1 Data Level Parallelism

Mode	Task	SIMD	Scalar	Overhead
Preview / Picture-Taking	Black Clamp	100%	0%	0%
	White Balance	100%	0%	0%
	Auto Focus	71%	14%	14%
	Gamma Correction	0%	100%	0%
	CFA Interpolation	84%	3%	13%
	Auto Exposure	74%	11%	15%
	Color Conversion	100%	0%	0%
	Edge Detect/Enhance	81%	2%	17%
Post-Processing	Histogram Equalize	37%	44%	19%
	Spatial Freq. Filter	77%	3%	20%

Table 5.1: Data level parallelism analysis for DSC image signal processing algorithms. Instructions are categorized into three groups: SIMD, scalar, and overhead instructions.

Table 5.1 presents the data level parallelism (DLP) analysis of the DSC signal processing algorithms. Instructions are broken down into three categories: SIMD, scalar, and overhead workloads. The SIMD workload consists of traditional arithmetic/logical functional operations and load/store operations that can be executed in SIMD-fashion. The scalar workload consists of instructions running only on the scalar datapath such as control instructions and address generations for local SIMD and scalar memories. The overhead workload consists of instructions to assist SIMD computations and scalar computations such as shuffle operations, predication operations, and data movements between the SIMD datapath and scalar datapath. The workloads of each category are calculated based on hand-written assembly codes and are weighted by dynamic execution frequency.

As can be seen in Table 5.1, most of the DSC signal processing algorithms have significant DLP. Exceptions are *Gamma Correction* and *Histogram Equalization*, where

memory access patterns inhibit parallelization. The remaining DSC signal processing algorithms can be grouped into three categories.

(1) Pixel Independent Kernels: In this set of kernels, some basic arithmetic/logical and multiply-and-accumulate (MAC) operations are applied on every pixel independently. Therefore, these kernels can easily be mapped onto a SIMD architecture. *Black Clamp*, *Color Space Conversion*, *Brightness/Contrast Enhancement*, and *Hue/Saturation Enhancements* fall into this category. Although *Gamma Correction* is also a pixel-independent operation, this kernel cannot be easily parallelized on a SIMD architecture because each SIMD lane has to access different memory locations at the same time.

(2) Pixel Dependent Kernels: This set of kernels includes *CFA Interpolation*, *Edge Detection/Enhancement*, and *Spatial Frequency Filtering* that operate on pixels in a 2D neighborhood. The size of the 2D neighborhood is typically 3x3, though 5x5 or 7x7 sizes are also used. Traditional processor architectures spend more than half of the total instructions aligning the 2D data [42]. Therefore, for these kernels, 2D data access must be supported. This is done by a combination of multi-bank memory organization and a SIMD shuffle network.

(3) Statistics Gathering Kernels: The statistical information of the whole or partial frame is gathered for *White Balance*, *Auto Exposure*, *Histogram Calculation* and *Histogram Equalization*. Some of these kernels can be supported by a SIMD adder tree. *Histogram Calculation* is another kernel where memory access patterns inhibit parallelization for a SIMD architecture.

5.3.2.2 Instruction Pairs

Next, we study pairs of instructions that could possibly be combined to reduce the number of register file accesses. Many current DSP processors exploit this feature to increase performance while decreasing energy.

	Black Clamp / White Balance	**Color Conv. / Color Correct**	**CFA Interpolation**	**Edge Detect / Edge Enhance**
***LD-OP-ST**	100 %	-	10 %	-
MULT-ALU	-	57 %	-	50 %
ALU-ALU	-	10 %	17 %	15 %
SHFL-ALU	-	-	6 %	-
****Others**	-	33 %	67 %	35 %

Table 5.2: Instruction pairs for some DSC image processing algorithms. *LD-OP-ST represents an operation chain — LOAD-ALU/MULT-STORE. **Others represents instructions that cannot be paired.

Table 5.2 shows the frequently used instruction pairs of key DSC image processing algorithms. The multiply-and-accumulate (MAC) instruction is most frequently used because many of these algorithms are based on 2D convolution and 2D matrix multiplication, all of which involve vector inner-product calculations. The ALU-ALU instruction pair is also commonly used in some of the kernels. In particular, *Black Clamp* and *White Balance* algorithms have a very simple operation flow, which is to load pixel values, add/subtract or multiply them with predefined values, and then store them back to memory. This simple operation flow opens a possibility of not using a SIMD register file (RF) thus reducing energy consumption.

5.4 Diet SODA Architecture

In this section, we propose a power-efficient architecture, referred to as Diet SODA, for DSC processors. Diet SODA exploits key characteristics of the DSC image processing algorithms described in Section 5.3. This architecture operates in two modes: 1) dual voltage (DV) mode to handle low power applications such as the DSC image processing pipeline, and 2) the full voltage (FV) mode to handle advanced wireless communications. In DV mode, the memory operates at full voltage but the SIMD pipelines operate at near-threshold voltage. In FV mode, the SIMD data engines operate at full voltage as well.

5.4.1 Diet SODA PE Design

Figure 5.2 shows the architectural details of a single processing element (PE) of Diet SODA. The PE contains two different voltage domains: full voltage (FV) and dual voltage (DV). DV domain operates at either full or near-threshold supply voltage. The PE consists of: 1) multi-bank SIMD memory; 2) scalar memory; 3) SIMD data prefetcher; 4) SIMD pipeline; 5) scalar pipeline; and 6) 4-wide address generation unit (AGU) pipeline.

The SIMD pipeline consists of a 128-wide 16-bit datapath with a SIMD register file (RF), 128 functional units, 128 4-entry buffers used for intermediate data, a SIMD shuffle network (SSN), and a multi-output adder tree. The SIMD datapath consists of four groups of 32-wide SIMD units. With the support of a multi-banked SIMD memory and an 4-wide AGU pipeline, these groups of SIMD partitions can work on

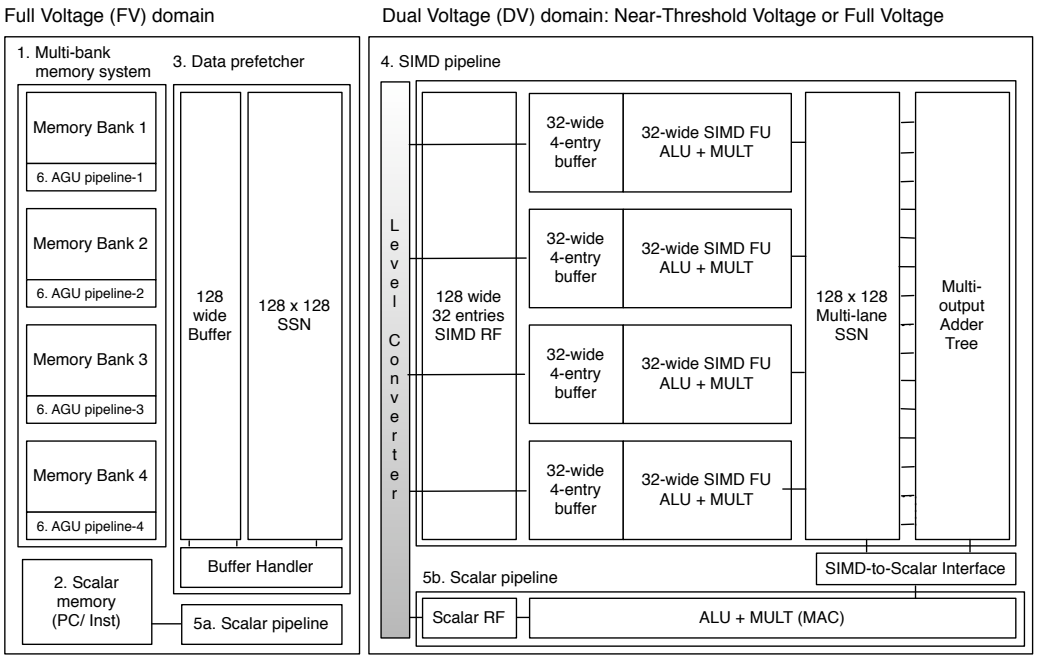

Figure 5.2: Diet SODA processing element (PE) for DSCs. The PE contains two different voltage domains: full voltage (FV) and dual voltage (DV). DV domain operates at either full or near-threshold supply voltage. The PE consists of: 1) multi-banked SIMD memory; 2) scalar memory; 3) SIMD data prefetcher; 4) SIMD pipeline; 5a) scalar pipeline in full voltage domain; 5b) scalar pipeline in dual voltage domain; and 6) 4-wide address generation unit (AGU) pipeline.

four different memory sections concurrently. There are two scalar pipelines, one in each voltage domain; both pipelines consist of one 16-bit datapath and are used to perform sequential algorithms in addition to coordinating the SIMD units. The 4-wide AGU pipeline handles memory address calculation for the 4-bank SIMD memory and the data prefetcher.

5.4.2 SIMD Pipeline Width

Although near-threshold operation allows circuits to consume significantly less power, the processing performance also degrades. To compensate for the degraded

performance, the number of SIMD lanes is increased.

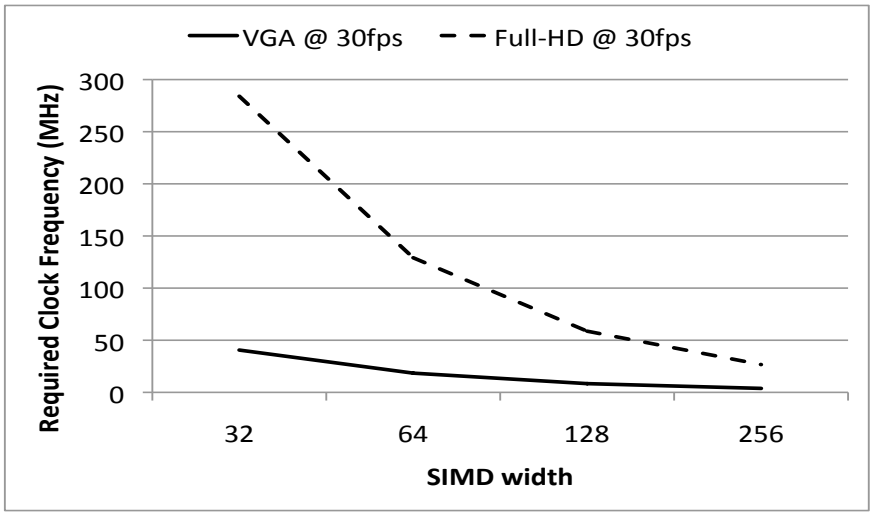

Figure 5.3: Minimum clock frequencies based on different SIMD width configurations to run the preview mode of DSC signal processing pipeline shown in Figure 5.1.

The DSC signal processing pipeline for a VGA-size (640x480) image and a full-HD (1920x1080) image at 30 fps is used as the evaluation point to decide the number of SIMD lanes. Figure 5.3 shows the minimum clock frequency required for VGA and full-HD for different SIMD width configurations — 32, 64, 128, and 256. Thus, to process full-HD images at 30 fps, a 32-wide SIMD pipeline needs to operate at more than 270MHz, while a 256-wide SIMD pipeline needs to operate at around 30 MHz.

To investigate how much voltage/frequency scaling can be achieved while still meeting the performance requirements, the power consumption for each SIMD width configuration was measured. First, a representative test circuit was laid out in IBM 90nm technology, parasitic extraction was performed and annotated. Then, SPICE simulations were done to determine the voltage, frequency, and power characteris-

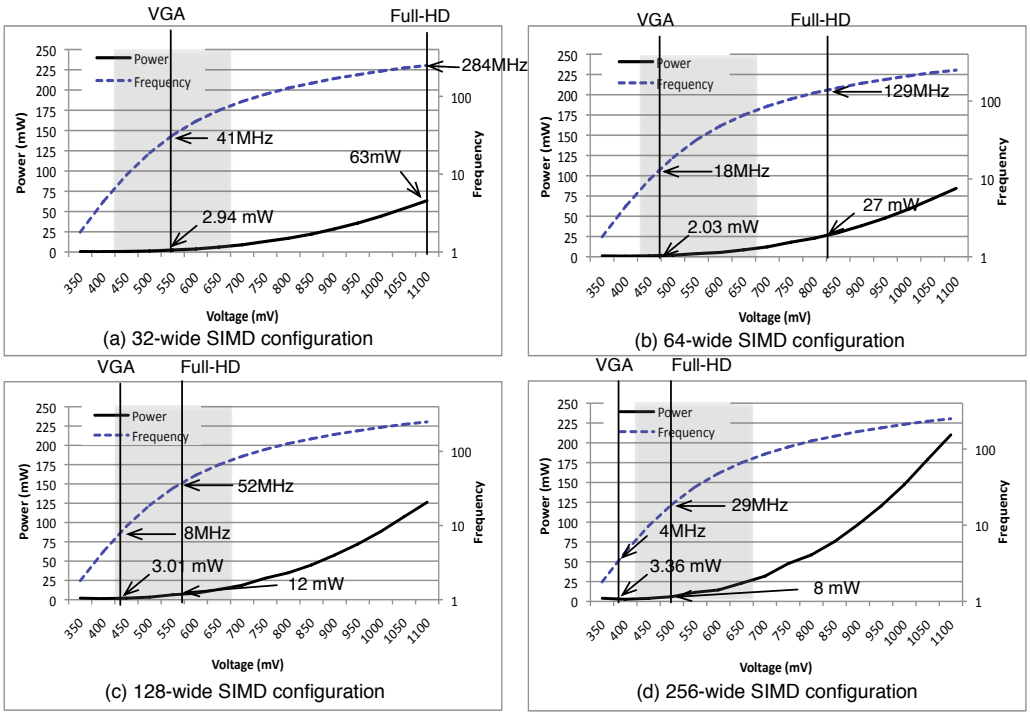

Figure 5.4: Near-threshold operation is applied to four different SIMD width configurations: 32, 64, 128, and 256. Solid vertical lines provide guidelines for the minimum supply voltage necessary to meet VGA and full-HD processing demands. Gray boxes represent the near-threshold regions.

tics at different supply voltages. To obtain power numbers, the SIMD pipeline logic was then synthesized with Synopsys Physical Compiler and scaled to match the representative test circuit. Figure 5.4 shows power consumption and achievable clock frequencies depending on the corresponding supply voltage for each candidate SIMD width. The solid vertical lines provide guidelines on what the minimum supply voltage is required to process VGA and full-HD images at 30 fps. The results show that although a 32-wide SIMD data engine is capable of handling VGA processing requirements, to support full-HD images, a SIMD width of greater than 32 is required. On the other hand, wider SIMD widths do not always guarantee better energy efficiency

due to the additional hardware and critical path delay increases, resulting in a higher minimum clock frequency. In this chapter, a 128-wide SIMD configuration is chosen to maximize the benefit of using near-threshold operation while maintaining the real time processing constraints of both VGA and full-HD. With this configuration, the supply voltage needs to be 600mV using a clock frequency of 50MHz.

5.4.3 Scatter-Gather Data Prefetcher

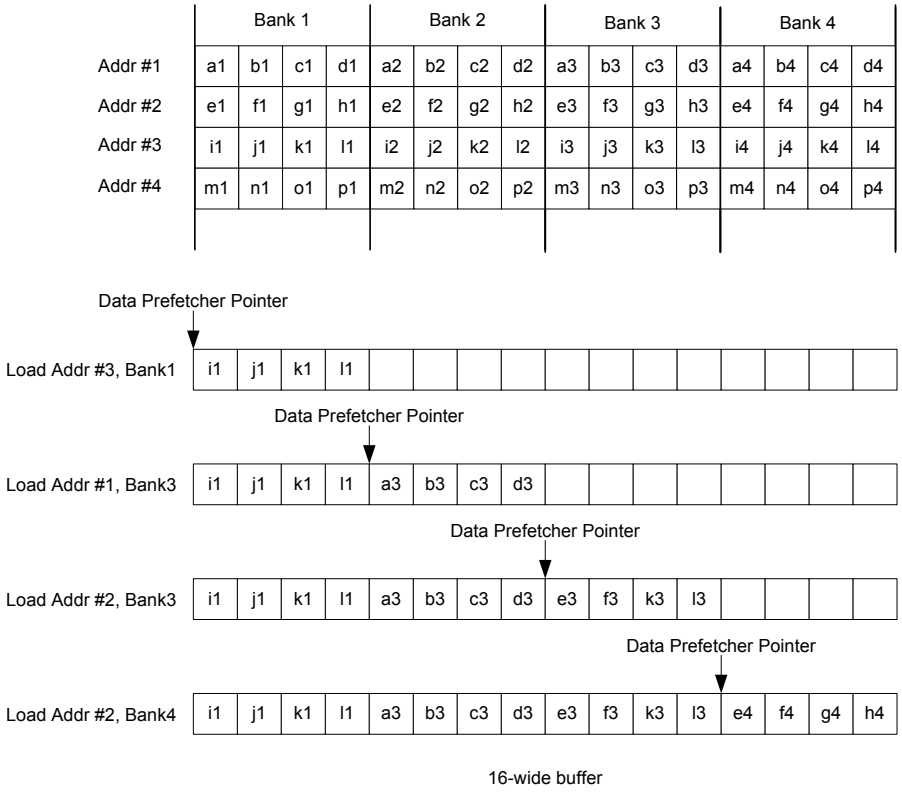

Figure 5.5: Example of complex data shuffling with 4-bank 4-wide SIMD memory, SIMD data prefetcher, and 16-wide buffer.

While operating in the DV mode, the SIMD memory operates significantly faster than the SIMD pipeline. Therefore, two-dimensional (2D) data accesses can be

achieved by performing multiple memory accesses to the same memory banks in a single cycle of the SIMD pipeline. There is also sufficient time to perform complicated shuffling operations before delivering the data. Because of these non-traditional memory access patterns and additional shuffling capabilities, a significant reduction in the required number of SIMD instructions can be obtained.

Figure 5.5 shows the process of data alignment using the SIMD data prefetcher. First, the required data is read from a multi-bank memory. Then the data prefetcher stacks the data in the location indicated by the data prefetcher pointer. The pointer then advances to the next data section and repeats the process for the next load operation. In addition, with the support of SSN, more complex shuffle operations can be implemented.

5.4.4 Operating Modes

In this section, dual voltage (DV) and full voltage (FV) modes in Diet SODA are described. Table 5.3 provides the configuration of each component of Diet SODA PE for each mode.

5.4.4.1 DV Mode

In the DSC signal processing pipeline, preview and picture-taking tasks are performed in DV mode because these tasks do not require very high data processing rates. Consequently, the supply voltage of the SIMD data engine is operated at near-threshold voltage to significantly lower energy consumption. More specifically,

	Components	DV Mode	FV Mode
PE	1. Multi-Bank SIMD Memory	on	on
	2. Scalar Memory	on	on
	3. Data Prefetcher - Buffer/Buffer Handler	on	off
	3. Data Prefetcher - SSN	on	on
	4. SIMD pipeline - SIMD RF	off	on
	4. SIMD pipeline - Other modules except SIMD RF	on@NTV	on
	5a. Scalar pipeline	on	off
	5b. Scalar pipeline	on@NTV	on
	6. 4-wide AGU pipeline	on	on

Table 5.3: Architectural modules that are turned on and off for dual voltage (DV) and full voltage (FV) modes.

the SIMD pipeline and scalar pipeline (5b in Figure 5.2) in the DV domain operate at near-threshold voltage, while the SIMD memory, scalar memory, SIMD data prefetcher, and 4-wide AGU pipeline operate at full voltage. As can be seen in Table 5.3, the SIMD RF is switched off because the latency of the SIMD memory is much lower than that of SIMD data engine so the SIMD pipeline is capable of directly handling data from the SIMD memory. This results in a reduction of energy consumption by eliminating SIMD RF accesses. The 4-entry buffer in each SIMD lane operates as a small RF to hold recently produced values for consumption by subsequent instructions.

5.4.4.2 FV Mode

Recent DSCs support video recording at full-HD (1920x1080) resolution. This necessitates additional processing capability, and therefore requires Diet SODA to operate in FV mode. In this mode, the SIMD pipeline operates at full voltage along with the SIMD memory and 4-wide AGU pipeline. On the other hand, the SIMD

data prefetcher is turned off because there is no time slack between the SIMD memory and the SIMD data engine to prefetch data in advance. Also, the scalar pipeline (5a in Figure 5.2) in the FV domain is turned off and another scalar pipeline (5b in Figure 5.2) in the DV domain works for the overall system. In this mode, the SIMD RF is switched on so that faster operations are supported.

5.4.5 Buffer and Bypass Network

To reduce the power consumption of register files (RFs), two techniques are applied: short-live value buffering, and data bypass network support. Each SIMD lane has a 4-entry temporary buffer that stores intermediate data (short-lived values) to decrease the amount of SIMD memory accesses and RF accesses. This small buffer consumes less power than the SIMD memory and the main RF and also helps to reduce register pressure in the main RF. The writeback stage is modified to support data bypass by explicitly encoding forwarding information into the instructions.

5.4.6 Mapping Examples

5.4.6.1 CFA Interpolation

This section describes how one of the key DSC algorithms, *CFA interpolation*, is mapped onto Diet SODA. The edge-directed CFA interpolation compares horizontal gradients and vertical gradients, and the interpolation method is chosen as described in Figure 5.6-(a). In Diet SODA, the data prefetcher loads Addr#1 and Addr#3 from SIMD memory, and shuffles them to obtain $v0$ by using the data prefetcher and

SSN, Figure 5.6-(b). The aligned $v0$ is then used to calculate the vertical gradients ($v2$), Figure 5.6-(c). Similarly, Addr#2 is loaded and shuffled to obtain $v1$ which is used to calculate the horizontal gradients ($v3$). With these SIMD registers ($v2$ and $v3$), the edge-directed interpolation is easily implemented. For example, based on the comparison of the first elements of the two vectors ($b1 - j1$=ΔH and $e1 - g1$=ΔV), the second elements of the vectors are select-shifted ($(b1 + j1) >> 1$, $(e1 + g1) >> 1$) or add-shifted ($(b1 + j1 + e1 + g1) >> 2$). This series of operations rely upon the data prefetcher and SSN in FV domain and the SIMD pipeline in DV domain.

5.4.6.2 3x3 Convolution

A 3x3 convolution (5.1) is one of the commonly used algorithm in DSC signal processing pipeline such as edge detection and edge enhancement. There are many ways to program this algorithm on Diet SODA and here, Figure 5.7 shows one of the methods how to map 3x3 convolution on Diet SODA.

$$\begin{bmatrix} a_1 & a_4 & a_7 \\ a_2 & a_5 & a_8 \\ a_3 & a_6 & a_9 \end{bmatrix} * \begin{bmatrix} b_1 & b_4 & b_7 \\ b_2 & b_5 & b_8 \\ b_3 & b_6 & b_9 \end{bmatrix} = \sum_{k=1}^{9} a_k * b_k \tag{5.1}$$

The three pixel rows are loaded into three SIMD registers: $v1$, $v2$, and $v3$. The 3x3 mask values are loaded into nine scalar registers. The $v1$ is multiplied by c_{11}, and then $v1$ is shuffle down by 1 and multiplied by c_{12}, and $v1$ is shuffled down by 1 again and multiplied by c_{13}. These three multiplied SIMD registers are added, then $v4$ contains inner-product of the first rows. The $v2$ and $v3$ are process in the same

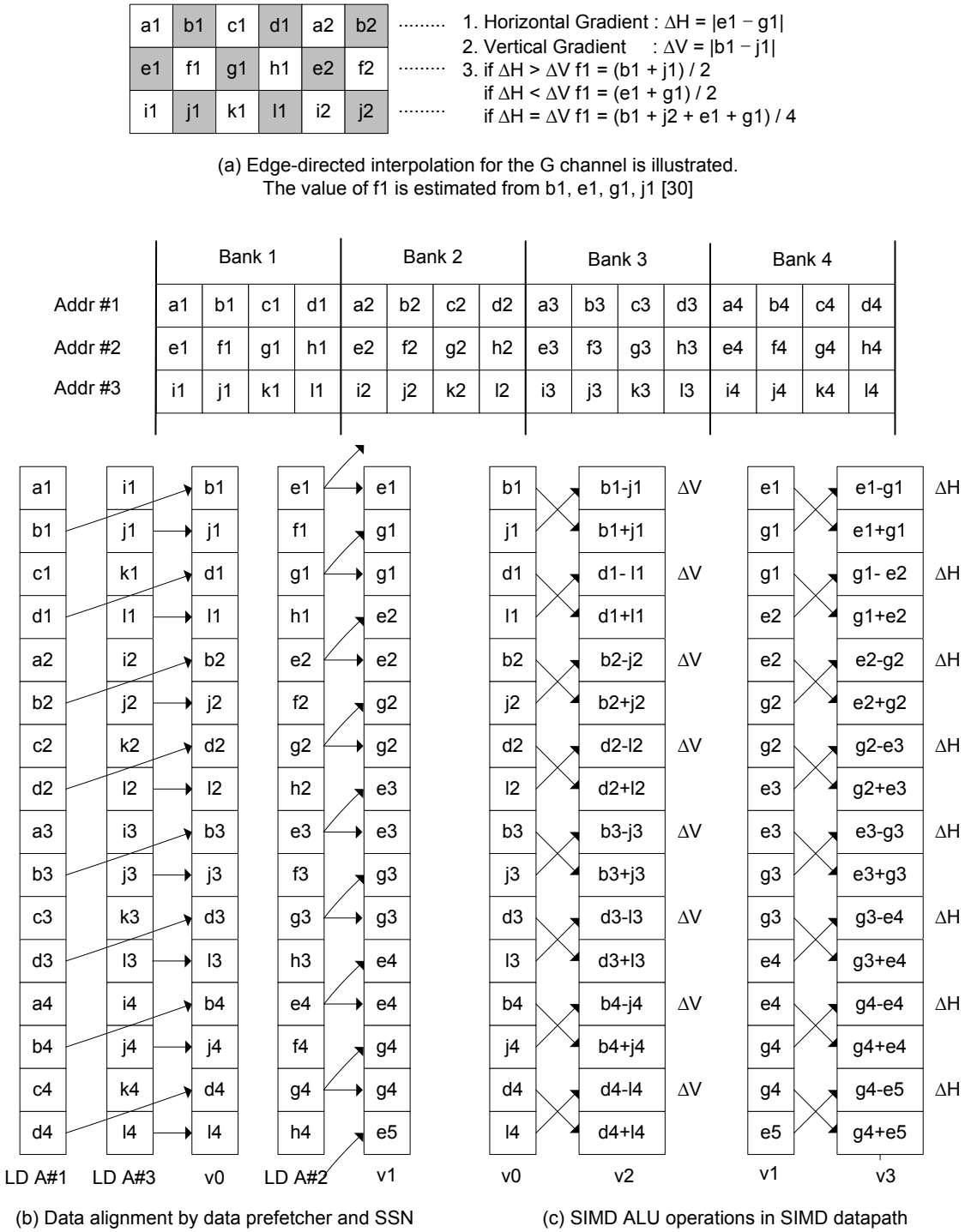

Figure 5.6: An Edge-directed CFA interpolation mapped on Diet SODA.

way to generate $v5$ and $v6$. The SIMD addition of $v4$, $v5$, and $v6$ generate the final convolution values per each lane. For example, the first element is $\sum_{k=1}^{9} a_k * b_k$, and the second element is $\sum_{k=4}^{12} a_k * b_k$. In this way, 3x3 convolution process is parallelizable.

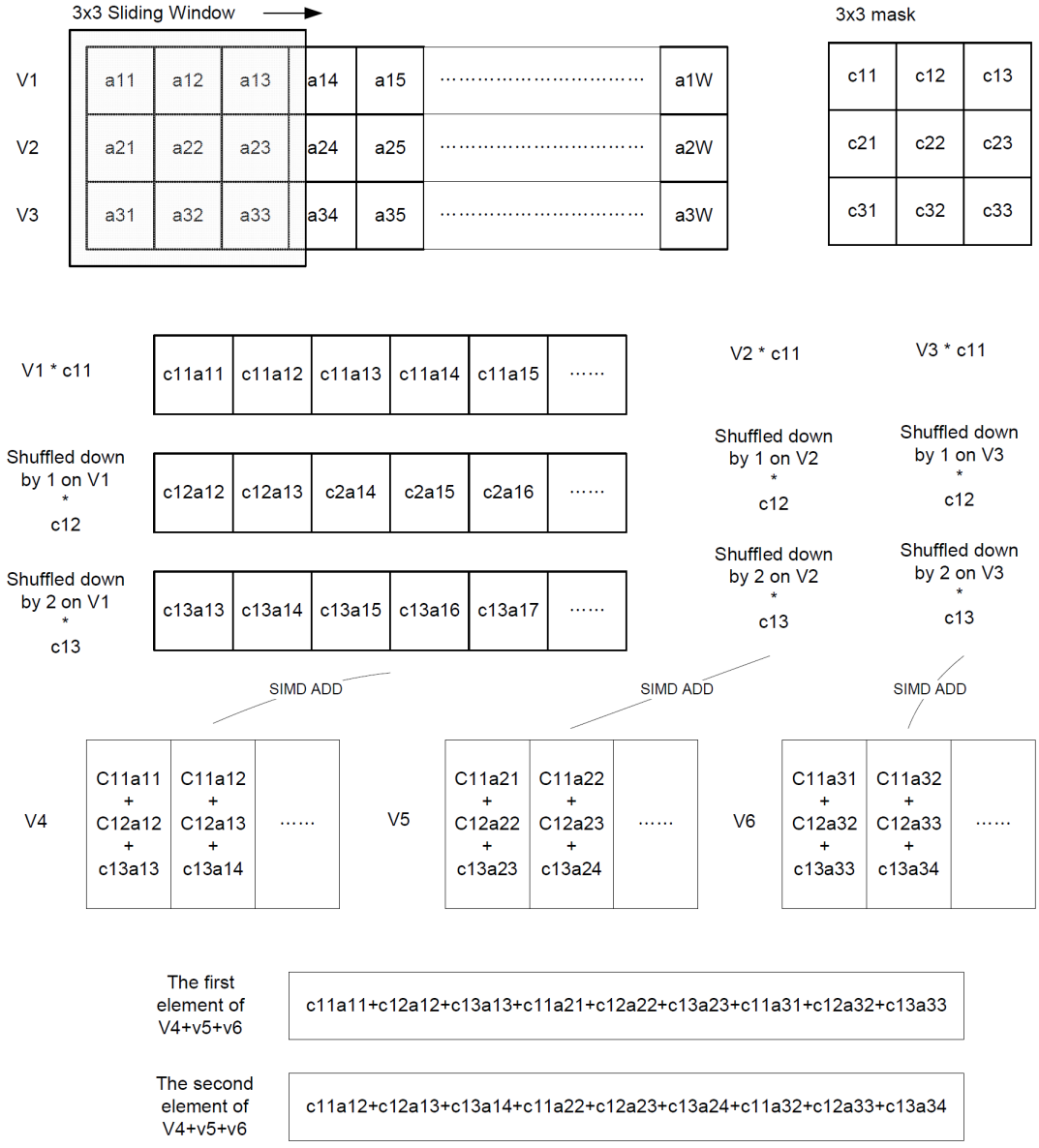

Figure 5.7: A 3x3 Convolution operation mapped on Diet SODA. A 3x3 convolution mask is applied to 3x3 pixels.

5.5 Results and Analysis

5.5.1 Methodology

The DSC image signal processing pipeline algorithms are implemented in C to evaluate system performance, memory requirements, and non-parallelizable bottlenecks. Next, the C benchmark codes [56] are transformed to assembly codes for Diet SODA. The Diet SODA processor is implemented as an RTL Verilog model and synthesized for IBM's 90nm technology using the Synopsys Physical Compiler. The clock frequency is targeted for 400MHz @ 1V, and the power numbers for the SIMD data engine are scaled down for 50MHz @ 600mV by the process shown in Section 5.4.2.

5.5.2 Area and Power

The area and power breakdown of this processor are presented in Table 5.4. The preview mode of full-HD images at 30 fps consumes about 122mW and 1228mW in DV mode and FV mode, respectively. Therefore, the DV mode offers about *sim*10x better power efficiency.

About 68% of the total power dissipation in DV mode is consumed by SIMD memory, the scalar/AGU pipeline, and data prefetcher operating at full voltage. In particular, the SIMD memory and data prefetcher consume a large part of the power because the number of SIMD memory accesses and shuffle operations is increased due to the SIMD RF being switched off. However, the SIMD data engine operating in DV mode consumes *sim*21x less power than the SIMD datapath operating in FV

				DV mode		FV mode	
	Components	**Area (mm2)**	**Area (%)**	**Power (mW)**	**Power (%)**	**Power (mW)**	**Power (%)**
PE	SIMD banked-memory (64KB)	3.41	33%	28	23%	36	3%
	SIMD Register Files (4KB)	1.58	15%	0	0%	310	25%
	SIMD Buffer (1KB)	0.41	4%	3	2%	65	5%
	SIMD ALU/Multiplier, SSN	2.26	22%	23	19%	519	42%
	SIMD Adder Tree	0.12	1%	1	1%	28	2%
	SIMD pipeline+Clock+Routing	0.68	7%	12	10%	213	17%
	Data Prefetcher	1.63	16%	33	27%	27	2%
	Scalar/AGU Pipelines & Misc.	0.18	2%	22	18%	30	2%
Total	90nm(1V@400MHz, 600mV@500MHz)	10.27	100%	122	100%	1228	100%

Table 5.4: Area and Power Summary of Diet SODA for Preview Mode of Full-HD Images at 30 fps. For comparison, the results of both DV mode and FV mode are presented.

mode, which offsets the increased SIMD memory power and highlights the advantage of using near-threshold operation.

5.5.3 Performance

Table 5.5 presents the latencies of DSC processing algorithms - preprocessing (*Black Clamping*, *Lens Distortion Compensation*, *Fault Pixel Correction*, *White Balance*, *Gamma Correction*), *CFA Interpolation*, *Color Space Conversion*, *Edge Detection/Enhancement*, *Scaling*, and *JPEG Compression*. As can be seen in Table 5.5, the preview modes of both VGA and Full-HD images are processed within a time constraint of 33 ms thus meeting the 30 fps requirement.

CFA Interpolation and *Edge Detection/Enhancement* are the most demanding workloads taking about 60% of the processing time. While most algorithms deals with only one color component (R, G, or B) per each pixel location, *CFA interpolation* generates all of three components for each pixel locations. Therefore, the

Task	Latency (VGA)	Latency (Full-HD)
Black Clamp, Distortion Compensation, Fault Pixel Correction, White Balance, Gamma Correction	0.57 ms	3.89 ms
CFA Interpolation	1.02 ms	6.67 ms
Color Conversion	0.36 ms	2.43 ms
Edge Detection Edge Enhancement	0.82 ms	5.29 ms
False Color Suppression Scaling	0.31 ms	2.11 ms
Total	**3.08 ms**	**20.38 ms**

Table 5.5: The Latencies of DSC signal processing pipeline algorithms for the preview mode of a VGA image and a Full-HD image.

memory size and workload for this interpolation algorithm are increased. *Edge Detection/Enhancement* works with only one component per each pixel location, but 3x3 matrix convolutions in this task require significant processing time and shuffling for MAC calculations and realignments.

5.5.4 Comparison with SODA

In this section, we present the performance and energy analysis of key algorithms in DSC signal processing pipeline, namely CFA Color Interpolation, Color Conversion, Edge Detection/Enhancement, and JPEG Compression. Figure 5.8 shows the normalized throughput (higher is better) and Figure 5.9 shows the normalized energy (lower is better) for running these kernel algorithms on Diet SODA FV and DV modes over SODA [1]. In Figure 5.8, each improvement is categorized by several architectural enhancements: wider SIMD width (from 32 to 128), XRAM shuffle

network, buffer+bypass support, fused instruction, and data-prefetcher. All enhancements except data-prefetcher boost the processing throughput by ∼3x for Diet SODA FV mode. However, when we run the Diet SODA in DV mode, the throughput is decreased by ∼10x because of reduced clock frequency and increased hardware overhead. In Diet SODA DV mode, data-prefetcher performs alignment operations on behalf of SIMD datapath, which result in ∼10x speedup again. On average, Diet SODA DV mode decreases the processing throughput by a factor of 2.5. However, Figure 5.9 show that Diet SODA DV mode achieves ∼50% and ∼65% energy improvement over SODA and SODA FV mode respectively. A detailed analysis is presented next.

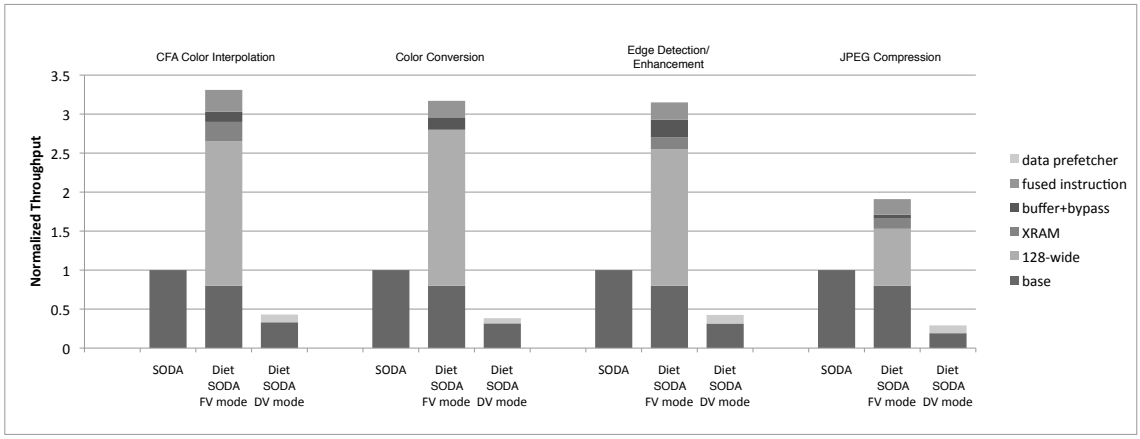

Figure 5.8: Normalized throughput of Diet SODA FV and DV modes over SODA for kernel DSC algorithms. Speedups are broken into five categories: wider SIMD width (128), XRAM crossbar, buffer+bypass, fused instruction, and data prefetcher. Data-prefetcher runs only in DV mode.

CFA color interpolation: While preprocessing algorithms deal with only one color component (R, G, or B) per each pixel location, CFA color interpolation generates all three components for each pixel location. Therefore, the efficient shuffle operation within neighboring pixels is crucial in this kernel algorithm. A wider SIMD

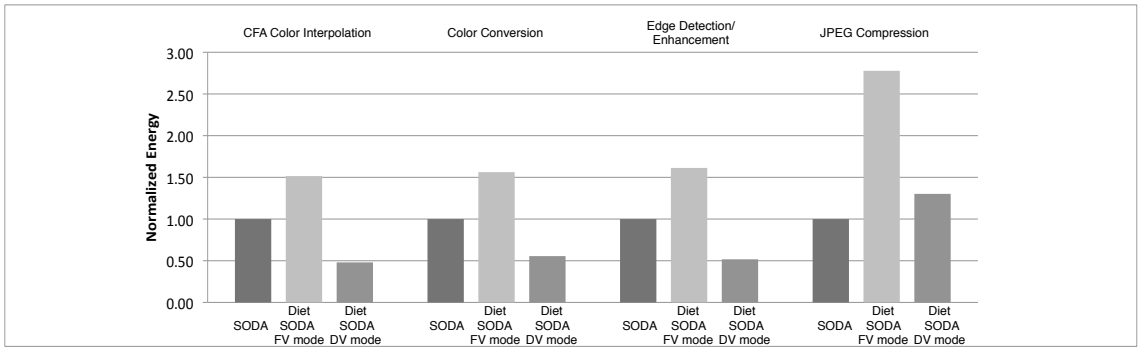

Figure 5.9: Normalized energy for the DSC kernel algorithms over SODA.

width and XRAM crossbar are the main contributors (2.2x) of the speedup. In addition, fused-operation such as shuffle/add and shuffle/subtract helps reducing latency. In DV mode, the SIMD data-prefetcher offloads the shuffling operations from the SIMD pipeline, which achieves an additional 23% of speedup from reduced performance due to near-threshold operations.

Color conversion: As shown in Table 1, Color Conversion contains large amounts of DLP because the main operation of the kernel algorithm is the vector-multiplication. Therefore, the wider SIMD width contributes a large speedup (2x). In Diet SODA DV mode, 2D memory access for vector multiplication is enabled by using SIMD data prefetcher, which makes the algorithm run more efficiently because the SIMD pipeline do not need to take unnecessary clock cycles for alignment operations.

Edge detection/enhancement: Because human eyes are more sensitive to the luminance component (Y) compared to color components (Cr and Cb), *Edge Detection/Enhancement* works with only the Y component. Although this edge detection/enhancement algorithm deals with only one component per each pixel location, a 3x3 matrix convolution is performed requiring significant processing time and shuf-

fling for MAC calculations and alignment respectively. Figure 5.8 shows 1.8x speedup from the wider SIMD width and 0.7x speedup from XRAMs and fused-instructions.

JPEG compression: Discrete cosign transform (DCT), quantization, and entropy decoding are main algorithms for JPEG compression. Although DCT and quantization are well suitable for SIMD datapath, entropy decoding is the representative sequential process including run-length encoding and Huffman coding. Therefore, the speedup using the wider SIMD width is limited by the portion of DCT and quantization execution time as shown in Figure 5.8. The reduced performance induces the worst energy efficiency among other kernel algorithms shown in Figure 5.9.

5.5.5 Comparison With Other Solutions

The DSC image signal processing pipeline in Figure 5.10 [42] is used to compare the performance of Diet SODA with one high-end commercial DSP and one coarse-grained reconfigurable image stream processor — TI TMS320C64x [48] and CRISP [42]. The pipeline is divided into three task groups: 1) color gain adjustment, gamma correction and CFA interpolation; 2) noise reduction and smooth filter; and 3) color space conversion and edge enhancement.

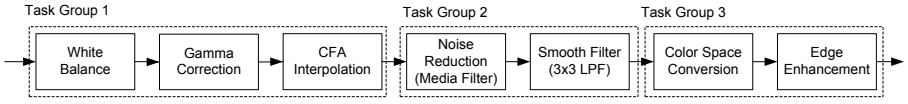

Figure 5.10: A Test DSC image signal pipeline [42]

Table 5.6 shows the execution time comparison with TMS320C64x, CRISP, and Diet SODA for a 4072x2720 image. Results show that Diet SODA is approximately

140x and 1.6x faster than TMS320C64x and CRISP, respectively. The wide SIMD datapath allows the DSC image signal processing algorithms to operate on many pixels at the same time. In addition, scatter-gather data prefetcher helps data alignment issues.

	TMS320C64x [48]	**CRISP** [42]	***Diet SODA PE**
Task Group 1	6440 ms	220 ms	110 ms
Task Group 2	20550 ms	110 ms	80 ms
Task Group 3	9690 ms	110 ms	80 ms
Total	36680 ms	440 ms	270 ms

Table 5.6: Execution Time Comparison with TI TMS320C64x, CRISP, and Diet SODA. Task Group 1 - White Balance, Gamma Correction, CFA Interpolation; Task Group 2 - Noise Reduction, Smooth Filter; Task Group 3 - Color Space Conversion, Edge Enhancement. *Diet SODA operates in DV mode.

Table 5.7 shows comparisons of technology, area, power consumption, and normalized energy with TMS320C64x and CRISP. Normalized power and area results for 90nm technology are estimated using a quadratic scaling factor based on Predictive Technology Model [49]. The results show that the energy efficiency of Diet SODA is more than 340x better than that of TMS320C64x and also comparable to that of the reconfigurable image stream processor, CRISP [42]. Even though we show comparable energy and only slightly improved performance numbers over the CRISP design, our design is more flexible and maintainable because the reconfigurable interconnection and heterogeneous processing elements of CRISP must be manually designed before fabrication.

	TMS320C64x [48]	**CRISP** [42]	**Diet SODA PE**
Tech.	$0.13um$	$0.18um$	$90nm$
Freq.	600MHz	115MHz	400MHz,50MHz
Power	718mW@1.2V	218mW@1.8V	122mW@DV**
Area*	$34.5mm^2$	$1.9mm^2$	$10.3mm^2$
Energy*	11k	9.3	32.4

Table 5.7: Chip Statistics and Energy Comparison with TI TMS320C64x, CRISP and Diet SODA. *Area and energy are normalized to $90nm$ technology. **Diet SODA operates in DV mode - 1V and 600mV.

5.6 Related Work

The real-time constraints of media (image/video) applications require high-performance but low-power processors for portable devices. In addition, as pre-/post-processing to improve image/video quality are becoming more important, flexibility is another important decision factor. To satisfy real-time constraints and programmability, three types of image/video processors have been used: SIMD, stream processors and reconfigurable processors.

SIMD-based processors such as SODA [1], NXP's EVP [8], Sandbridge's Sandblaster [6] and Icera's DXP [4], use multiple processing elements working in SIMD fashion to exploit high data level parallelism. These types of architectures usually support VLIW execution and use software-managed scratchpad memories to meet the real time constraints. Each SIMD processor includes special characteristics such as very wide SIMD width in SODA [1], deeply pipelined execution for chained operations in DXP [4], and multi-threading in Sandblaster [6]. Although many DLP-intensive DSC algorithms are efficiently implemented in SIMD manner [52, 53], SIMD-based processors usually suffer from large power consumption and hardware cost because of

high bandwidth requirements. Diet SODA uses near-threshold operation to reduce energy consumption.

Stream processors such as Imagine [50] and SPI [51] have proved to be efficient solutions for media processing applications. Stream processors organize an application explicitly into streams of data and compute-intensive kernels. A host processor sends stream instructions to the processors and the arithmetic clusters in the processors operate in SIMD fashion. In addition, stream processors employ data locality and concurrency by compounding complex SIMD kernel computations to reduce the number of vector register read/write operations and power dissipation. Although existing stream processors achieve high performance for media applications, complex architectural components are an overkill for DSC applications.

Reconfigurable architectures can be classified into two types: coarse-grain and fine-grain. Coarse-grained reconfigurable architecture such as REMARC [19] have been used for media processing. In addition, ADRES [10] automatically maps applications onto coarse-grained reconfigurable arrays that are tightly coupled to VLIW processors and exploits loop-level and instruction-level parallelism to maximize functional units. XiSystem's XiRisc [13] is an example of a fine-grain reconfigurable architecture.

Diet SODA differs from all these architectures in that it operates in the near-threshold voltage region. This enables new memory system designs and radically reduced power consumption.

5.7 Summary

In this chapter, we have proposed a programmable substrate for an ultra-low power signal processor using near-threshold operation. Near-threshold operation reduces energy but suffers from degraded performance, but this can be overcome by using parallelism. DSC algorithms on SIMD architectures offer an abundant amount of data level parallelism, forming a natural synergy with near-threshold operation. In addition, because memory systems operate at faster rates than SIMD data engines in near-threshold operation mode, scatter-gather prefetcher was introduced to exploit latency difference and lower instruction counts. Diet SODA also uses a dual voltage mode to increase performance for kernels that requires high processing power. Our results show that Diet SODA with a 128-lane SIMD unit operating at 600mV and 50MHz in an IBM 90nm technology can meet the processing requirements of full-HD resolution at 30 fps while consuming only 122mW. This is on the order of 130x better performance and approximately 340x better energy efficiency over a DSP solution, and provides a more flexible solution than equivalently powered ASIC designs.

Although the near-threshold techniques brings a large opportunity to energy-efficient and high-performance architecture designs like Diet SODA, they suffer from large delay variations due to increased process variability. Therefore, our next research steps are to assess the effects of variations in near-threshold operations on a SIMD architecture, and to explore architectural design spaces to tolerate the process variability.

CHAPTER 6

Managing Process Variation in Near-Threshold Wide SIMD Architectures

Near-threshold operation has emerged as a competitive approach for energy-efficient architecture designs. In particular, a combination of near-threshold circuit techniques and parallel SIMD computations achieves excellent energy efficiency for easy-to-parallelize applications. However, near-threshold operations suffer from delay variations due to increased process variability. This is exacerbated in wide SIMD architectures where the number of critical paths are multiplied by the SIMD width. This chapter provides a systematic in-depth study of delay variations in near-threshold operations and shows that delay variations are not that large. As a result simple techniques such as structural duplication, supply voltage margining, and frequency margining are sufficient to mitigate the timing variation problems even in wide SIMD architectures. By employing these simple techniques without complex micro-architectural modification, we show that variation-induced timing errors

in wide SIMD architectures can be effectively reduced at the cost of marginal area and power overhead. As a case study in 90nm technology, a variation-aware near-threshold wide SIMD architecture is presented with the following key enhancements: 1) replication of SIMD functional units to replace underperforming ones and 2) use of an XRAM crossbar to efficiently set up the new error-free SIMD datapath.

6.1 Introduction

A near-threshold wide SIMD architecture for DSC signal processing algorithms, Diet SODA [60], has demonstrated that a combination of near-threshold circuit techniques and wide SIMD platforms achieves significant energy savings while meeting real-time processing constraints. However, such near-threshold designs are impacted greater by process variations than traditional designs, because the on-current (I_{on}) in the near-threshold voltage region is highly sensitive to variations in V_{th}. Increased process variations in advanced technology nodes further exacerbates the problem, providing many challenges for process engineers and circuit designers [62]. These variation-induced timing errors are much more critical in wide SIMD architectures for two reasons. First, the probability that all SIMD datapaths are error-free decreases when variations are severe, because the number of critical paths are multiplied by the SIMD width. Recent work also shows that there is a significant performance drop in SIMD architectures as single-stage-error probabilities increase [63]. Second, commonly used error-tolerating methods such as pipeline stalling or re-execution result in greater performance and power penalties due to problems in one lane impacting

all other lanes. To tolerate variation-induced timing errors in near-threshold operations, complex architectural enhancements have been considered. For example, Synctium [63] proposed decoupled parallel SIMD pipelines and pipeline weaving using decoupling queues and micro-barriers.

In this chapter, we investigate the effect of process variations in wide SIMD architectures operating at near-threshold voltages. Delay variations in the near-threshold regime are first analyzed for present and future technology nodes (90nm, 45nm, 32nm, and 22nm). Our study shows that delay variations in near-threshold operations have been over-estimated in the past. In 90nm technology, although delay variation ($\frac{3\sigma}{\mu}$) at 0.5V in a single gate increases by $\sim$2.5x compared to that at 1V, the variation decreases in a chain of gates. For instance, the variation is only $\sim$1.5x for a chain of 50 gates. This is an example of mean-value theorem where the uncorrelated variations are averaged out over the chain. Working against this effect is the fact that the datapath is a wide SIMD machine, thus increasing the number of these critical paths. Nevertheless, the corresponding performance degradation for such wide systems in 90nm technology is less than 5%. Therefore, simple techniques are sufficient to tolerate and mitigate the timing variation problems. Three techniques are explored in this work: 1) structural duplication to replace underperforming modules, 2) voltage margining to reduce both average delay and its variation, and 3) frequency margining to increase delay margins. The analysis shows a combination of these simple techniques can effectively reduce variation-induced timing errors in wide SIMD architectures with marginal area and power overhead.

We present a case study of an energy-efficient variation-aware wide SIMD architecture in 90nm technology. The architecture is based on Diet SODA [60] and has enhanced features such as 1) simple duplication of SIMD functional units and 2) use of XRAM [66] crossbars. The duplicated SIMD modules serve as spares to replace slow (or faulty) ones. The XRAM crossbars set up correct (error-free) SIMD datapath connections as well as support global sparing scheme to minimize the required duplication. The customized architecture is implemented in Verilog and synthesized in IBM 90nm technology using Synopsys physical compiler to obtain power and performance numbers.

The rest of the chapter is organized as follows. Section 6.2 discusses variation issues of near-threshold operations at circuit- and architecture-levels. Section 6.3 explores techniques to tolerate and mitigate the variation-induced timing errors. Section 6.4 introduces a case study on variation-aware wide SIMD architectures. Section 6.5 discusses the related work and Section 6.6 concludes the chapter.

6.2 Variations in Near-threshold Operation

As described in Section 5.2, near-threshold designs significantly reduce energy consumption. However, I_{on} is highly sensitive to variations in V_{th}, resulting in delay variations which diminish the advantage of near-threshold operations. RDFs (Random Dopant Fluctuations) are known to be the dominant factor of I_{on} variations in near-threshold operations [61]. In addition, LER (Line Edge Roughness) is a significant factor for advanced technology nodes. To evaluate the effect of cross

chip variations in the near-threshold voltage regime, Monte Carlo simulations with Hspice are performed for 90nm/45nm commercially used GP (General Purpose) models and 32nm/22nm PTM (Predictive Technology Model [49]) HP (High Performance) models. Normal distributions for two dominant variation sources, V_{th} and LER, are inserted into the 32nm/22nm PTM HP models.

In this section, we examine how much delay variations occur in the near-threshold voltage region at two levels: (A) circuit-level variations and (B) architecture-level variations.

6.2.1 Circuit-level Variations

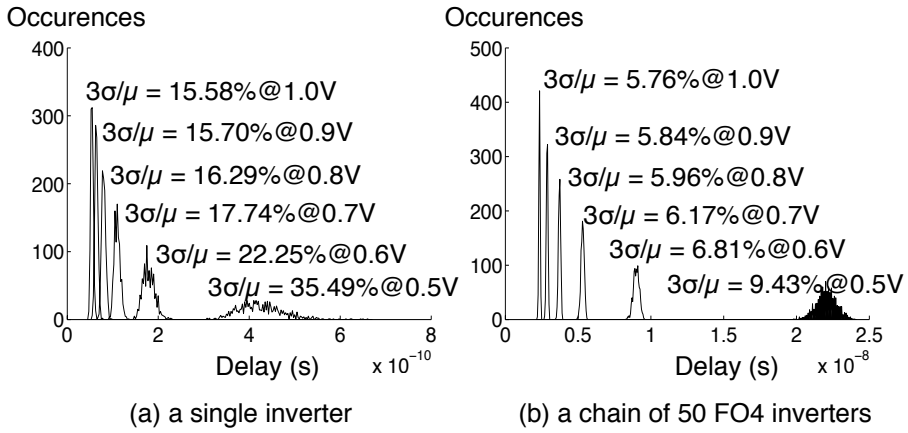

Figure 6.1: Delay distributions of (a) a single inverter and (b) a chain of 50 FO4 inverters with different supply voltages (0.5V, 0.6V, 0.7V, 0.8V, 0.9V and 1.0V) using 90nm GP technology. A thousand samples for each supply voltage are simulated.

Figure 6.1 shows that the delay distributions of a single inverter and a chain of 50 FO4 (Fan-out of 4) inverters using 90nm GP models. The delay variation ($3\sigma/\mu$) of a single inverter significantly increases as V_{dd} reduces; for example, $3\sigma/\mu$

is increased from 15.58%@1.0V to 35.49%@0.5V. Although the delay variations in near-threshold voltage region cause large performance degradation on a single gate, the uncorrelated random within-die variations average out over a long chain of gates as shown in Figure 6.1(b). The delay variation ($3\sigma/\mu$) of a chain of 50 FO4 inverters is only 9.43%@0.5V compared to that of a single inverter (35.49%@0.5V). Thus the delay variation is not significant for medium to long chains and is expected to not be significant for datapath components. A similar observation was made in [65] which showed only 8.4%@0.5V delay variation for a 64-bit Kogge-Stone adder. Therefore, part of the delay variation problem can be alleviated by implementing longer logic chains [61].

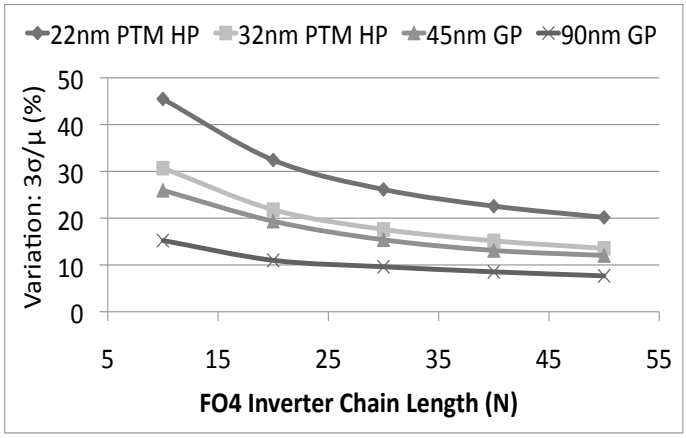

Figure 6.2: Delay variations ($3\sigma/\mu$) (%) at 0.55V of a chain of FO4 inverters vs. chain length (N) using four technology models (90nm GP, 45nm GP, 32nm PTM HP, and 22nm PTM HP). A thousand samples for each data point are simulated.

Figure 6.2 shows delay variation ($3\sigma/\mu$) at 0.55V as a function of a chain length (N) of FO4 inverters using four different technology models. Although delay variations reduce as N increases, the amount of reduction ($\frac{\Delta 3\sigma/\mu}{\Delta N}$) decreases with N.

Therefore, implementing the logic with a long chain of gates will not solve all the timing variation problems. In addition, technology scaling exacerbates the delay variations [62]. As can be seen in Figure 6.2, technology scaling from 90nm to 22nm increases delay variation ($3\sigma/\mu$) at 0.55V by ∼2.5x for a chain of 50 FO4 inverters.

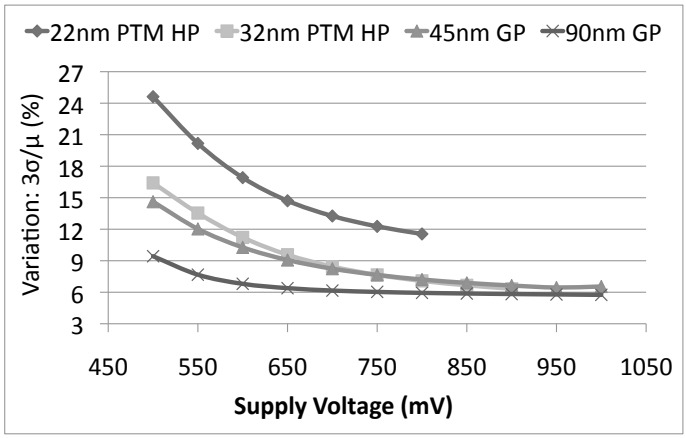

Figure 6.3: Delay variations ($3\sigma/\mu$) (%) of a chain of 50 FO4 inverters vs. supply voltage (V_{dd}) using four technology models (90nm GP, 45nm GP, 32nm PTM HP, and 22nm PTM HP). A thousand samples for each data point are simulated.

Figure 6.3 shows the delay variations of a chain of 50 FO4 inverters as a function of V_{dd}. The 22nm PTM HP and 32nm PTM HP models are simulated up to their nominal voltages—800mV and 900mV respectively. As V_{dd} decreases, the delay variations exponentially increases. This trend exacerbates with technology scaling; for example, the increase in delay variation ($3\sigma/\mu$) from 1V to 0.5V is only ∼4% in 90nm technology, which is very small compared to ∼14% increase in 22nm technology (from 11%@0.8V to 25%@0.5V). This is because LER causes relatively high variations on devices in advanced technology nodes [74]. However, strict design rules and new manufacturing processes such as the use of metal-gates with high-k material or

silicon-on-insulator (SOI) [73] can help limit the variability. In addition, advances in lithography like double patterning [75] and immersion [76] are likely to reduce the effect of LER as well.

6.2.2 Architecture-level Variations

To examine the variation effects in a wide SIMD architecture, the following two assumptions (A1 and A2) and two properties (P1 and P2) are used.

(A1) A critical path of a SIMD architecture is emulated with a chain of 50 FO4 inverters.

(A2) A hundred critical paths exist in one SIMD lane.

(P1) The delay of one SIMD lane (1-wide) is determined by the slowest critical path in the lane.

(P2) The delay of an N-wide SIMD datapath is determined by the slowest of the N SIMD lanes.

A chain of 50 FO4 inverters is used to emulate a critical path of a wide SIMD architecture because they are similar in terms of average delay and variation at all voltages, not just at near-threshold voltages. We chose a chain configuration because it is a standard practice in circuit-level analysis (A1). Although a generated synthesis report shows $\sim$50 critical paths in each SIMD lane, another 50 near-critical paths are also considered because they could become critical due to increased variations in the

near-threshold regime (A2). Properties, P1 and P2, are the standard simulation steps based on these assumptions.

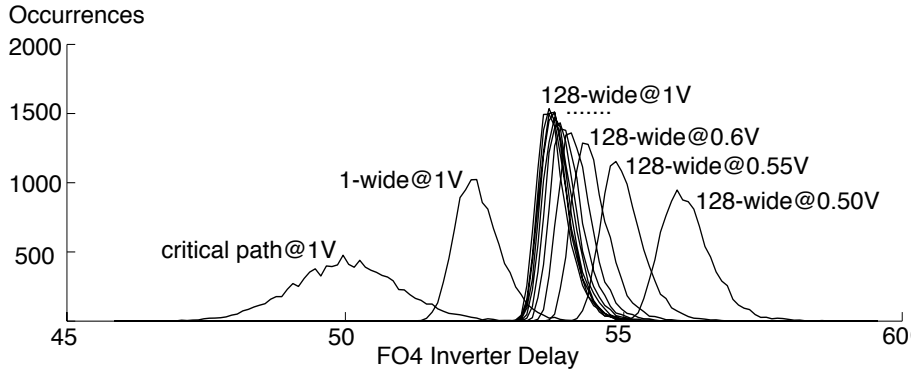

Figure 6.4: Delay distributions for a critical path (a chain of 50 FO4 inverters) at V_{dd}=1V, one SIMD lane at V_{dd}=1V, and 128-wide SIMD datapath at near-threshold supply voltages from 0.5V to 1V. 90nm GP model is used and a 10,000 samples are simulated.

Figure 6.4 shows the delay distributions for a critical path (a chain of 50 FO4 inverters), one SIMD lane (1-wide system) operating at 1V, and 128-wide systems operating at near-threshold supply voltages. The delay unit on the x-axis is FO4 inverter delay which is different from absolute delay (in ns) used in Figure 6.1. For example, the delay of a chain of 50 FO4 inverters operating at 0.5V is 22.05ns (= 50 FO4 delay@0.5V); on the other hand, that at 0.6V is 8.99ns (= 50 FO4 delay@0.6V). In this chapter, FO4 delay is used to measure variation effects in the near-threshold voltage region.

As can be seen in Figure 6.4, the delay distribution of a 1-wide SIMD datapath@1V is shifted to the right compared to that of one critical path@1V because the delay of a 1-wide system is determined by the maximum delay of a hundred critical paths. The same reasoning can be made to explain the shift in the delay distribution from

1-wide@1V to 128-wide@1V. The 128-wide SIMD datapath is slower than the 1-wide SIMD datapath because the possibility of having slow critical paths increases. Another characteristic is that the delay distributions of 128-wide systems operating at low supply voltages drift to the right. This shift is because the delay distribution of a critical path at near-threshold voltages has a wider spread than that at nominal voltage.

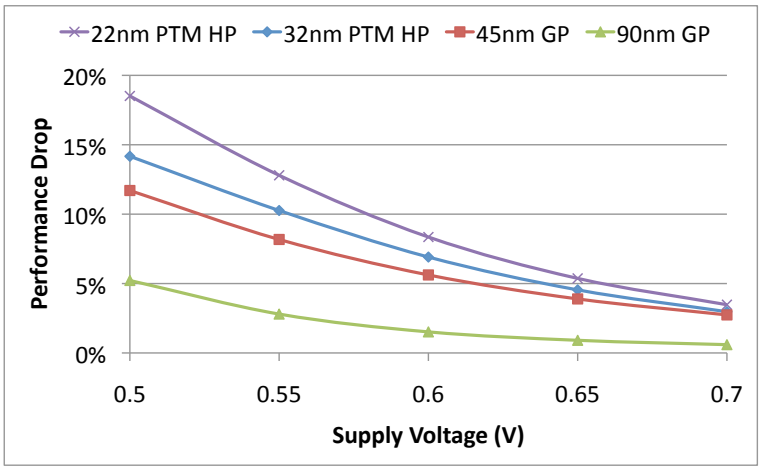

Figure 6.5: Performance drop (%) in the near-threshold voltage region for a 128-wide SIMD architecture. 90nm/45nm GP and 32nm/22nm PTM HP models are used.

In order to evaluate performance degradation due to near-threshold voltage operations, we compare the 99% point of FO4 chip delay ($fo4chipD$) distributions. The performance degradation of a 128-wide SIMD architecture operating at near-threshold voltage (NTV) region compared with the performance at nominal voltage (or full voltage, FV) is given by $\frac{fo4chipD@NTV - fo4chipD@FV}{fo4chipD@FV}$. Figure 6.5 shows the performance drop as a function of supply voltage in four technology nodes. As expected, the performance drop increases as the supply voltage decreases. For example, in

90nm GP model, the performance drop at 0.5V, 0.55V, and 0.6V is ∼5%, ∼2.5%, and ∼1.5% respectively compared to 1V operation. In addition, the increase in performance degradation of lower technology nodes is a lot higher. For example, the performance drop at 0.5V climbs to ∼18% in 22nm PTM HP model.

This analysis shows that delay variations in wide-SIMD architectures is not that large. It is only ∼5%@0.5V in 90nm GP and increases to ∼20% for 22nm PTM HP model. It is very likely that the variations will be lower in 22nm real silicon. Thus complex architectural enhancements are not needed to handle these delay variations. In fact, simple techniques are sufficient to handle./pact2011/figures/ the variation-induced delay variations in wide SIMD architectures, as will be described in the following Section.

6.3 Techniques to Control Effect of Variations

There are two mechanisms to tolerate variation-induced timing errors in a scalar pipeline: 1) flushing the pipeline and re-executing a instruction with relaxed timing or 2) waiting one more cycle for the pipeline to generate the correct output. However, applying these approaches to wide SIMD architectures is problematic because the power penalty of the flush-rollback process in the SIMD pipeline is much larger than that of a scalar pipeline. For example, an error encountered in one SIMD lane would cause the other SIMD lanes to stall, flush, and execute the same operations again. Recent work also shows that there is a significant performance drop in SIMD architectures as single-stage-error probabilities increase [63]. To prevent variation-induced

timing errors in near-threshold operation, we analyzed the effect of three techniques: 1) structural duplication, 2) voltage margining, and 3) frequency margining.

6.3.1 Structural Duplication

Structural duplication is a well-known technique for extending reliability. Redundant micro-architectural structures are added to the processor and designated as spares [72]. When some architectural modules fail in time, the spare structures replace the failed ones to extend lifetime reliability. This structural duplication idea can be used to handle slow SIMD lanes that fail to operate within a given clock period. If the faulty SIMD lanes can be identified at test time, the spare SIMD lanes can be used to replace them.

We studied a 128-wide SIMD architecture and analyzed how many SIMD functional unit duplications (α spares) are required to tolerate variation-induced timing errors while running in the near-threshold voltage regime. Monte Carlo simulations were performed to generate FO4 delay distribution curves for the duplicated systems as shown in Figure 6.6.

The delay distribution of a 128-wide machine@1V is used as the baseline and the delay distribution of 128-wide+α-spares system@0.55V is used to demonstrate the effect of SIMD functional unit duplications. For example, the distribution curve of *128-wide+6-spares*@0.55V is essentially the distribution of 128 *good* SIMD datapaths out of 134 (128+6) SIMD datapaths; i.e. six slowest SIMD datapaths are dropped to generate this delay distribution. As can be seen, extra SIMD datapaths help shift

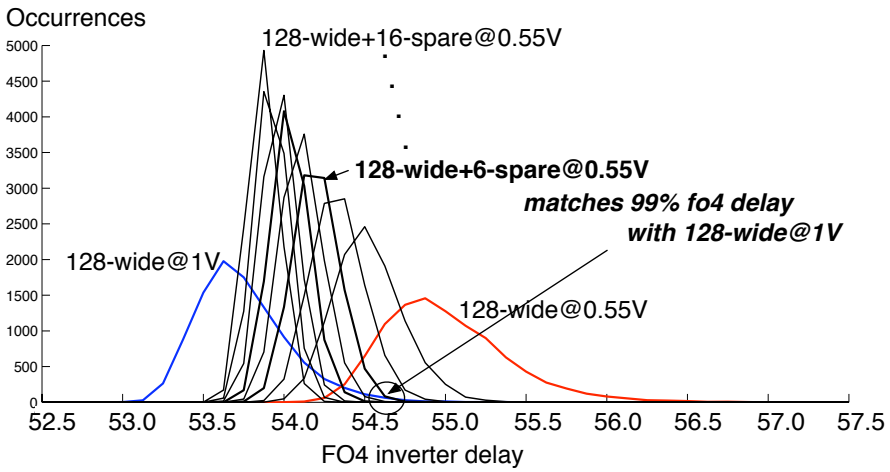

Figure 6.6: Delay distributions for SIMD duplicated systems (128-wide + α-spares) using 90nm GP model. A 10,000 samples for each curve are simulated.

delay distributions to the left and make the spread smaller.

	90nm			45nm			32nm			22nm		
Vdd	**spares**	**area ovhd.**	**power ovhd.**	**spares**	**area ovhd.**	**power ovhd.**	**spares**	**area ovhd.**	**power ovhd.**	**spares**	**area ovhd.**	**power ovhd.**
0.50V	28	12.1%	4.6%	>128	> 57.8%	> 25.0%	>128	> 57.8%	> 25.0%	>128	> 57.8%	> 25.0%
0.55V	6	2.6%	1.0%	84	37.2%	15.3%	>128	> 57.8%	> 25.0%	80	35.3%	14.5%
0.60V	2	0.9%	0.3%	26	11.2%	4.3%	48	20.9%	8.2%	22	9.5%	3.6%
0.65V	1	0.4%	0.2%	10	4.3%	1.6%	12	5.1%	1.9%	7	3.0%	1.1%
0.70V	1	0.4%	0.2%	4	1.7%	0.6%	6	2.6%	1.0%	3	1.3%	0.5%

Table 6.1: The required number of spares and corresponding area and power overhead of structural duplication scheme for four technology nodes. The area and power numbers are based on Diet SODA [60].

We match the 99% FO4 delay point of the duplicated systems operating at near-threshold voltages with that of the baseline architecture (128-wide) operating at nominal voltage to obtain the required number of additional SIMD spares. This experiment is repeated for four technology nodes (90nm, 45nm, 32nm, and 22nm), and the number of spares and corresponding area and power overhead at each supply voltage are presented in Table 6.1. We see that as supply voltage reduces, the number of SIMD spares exponentially increases to tolerate effect of delay variations. For example, in

90nm technology node, the number of spares increases from two spares for 0.6V to six spares for 0.55V and 28 spares for 0.5V. This is because, as shown in Figure 6.6, adding more spare units shifts the chip delay distribution to the left, but makes it tighter. For lower technology nodes, delay variations are larger and excessive number of spares is required to match the 99% FO4 delay point of the baseline architecture.

The additional SIMD functional unit (FU) spares are used to replace underperforming ones that are identified at test time. The faulty SIMD FUs can be power-gated because they are not used at run time. Therefore, the power overhead of the structural duplication scheme is limited only to enlarged routing, thus leading to minimal impact on power consumption. However, the increased SIMD width also requires a wider shuffle network operating at nominal voltage whose power consumption cannot be ignored. Thus, for low voltages (∼0.50V) where the variation-induced timing errors are severe, the structural duplication scheme has a large overhead.

6.3.2 Voltage Margining

	90nm		45nm		32nm		22nm	
Vdd	**Vdd margin**	**power ovhd.**	**Vdd margin**	**power ovhd.**	**Vdd margin**	**power ovhd.**	**Vdd margin**	**power ovhd.**
0.50V	5.8 mV	1.0%	19.6 mV	3.3%	12.1 mV	2.0%	16.4 mV	2.8%
0.55V	4.1 mV	0.6%	18.2 mV	2.8%	11.1 mV	1.7%	17.6 mV	2.7%
0.60V	2.9 mV	0.4%	16.2 mV	2.3%	10.4 mV	1.5%	11.1 mV	1.6%
0.65V	2.2 mV	0.3%	14.0 mV	1.8%	8.9 mV	1.1%	11.5 mV	1.5%
0.70V	1.7 mV	0.2%	12.8 mV	1.5%	7.7 mV	0.9%	9.6 mV	1.1%

Table 6.2: Required voltage margin to tolerate variation-induced timing errors for a 128-wide SIMD architecture operating at near-threshold voltages and corresponding power overhead for four technology nodes. The final supply voltage should be $V_{dd} + V_{dd}$ *margin* (V_M). The power overhead is based on Diet SODA [60].

As supply voltage (V_{dd}) decreases, the delay of a chain of 50 FO4 inverters expo-

nentially increases. Therefore, a small increase in supply voltage in the near-threshold voltage region can help compensate for variation-induced timing errors without increasing the clock period.

To gauge how much extra supply voltage is required, we first generated the FO4 chip delays ($fo4chipD$) and the corresponding absolute chip delays ($chipD$ in ns) of a 128-wide SIMD architecture operating at near-threshold voltages (NTVs). Then, the $chipD@NTV$ is scaled based on the ratio of $fo4chipD@FV$ and $fo4chipD@NTV$. The normalized $chipD@NTV$ is used as the baseline *target delay* for the architecture operating at near-threshold voltage to achieve the same level of variations at nominal voltage. Next, we increase supply voltage at a fine grain to find required voltage margin (V_M) that makes $chipD@NTV+V_M$ less than the *target delay*. Figure 6.7 illustrates how voltage margin is obtained for a 128-wide SIMD datapath operating at 600mV for a specific *target delay*. Delay distributions of a 128-wide SIMD architecture operating at 600mV, 605mV, 610mV, 615mV, and 620mV are generated. In addition, delay distributions of 128-wide+α-spare SIMD duplicated systems operating at 600mV are also shown in the figure. As can be seen, the $chipD$ (99% point of delay distribution) of a 128-wide SIMD architecture operating at $\sim$615mV is less than *target delay*. Therefore, $\sim$15mV is the voltage margin at design time that is required for a 128-wide SIMD architecture operating at 600mV to tolerate its delay variation.

Table 6.2 lists supply voltages (V_{dd}), voltage margins and the corresponding power overhead for four different technology nodes. Although a very small increase in sup-

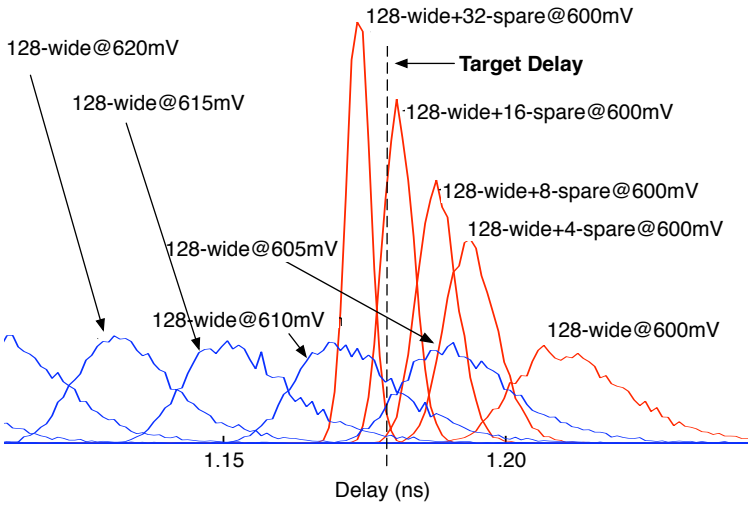

Figure 6.7: Delay distributions of 128-wide SIMD architecture operating at 600mV, 605mV, 610mV, 615mV and 620mV. For comparison, delay distributions of 128-wide+α-spare SIMD duplicated systems operating at 600mV are also presented. A 10,000 samples for each curve are simulated with 45nm GP model.

ply voltage is sufficient for the 90nm technology node, lower technology nodes require much larger supply voltage margins. For example, in 90nm technology, at V_{dd}=500mV, the supply voltage has to be increased to ∼506mV (500mV+5.78mV), but this jumps to ∼520mV in 45nm technology.

This extra supply voltage margin applies to all modules operating in the near-threshold voltage domain and thus incurs more power consumption than structural duplication methods for low variations. However, as variation increases, the voltage margining method offers a more power-efficient solution than the structural duplication scheme.

	90nm			45nm			32nm			22nm		
Vdd	**Tclk(ns)**	**Tva-clk(ns)**	**perf. drop**	**Tclk(ns)**	**Tva-clk(ns)**	**perf. drop**	**Tclk(ns)**	**Tva-clk(ns)**	**perf. drop**	**Tclk(ns)**	**Tva-clk(ns)**	**perf. drop**
0.50V	24.0	25.3	5.2%	1.9	2.1	11.7%	5.5	6.3	14.2%	3.1	3.7	18.5%
0.55V	14.3	14.7	2.8%	1.4	1.6	8.2%	2.9	3.2	10.3%	1.8	2.0	12.8%
0.60V	9.8	9.9	1.5%	1.2	1.2	5.6%	1.8	1.9	6.9%	1.3	1.4	8.4%
0.65V	7.3	7.4	0.9%	1.0	1.0	3.9%	1.3	1.3	4.5%	0.9	0.9	5.4%
0.70V	5.8	5.8	0.6%	0.9	0.9	2.7%	1.0	1.0	3.0%	0.7	0.7	3.5%

Table 6.3: Designed clock period (T_{clk}), variation-aware clock period (T_{va-clk}), and corresponding performance degradation at near-threshold voltages for four technology nodes. The power overhead is based on Diet SODA [60].

6.3.3 Frequency Margining

To avoid variation-induced timing errors, the clock period can be increased when there is very loose realtime constraint so that the increased clock period can still make the timing requirements. Table 6.3 presents desired clock period (T_{clk}), variation-aware clock period (T_{va-clk}), and corresponding performance degradation for several near-threshold voltages. As we move to advanced technology nodes, required delay margins reach almost 20%, which makes this frequency margining scheme an infeasible method to tolerate variation-induced timing errors.

In addition, the clock frequency of near-threshold SIMD datapath is closely related to that of a memory system; for example, the SIMD clock period ($T_{clk}@NTV$) has to be multiples of the memory clock period ($T_{clk}@FV$) to avoid complex synchronization between two sub-systems. Therefore, frequency margining has to be made with careful consideration of the underlying architecture.

Another method of handling variation-induced timing errors is to synthesize the design for a higher frequency. An issue with this technique is that it cannot be used when an architecture is already at its maximum synthesizable frequency.

6.3.4 Comparisons Between Variation-Tolerating Techniques

In this section, the power overhead of structural duplication and voltage margining is compared and summarized (see Figure 6.8). To achieve iso-throughput performance, frequency margining is not considered here.

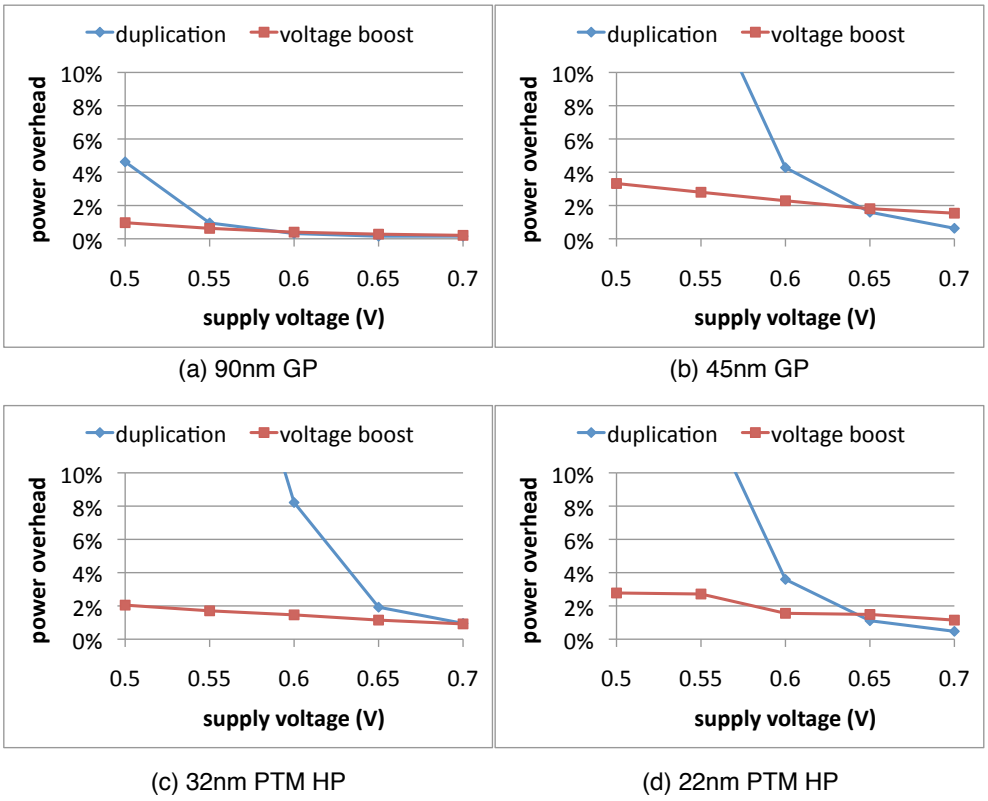

Figure 6.8: Power overhead comparison between structural duplication and voltage margining schemes for four technology nodes: (a) 90nm GP, (b) 45nm GP, (c) 32nm PTM HP, and (d) 22nm PTM HP

Structural duplication scheme outperforms voltage margining scheme in high near-threshold voltage regions (0.6V $\sim$ 0.7V) where variations are very low. However, as technology scales and supply voltage decreases, the voltage margining scheme starts to outperform the structural duplication scheme. This is because a slight increase in supply voltage exponentially reduces delay. Figure 6.8 serves as a guideline in which

variation-tolerating scheme must be selected for each supply voltage. For example, in 45nm technology node, when V_{dd}=0.6V, duplication method incurs ∼4% power overhead compared to ∼2% overhead of voltage margining scheme; therefore voltage margining is the preferred choice.

Although voltage margining offers a better solution than structural duplication for lower technology nodes as V_{dd} decreases, the structural duplication scheme still can significantly help manage variation-induced timing errors. Figure 6.9 shows chip delays for a 128-wide SIMD architecture operating at 600mV, 605mV, 610mV, 615mV, and 620mV using 45nm GP model. Target chip delay is calculated as described in Section 6.3.2. Based on this figure, the target chip delay can be achieved by having two additional SIMD lanes with 10mV voltage margin or eight additional SIMD lanes with 5mV voltage margin. Table 6.4 summarizes several design choices and the corresponding power overhead. As can be seen, a combination of two additional SIMD lanes and 10mV voltage margin achieves minimal power overhead (1.72%) compared to only structural duplication (4.28%) or only voltage margining (2.39%). Therefore, a combination of voltage margining and structural duplication can effectively tolerate and mitigate timing variation problems for lower technology nodes.

duplications	voltage margin	power overhead
26	0 mV	4.3 %
8	5 mV	2.0 %
2	10 mV	1.7 %
1	15 mV	2.3 %
0	17 mV	2.4 %

Table 6.4: Design choices for a 128-wide@600mV system in 45nm technology node. Combinations of structural duplication and voltage margining are presented with corresponding power overhead./pact2011/figures/.

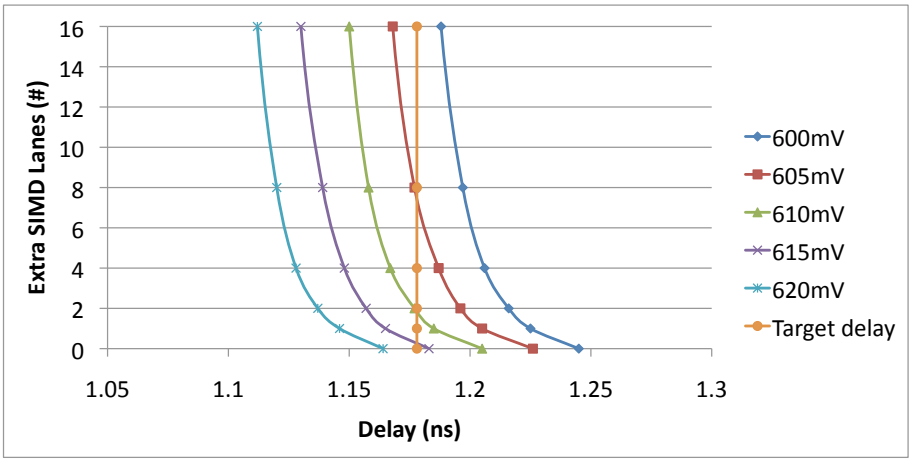

Figure 6.9: Chip delays for a 128-wide SIMD datapath operating at from 600mV to 620mV. Target delay is a design constraint for the 128-wide near-threshold system operating at 600mV. 45nm GP model is used.

6.4 Variation-Aware SIMD Architecture

In this section, we propose a variation-aware version of Diet SODA [60] to tolerate and mitigate variation-induced timing errors. The architecture is implemented in 90nm technology and the near-threshold SIMD datapath operates at 0.60V. Based on our analysis (Section 6.3.4), we exploited structural duplication scheme (two SIMD functional unit spares) for better power efficiency.

6.4.1 PE Design

Figure 6.10 shows the architectural details of a single processing element (PE) of a variation-aware wide SIMD architecture. The PE consists of 1) 64 KB multi-banked SIMD memory, 2) 4 KB scalar memory, 3) SIMD data prefetcher, 4) SIMD pipeline for vector operations, 5) scalar pipelines for sequential operations, and 6) 4-wide address generation unit (AGU) pipeline for providing local memory addresses for four

memory banks. The PE operates in two different voltage domains: full voltage and near-threshold voltage. Memory-related modules (1, 2, 3, 5a, and 6 in Figure 6.10) operate at full voltage because of data retention issues in the near-threshold voltage regime while SIMD datapath (4 and 5b in Figure 6.10) operates at near-threshold voltage to lower power consumption.

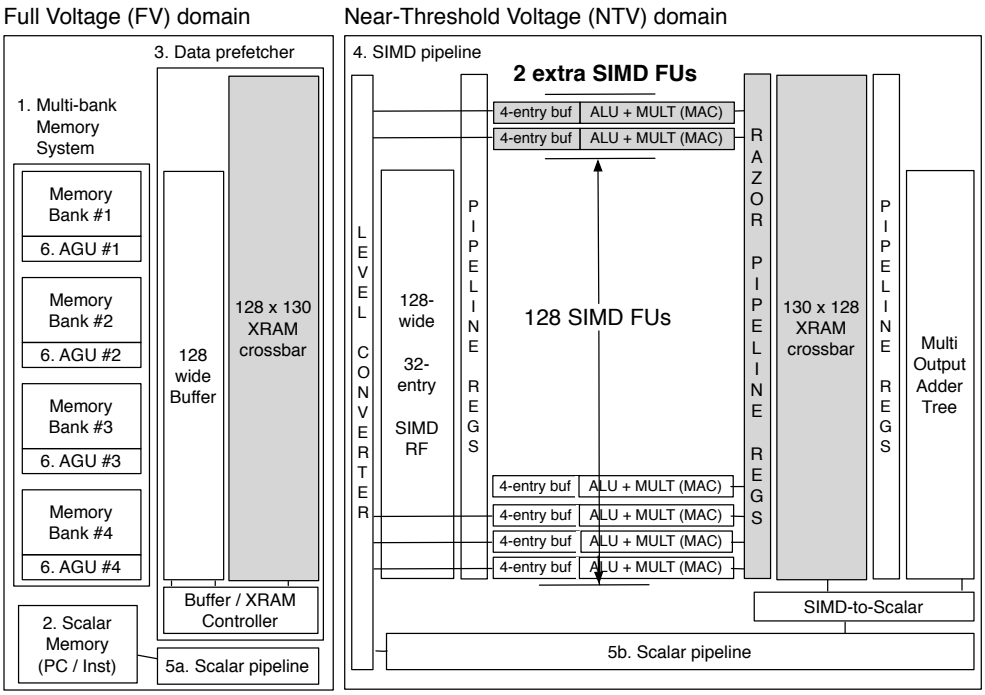

Figure 6.10: Processing element (PE) of a variation-ware SIMD architecture. The PE contains two different voltage domains: full voltage (FV) and near-threshold voltage (NTV). The PE consists of 1) multi-banked SIMD memory; 2) scalar memory; 3) data prefetcher, 4) SIMD pipeline, 5a) scalar pipeline in FV domain, 5b) scalar pipeline in NTV domain, and 6) four address generation unit (AGU) pipelines. The modified and inserted modules have been shown using shaded blocks.

The multi-banked SIMD memory system consists of four memory banks; each bank is 32-wide 16-bit 256-entries (16KB). The SIMD data prefetcher coordinates with 128-wide buffer and 128 x 130 XRAM crossbar to support complex alignment operations such as two-dimensional data access that are widely used in multimedia

algorithms. The four AGU pipelines are dedicated to the four SIMD memory banks and SIMD data prefetcher to handle memory address calculations. The SIMD pipeline consists of a 128-wide 16-bit 32-entry SIMD register file (RF), 130 functional units (FUs), a 130 x 128 XRAM crossbar (SIMD shuffle network (SSN)), and a multi-output adder tree. There are two scalar pipelines, one in each voltage domain; both pipelines consist of one 16-bit datapath and are used to perform sequential algorithms in addition to coordinating the SIMD datapath.

A key process in tolerating variation-induced timing errors is to identify which SIMD lanes fail in a given clock period. Razor techniques [67] are well known for error detection in scalar pipelines. Although Razor techniques can also help eliminate dynamic timing errors, this is not considered here because a failure in a single lane requires entire SIMD datapath to rollback and re-execute. In this context, a more appropriate use of Razor is to replace the standard pipeline registers next to SIMD functional units with Razor flip-flops that detect errors at test time and function as standard pipeline registers at run time.

The SIMD functional units (4-entry buffers and ALU/Multipliers) in the SIMD pipeline are the only units that are considered for handling the increased delay variations in near-threshold operations. This is because the critical paths of the system are in the SIMD functional units. The SIMD RF is not considered here because it is switched off during near-threshold operations [60]. Although the XRAM crossbar is vulnerable to variations in the near-threshold voltage regime as conventional SRAMs are, it is not affected by increased variations because this execution unit already has

∼15% time slack [66]. Therefore, in this work, we focus only on SIMD functional units because they were much more sensitive to timing variations.

Based on the analysis in Table 6.1, two additional SIMD functional units are inserted as spares. In addition, two XRAM crossbars are also expanded from 128x128 to 128x130 and 130x128 to support the spares. Although the number of additional SIMD spares has been determined, how to place the spares is another interesting design choice in wide SIMD architectures. This is because the placement method significantly affects the effectiveness of managing variation-induced delays, as will be elaborated in the following section.

6.4.2 Placement Method: Global vs. Local

We investigate two placement methods: global sparing and local sparing. The local sparing scheme groups SIMD functional units into clusters and places a spare for each cluster. The global sparing scheme places all the spares together.

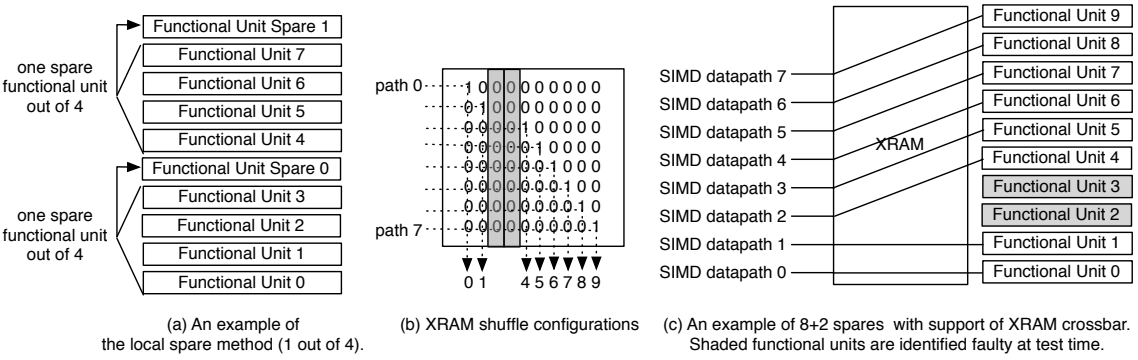

Figure 6.11: (a) Local sparing method. An example of *1 out of 4*. (b) XRAM shuffle configuration to bypass faulty SIMD lanes. (c) Global sparing method. An Example of 10 functional units (8 + 2 spares) with support of XRAM crossbar. Shaded SIMD functional units are identified as faulty ones at test time.

Recently proposed Synctium [63] suggests a local sparing method such as assigning one spare per every group of four SIMD lanes. Although the local redundancy overcomes complex re-routing problems, this local sparing method does not work when there are more than one faulty SIMD lanes in a cluster. Figure 6.11(a) shows how local functional unit (FU) spares work. Here, functional unit spare (FU-S-0) is used as a spare for a cluster consisting of FU-0, FU-1, FU-2, and FU-3. If multiple timing errors occur in this cluster, FU-S-0 cannot replace all the failing FUs. In that case, either the entire system must slow down or waste energy by increasing the voltage to meet timing constraints. On the other hand, a global sparing method is capable of dealing with bursty FU failures because spares are not assigned to specific clusters.

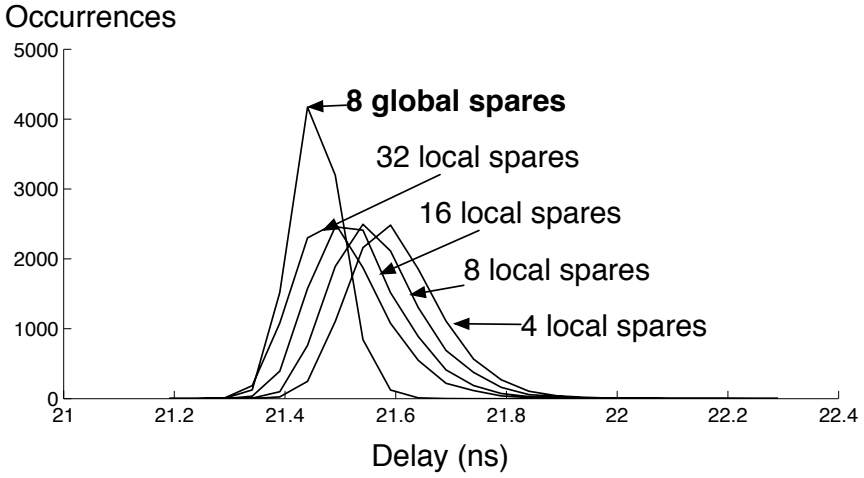

Figure 6.12: Delay distribution of local sparing and global sparing schemes. One global sparing scheme (8 spares) and four local sparing schemes (4, 8, 16 and 32 spares) are considered. A 128-wide SIMD datapath is used for the simulation.

Figure 6.12 compares the effectiveness of local sparing and global sparing schemes. Five schemes are considered for the 128-wide SIMD datapath — 8 global spares, 4 local spares (1 out of 32), 8 local spares (1 out of 16), 16 local spares (1 out of 8)

and 32 local spares (1 out of 4). We see that the global replication scheme with only 8 spares has a lower average delay and smaller spread compared to all the local replication schemes. Thus, the global sparing scheme has a lower area and power cost compared to the local sparing schemes.

Although global sparing effectively solves timing variability issues, it requires complex re-routing. Satpathy et al. recently proposed an area- and power-efficient XRAM crossbar [66], which exploits the circuit topology of SRAM cells and stores shuffle configurations at crossing points of the cells to improve performance while reducing area, power and routing congestions. We make use of the XRAM crossbar to effectively support bypassing underperforming SIMD lanes. Figure 6.11(c) shows how an XRAM crossbar bypasses faulty SIMD FU-2 and FU-3, and fully utilizes the remaining eight SIMD functional units based on the configuration registers stored in the XRAM crossbar shown in Figure 6.11(b).

Two 128 x 128 XRAM crossbars already exist in Diet SODA; one in FV domain and the other one in NTV domain. To support two additional SIMD lanes, the sizes of the XRAM crossbars are increased to 128x130 (in FV domain) or 130x128 (in DV domain). The corresponding increase in area and power is not substantial. Furthermore, XRAM crossbar scales well compared to other SIMD shuffle networks due to inherent circuit topology and smaller control overhead. Thus a combination of global spares and XRAM crossbars can effectively tolerate the timing variability problems of wide SIMD architectures.

6.4.3 Results and Analysis

6.4.3.1 Methodology

The variation-aware SIMD architecture is implemented as an RTL Verilog model and synthesized in IBM's 90nm technology. To investigate how much voltage/frequency scaling in near-threshold operations can be achieved, different types of representative test circuit (SRAM/RF/ALU) were laid out, and parasitic extraction was performed and annotated. Then Hspice simulations were done to determine the voltage, frequency, and power characteristics at different supply voltages for each module. To obtain power numbers, the PE logic of Diet SODA [60] was synthesized with Synopsys Physical Compiler and scaled to match the representative test circuit.

6.4.3.2 Area and Power

The area and power breakdown of the variation-aware SIMD architecture are presented in Table 6.5. The preview mode of full-HD images at 30 fps is considered for the analysis.

The increased area overhead in two additional SIMD functional units, enlarged XRAM crossbar, and Razor flip-flops are minimal compared to overall system. Underperforming SIMD functional units are power-gated at run time. In addition, Razor is used only at test time to identify the faulty SIMD functional units; at run time, inserted delay buffers to prevent hold/setup violations and shadow latches in Razor are power-gated, and only main flip-flops function as standard pipeline registers. Therefore, the increased process variation issues in near-threshold operations can be

				NTV mode	
	Components	**Area (mm^2)**	**Area (%)**	**Power (mW)**	**Power (%)**
PE	SIMD banked-memory (64KB)	3.41	33%	27.6	22%
	SIMD Register Files (4KB)	1.58	15%	0.0	0%
	SIMD Buffer (1KB)	0.41	4%	2.7	2%
	SIMD ALU/Multiplier, SSN	2.34	23%	24.1	20%
	SIMD Adder Tree	0.12	1%	1.0	1%
	SIMD pipeline+Clock+Routing	0.68	7%	11.8	10%
	Data Prefetcher	1.63	16%	33.7	27%
	Scalar/AGU Pipeline & Misc.	0.18	2%	21.9	18%
	Razor Flip-Flops & Delay Buffers	0.05	0%	0.1	0%
Total	Variation-Aware SIMD Architecture	10.40	100%	122.9	100%
	Diet SODA	10.27		122.5	
	Overhead over Diet SODA	0.13	1.3%	0.4	0.3%

Table 6.5: Area and power summary of the variation-aware SIMD architecture running preview mode of full-HD images at 30 fps using near-threshold operation. The area and power numbers of Diet SODA are also provided for comparison.

alleviated by simple SIMD functional unit duplications, marginally enlarged XRAM crossbar, and Razor flip-flops without sacrificing much area and power.

6.4.3.3 Performance

The loss in performance of Diet SODA due to increased variations in near-threshold computing is $\sim$2% for 90nm technology as presented in Figure 6.5. As shown in Table 6.5, a combination of SIMD functional unit duplication and enlarged XRAM crossbar eliminates the variation-induced timing errors with little power overhead (0.3%) without adjusting clock frequency. The deadline timing constraints are met and the overall application performance remains unaffected. Although it is possible to exploit extra structural duplication to improve the overall performance, this is not in this chapter's scope.

6.5 Related Work

There has been a large interest in subthreshold designs, resulting in a wide range of working processors for ultra low power applications. Examples include Subliminal [64], Phoenix processors [71], and the 180mV FFT processor [58]. However, to reduce the high energy efficiency marginally and improve processing throughput significantly, near-threshold operations are proposed. In addition, near-threshold operation also combines with parallel computing platforms in a synergistic manner. Zhai et al. show that exploiting near-threshold techniques achieves substantial energy savings in chip multi-processing [57] and Kaul et al. presents 494 GOPS/W SIMD vector processing accelerators operating at 300mV [69].

Although these subthreshold and near-threshold techniques offer great energy efficiency, variability has become a serious concern for operating at extremely low voltages. Variation-aware architectures are implemented using circuit techniques such as clock/power gating and dynamic voltage-frequency scaling [70], and fine-grained power management using both dual-supply voltage and power gating [69]. EVAL [68] provides a framework to show how several techniques such as ABB (Adaptive Body Biasing) / ASV (Adaptive Supply Voltage), FU (Functional Unit) replication, and issue-queue resizing can trade off variation-induced errors for power and performance. However, little analysis has been performed to investigate the impact of process variability on large parallel architectures such as a SIMD machine. Recently, Synctium [63] studied the variation issues in near-threshold SIMD architectures and proposed decoupled parallel SIMD pipelines and pipeline weaving using decoupling queues and

micro-barriers to tolerate variation-induced timing errors. Our work differs in that we first provide a detailed analysis of variation impact on wide SIMD architectures for different technology nodes, and show past studies have over-estimated the effect of delay variations in near-threshold operations. We also propose simple techniques to handle delay variations in multiple technology nodes and present a variation-aware wide SIMD architecture that effectively tolerate the timing variability problems in 90nm technology by exploiting simple SIMD functional unit duplications connected via an XRAM crossbar.

6.6 Summary

Near-threshold operation enables a more energy-efficient architecture. In particular, a combination of near-threshold circuit techniques and parallel SIMD computations has the capability of providing high energy efficiency with high-throughput performance. Although near-threshold techniques offer new promising architectural design options, they suffer from large delay variations due to increased process variability. In this work we provide a systematic study of variation issues of near-threshold wide SIMD architectures and show that the variation-induced timing errors in wide SIMD architectures are fairly small, and can be allayed with combinations of three simple techniques: structural duplication, voltage margining and frequency margining. Through a case study based on Diet SODA in 90nm technology node, we show that the variation-induced timing errors in wide SIMD architectures can be handled by increasing the number of SIMD functional units from 128 to 130 and exploiting

XRAM crossbars to build a new error-free datapath. However, for lower technology nodes, use of only structural duplication is not as efficient; rather a combination of structural duplication and voltage margining results in a solution with the lowest power overhead.

CHAPTER 7

Conclusion

Wireless mobile communication has become one of the central uses of computing technology. An increasing number of wireless protocols have emerged that the cost of supporting multiple protocols using hardwired ASIC solutions more expensive and complex. In addition, today's devices not only support advanced signal processing of wireless communication protocols, but also media processing such as video encoding/decoding and interactive video conferencing. The advanced functionalities for next generation mobile computing require higher data rates, more sophisticated algorithms, and greater computational diversity with stringent power requirements. This dissertation explores the architectural impacts of emerging wireless protocols and advanced signal processing on a SIMD-based architecture for SDR, SODA, to improve efficiency. The efficient mobile computing in Diet SODA exploits massively parallel systems and near-threshold voltage operations to provide efficiency and programmability as well.

7.1 Summary

This dissertation presents a set of design proposals for an energy-efficient programmable wireless protocol implementation. In order to satisfy demanding performance and power requirements of next generation mobile computing, this dissertation takes a hardware-software co-design approach that optimizes and evaluates a mobile computing platform based on the characteristics of wireless signal processing algorithms. This dissertation makes the following contributions.

Design and Analysis of advanced signal processing algorithms This dissertation presents algorithmic characterization of two major mobile signal processing algorithms: a representative 4G protocol algorithm (Low Density Parity Check (LDPC)) and high definition mobile video (H.264). Based on insights from their characteristics, a wide-SIMD architecture for SDR, SODA, is revisited and optimized to meet performance and power requirements. The key enhancements on SODA are 1) use of programmable crossbar to support complex shuffle operations, 2) SIMD partitioning to support fine-grain SIMD computation, 3) Bypass and temporary buffer to support efficient access for short-lived intermediate data, and 4) fused operation to support accelerating frequently used instruction pairs.

Design, implementation, and evaluation of an energy efficient signal processing architecture, Diet SODA This dissertation presents an energy efficient signal processing architecture, Diet SODA. The key design idea is to apply near-threshold operation on a wide-SIMD architecture to achieve both high energy efficiency and high throughput performance in a synergistic manner. A combination

of near-threshold circuit techniques and parallel SIMD computations offer several new promising architectural design options: 1) very wide SIMD datapath to compensate for degraded throughput performance induced by near-threshold operations, 2) scatter-gather data prefetcher to exploit the large latency gap between memory operating at full voltage and the SIMD datapath operating at near-threshold voltage, and 3) dual operating mode to support both less stringent realtime-constrained tasks and high-throughput demanding tasks.

In-depth study of variations in near-threshold operations This dissertation presents a systematic study of delay variations induced by near-threshold operations at both circuit- and architecture-levels. The variation-induced timing errors in wide SIMD architectures are shown to be fairly small; therefore three simple techniques—1) structural duplication, 2) voltage margining and 3) frequency margining—are explored to tolerate and mitigate the timing variability problems. Through a case study based on Diet SODA in 90nm technology node, the variation-induced timing errors in wide SIMD architectures can be handled by the structural duplication scheme by increasing the number of SIMD functional units to replace underperforming ones and exploiting XRAM crossbars to build a new error-free datapath. However, for lower technology nodes, use of only structural duplication is not as efficient; rather a combination of structural duplication and voltage margining leads to a solution with the lowest power overhead.

7.2 Future Work

There are many possible extensions and applications for the concepts and architectures presented in this dissertation.

Additional Applications. Mapping more applications to Diet SODA, particularly 4G wireless protocols and biometric applications, would provide additional insights into the performance and efficiency of the mechanisms and compositions in Diet SODA. Many applications that we have used have regular data-level and task-level parallelisms that map naturally to the architecture. Therefore, it would be interesting to explore how more control-dependent and less parallelizable applications map to or tweak the architecture.

Other Variations Effects. We explored the effects of V_{th} and LER variations in near-threshold voltage operations on wide-SIMD architectures. There are also other factors such as voltage droop (IR drop) and temperature variations that significantly impact timing errors. On-chip sensors can be used to detect the changes in voltage and temperature. With this information, voltage and/or frequency tuning can be exploited, which may achieve near-optimal energy efficiency on wide-SIMD architectures. Reliable SRAM and XRAM crossbar are interesting research topic because they are more vulnerable to process variations.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Y. Lin et al., SODA: A Low-power Architecture For Software Radio, *33rd International Symposium on Computer Architecture (ISCA '06)*, pages 89–101, June 2006.

[2] M. Woh et al., From SODA to scotch: The evolution of a wireless baseband processor, *41st IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*, pages 152–163, Nov. 2008.

[3] N. Clark et al., OptimoDE: Programmable accelerator engines through retargetable customization, *Hot Chips 16*, Aug. 2004.

[4] S. Knowles, Ths SoC Future is Soft, *IEE Cambridge Processor Seminar [online]: http://www.iet-cambridge.org.uk/arc/seminar05/slides/SimonKnowles.pdf*, Dec. 2005.

[5] A. Nilsson and D. Liu, Area Efficient Fully Programmable Baseband Processors, *Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 333–342, July 2007.

[6] J. Glossner et al., The Sandbridge Sandblaster Communications Processor, *3rd Workshop on Application Specific Processors*, pages 53–58, Sep. 2004.

[7] J. Fridman and Z. Greenfield, The TigerSharc DSP architecture, *IEEE Micro*, volume 20 (1), pages 66–76, Jan. 2000.

[8] K. van Berkel et al., Vector Processing as an Enabler for Software-Defined Radio in Handsets From 3G+WLAN Onwards, *EURASIP Journal on Applied Signal Processing*, pages 2613–2625, Jan. 2005.

[9] H. Bluethgen et al., A Programmable Baseband Platform for Software Defined Radio, *Proceedings of the 2004 SDR Technical Conference and Product Exposition*, Nov. 2004.

[10] 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*, pages 61–70, Jan. 2003.

[11] G. K. Rauwerda, Towards Software Defined Radios Using Coarse-Grained Reconfigurable Hardware, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 16, No. 1, pages 3–13, Jan. 2008.

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

[13] A. Lodi et al., XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module, *IEEE Journal of Solid-State Circuits*, volume 41, No.1, pages 85–96, Jan. 2006.

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

[15] M. Woh et al., The next generation challenge for software defined radio, *Proceedings of the 7th international conference on embedded computer systems: architectures, modeling, and simulation* pages 343–354, 2007.

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

[17] ARM Advanced SIMD Extension (NEON Technology), *[online]: http://www.arm.com/products/CPUs/NEON.html*.

[18] M. Woh et.al, AnySP: Anytime Anywhere Anyway Signal Processing, *Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.

[19] T. Miyamori and K. Olukotun, REMARC: Recongurable multimedia array coprocessor, *IEICE Trans. Inf. Syst.,* volume E82-D, No. 2, pages 389-397, 1999.

[20] R. Gallager, Low-density parity-check codes, *IRE Transactions on Information Theory,* volume IT-8, No.1, pages 21–28, Jan. 1962.

[21] D. J. C. MacKay and R. M. Neal, Near shannon-limit performance of low-density parity-check codes, *Electronics letters,* volume 32, pages 1645–1646, Aug. 1996.

[22] M. M. Mansour and N. R. Shanbhag, High-throughput ldpc decoders, *IEEE Transactions on VLSI Systems,* volume 11, No.6, pages 976–996, Dec. 2003.

[23] F. Guilloud et al., λ-min decoding algorithm of regular and irregular ldpc codes, *3rd International Symposium on Turbo Codes & related topics,* Sep. 2003.

[24] D. E. Hocevar, A reduced complexity decoder architecture via layered decoding of ldpc codes, *IEEE Workshop on Signal Processing Systems,* pages 107–112, 2004.

[25] IEEE Std 802.16e-2005, *[online]: http://standards.ieee.org/getieee802/download/802.16e-2005.pdf,* Feb. 2006.

[26] I. Richardson, H.264 and MPEG-4 video compression, *WILEY*, 2003.

[27] N. Goel et al., Power reduction in VLIW processor with compiler driven bypass network, *Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems*, pages 233–238, Jan. 2007.

[28] K. Fan et al., Systematic register bypass customization for application-specific processors, *Proceedings of IEEE 14th International Conference on Application-Specific Systems, Architectures, and Processors*, pages 64–74, June 2003.

[29] D. Talla et al., Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements, *IEEE Transactions on Computers*, volume 52, No. 8, pages 1015–1031, Aug. 2003.

[30] R. Wang et al., Motion compensation memory access optimization strategies for H.264/AVC decoder, *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages v97–v100, Mar. 2005.

[31] R. Wang et.al, High throughput and low memory access sub-pixel interpolation architecture for H.264/AVC HDTV decoder, *IEEE Transactions on Consumer Electronics*, volume 51, No. 3, pages 1006–1013, Aug. 2005.

[32] S. -Z. Wang et.al, A new motion compensation design for H.264/AVC decoder, *Proceedgins of IEEE International Symposium on Circuits and Systems*, volume 5, pages 4558–4561, May 2005.

[33] C. -M. Chen and C. -H. Chen, Configurable VLSI architecture for deblocking filter in H.264/AVC, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 16, No. 8, pages 1072–1082, Aug. 2008.

[34] P. Dang, High performance architecture of an application specific processor for the H.264 deblocking filter, *IEEE Transactions on Very Large Scale Integration Systems*, volume 16, No. 10, pages 1321–1334, Oct. 2008.

[35] E. B. Van Der Tol et al., Mapping of H.264 decoding on a multiprocessor architecture, *Proceedings of SPIE Conference on Image and Video Communications and Processing*, volume 5022, pages 707–718, Jan. 2003.

[36] E. Q. Li and Y. -K. Chen, Implementation of H.264 encoder on general-purpose processors with hyper-threading technology, *Proceedings of SPIE Conference on Visual Communications and Image Processing*, volume 5308, pages 384–395, Jan. 2004.

[37] T. -C. Chen et.al, Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder, *IEEE Transactions on Circuits and Systems for Video Technology*, volume 16, No. 6, pages 673–688, June 2006.

[38] T.-C. Chen et al., Hardware architecture design of an H.264/AVC video codec, *Proceedings of Asia and South Pacific Conference on Design Automation*, Jan. 2006.

[39] S. D. Kim et.al, ASIP approach for implementation of H.264/AVC, *Journal of Signal Processing Systems*, volume 50, No. 1, pages 53–67, Jan. 2008.

[40] T. C. Chen et.al, 2.8 to 62.7 mW low-power and power-aware H.264 encoder for mobile applications, *IEEE Symposium on VLSI Circuits*, pages 222–223, June 2007.

[41] M. Bhatnagar, TMS320DM6446/3 Power Consumption Summary, *Texas Instruments Application Reports, [online]: http://focus.ti.com/lit/an/spraad6a/spraad6a.pdf*, Feb. 2008.

[42] J. C. Chen and S. -Y. Chien, CRISP: Coarse-Grained Reconfigurable Image Stream Processor for Digital Still Cameras and Camcorders, *IEEE Transactions on Circuits and Systems for Video Technology,* volume 18, No. 9, pages 1223–1236, Sep. 2008.

[43] K. Illgner et al., Programmable DSP platform for digital still cameras, *Proceedings of IEEE International Conference on Acoustics, Speech, Signal Processing,* volume 4, pages 2235–2238, Mar. 1999.

[44] C. Chute, Worldwide Digital Still Camera 20092013 Forecast, *IDC,* Apr. 2009.

[45] H. Zen et al., A new digital signal processor for progressive scan CCD, *IEEE Transactions on Consumer Electronics,* volume 4, No. 2, pages 289–296, May 1998.

[46] N.Nakano et al., Digital still camera system for megapixel CCD, *IEEE Transactions on Consumer Electronics,* volume 44, No. 2, pages 289–296, May 1998.

[47] D. Talla et al., Anatomy of a portable digital mediaprocessor, *IEEE Micro,* volume 24, No. 2, pages 32–39, 2004.

[48] S. Agarwala et al., A 600MHz VLIW DSP, *IEEE International Solid-State Circuits Conference, Digest of Technical Papers,* volume 2, pages 38–39, 2002.

[49] Nanoscale Integration and Modeling (NIMO) Group, Predictive technology model (PTM), *[online]: http://www.eas.asu.edu/˜ptm/*

[50] B. Khailany et al., Imagine: media processing with streams, *IEEE Micro,* volume 21, No. 2, pages 35–46, 2001.

[51] B. K. Khailany et al., A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing, *IEEE Journal of Solid-State Circuits,* volume 43, No. 1, pages 202–213, Jan. 2008.

[52] O. Vermeulen et al., Ultra Fast Grey Scale Face Detection Using Vector SIMD Programming, *Third International IEEE Conference on Signal-Image Technologies and Internet-Based System,* pages 585–592, Dec. 2007.

[53] C. Wu et al., Mapping Vision Algorithms on SIMD Architecture Smart Cameras, *First ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC '07,* pages 27–34, Sep. 2007.

[54] R. Dreslinski et al., Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits, *Proceedings of the IEEE,* volume 98, No. 2, pages 253–266, Feb. 2010.

[55] B. Bayer, Color Imaging Arrary, *U.S.Patent 3 971 065,* July 1976.

[56] D. Philips, Image Processing in C, *R&D Publications Inc.*, 1994.

[57] B. Zhai et al., Energy efficient near-threshold chip multi-processing, *Proceedings of the 2007 International Symposium on Low-Power Electronics Design,* pages 32–37, 2007.

[58] A. Wang and A. Chandrakasan, A 180mV FFT processor using subthreshold circuit techniques, *IEEE International Solid-State Circuits Conference, Digest of Technical Papers,* pages 292–529, 2004.

[59] G. D. Vita and G. Iannaccone, Ultra-low-power series voltage regulator for passive RFID transponders with subthreshold logic, *Electronics letters*, volume 42, No. 23, pages 1350–1351, 2006.

[60] S. Seo et al., Diet SODA: A Power-Efficient Processor for Digital Cameras, *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 79-84, 2010.

[61] B. Zhai et al., Analysis and mitigation of variability in subthreshold design, *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pages 20–25, 2005.

[62] K. Bernstein et al., High-performance cmos variability in the 65-nm regime and beyond, *IBM Journal of Research and Development*, volume 50, No. 4.5, pages 433–449, 2006.

[63] E. Krimer et al., Synctium: a near-threshold stream processor for energy-constrained parallel applications, *IEEE Computer Architecture Letters*, pages 21–24, 2010.

[64] B. Zhai et al., Energy-efficient subthreshold processor design, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 17, No. 8, pages 1127–1137, Aug. 2009.

[65] N. Drego et al., All-digital circuits for measurement of spatial variation in digital circuits, *IEEE Journal of Solid-State Circuits*, volume 45, No. 3, pages 640–651, Mar. 2010.

[66] S. Satpathy et al., A 1.07 Tbit/s 128x128 Swizzle Network for SIMD Processors, *IEEE Symposium on VLSI Circuits*, June 2010.

[67] D. Ernst et al., Razor: A low-power pipeline based on circuit-level timing speculation, In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, pages 7–18, 2003.

[68] S. Sarangi et al., Eval: Utilizing processors with variation-induced timing errors, In *Proceedings of the 41st Annual International Symposium on Microarchitecture*, pages 423–434, Dec. 2008.

[69] H. Kaul et al., A 300mv 494gops/w reconfigurable dual-supply 4-way SIMD vector processing accelerator in 45nm CMOS," *IEEE Journal of Solid-State Circuits*, volume 45, No. 1, pages 95–102, 2010.

[70] S. Dighe et al., Within-die variation-aware dynamic-voltage-frequency scaling core mapping and thread hopping for an 80-core processor, *2010 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 174–175, Feb. 2010.

[71] M. Seok et al., The Phoenix processor: A 30pw platform for sensor applications, *IEEE Symposium on VLSI Circuits*, pages 188–189, June 2008.

[72] J. Srinivasan et al., Exploiting structural duplication for lifetime reliability enhancement, *Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 520–531, 2005.

[73] Y. Li et al., Process-variation- and random-dopants-induced threshold voltage fluctuations in nanoscale CMOS and SOI devices, *Microelectronic Engineering*, volume 84, No. 9–10, pages 2117–2120, 2007.

[74] Y. Ye et al., Statistical Modeling and Simulation of Threshold Variation Under Random Dopant Fluctuations and Line-Edge Roughness, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, No. 99, pages 1–10, 2010

[75] M. Drapeau et al., Double patterning design split implementation and validation for the 32nm node, *Proceedings of SPIE,* volume 6521, page 652109, 2007.

[76] S. Owa and H. Nagasaka, Immersion Lithography; its potential performance and issues, *Proceedings of SPIE*, volume 5040, page 724, 2003.