

SODA: A Low-power Architecture For Software Radio

Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge
Advanced Computer Architecture Laboratory
University of Michigan at Ann Arbor
{liny, leehzz, mw, yoavh, mahlke, tnm}@umich.edu

Chaitali Chakrabarti
Department of Electrical Engineering
Arizona State University
chaitali@asu.edu

Krisztián Flautner
ARM, Ltd.
Cambridge, United Kingdom
krisztian.flautner@arm.com

ABSTRACT

The physical layer of most wireless protocols is traditionally implemented in custom hardware to satisfy the heavy computational requirements while keeping power consumption to a minimum. These implementations are time consuming to design and difficult to verify. A programmable hardware platform capable of supporting software implementations of the physical layer, or software defined radio, has a number of advantages. These include support for multiple protocols, faster time-to-market, higher chip volumes, and support for late implementation changes. The challenge is to achieve this without sacrificing power. In this paper, we present a design study for a fully programmable architecture, SODA, that supports software defined radio — a high-end signal processing application. Our design achieves high performance, energy efficiency, and programmability through a combination of features that include single-instruction multiple-data (SIMD) parallelism, and hardware optimized for 16bit computations. The basic processing element is an asymmetric processor consisting of a scalar and SIMD pipeline, and a set of distributed scratchpad memories that are fully managed in software. Results show that a four processor design is capable of meeting the throughput requirements of the W-CDMA and 802.11a protocols, while operating within the strict power constraints of a mobile terminal.

1. INTRODUCTION

Communication has become one of the central uses of computing technology over the years. Applications that facilitate inter-personal communication, such as desktop publishing, graphic design, email, and web browsing, have been primary factors in driving the evolution of microprocessors and computer systems. Meeting the processing requirements of applications has historically dominated the concerns of processor and system architects. With the proliferation of wireless mobile communications, the problem emphasis has shifted to networking protocols and signal processing that are required to sustain the necessary bandwidth of these applications. In recent years, we have seen the emergence of an increasing number of wireless protocols that are applicable to different types of networks. Figure 1 lists some of these wireless protocols and their application domains. Software Defined Radio (SDR) promises to deliver a cost effective and flexible solution by implementing the wide variety of wireless protocols in software. In this paper, we present a design study for a fully programmable architecture, SODA (Signal-processing On-Demand Architecture), that supports

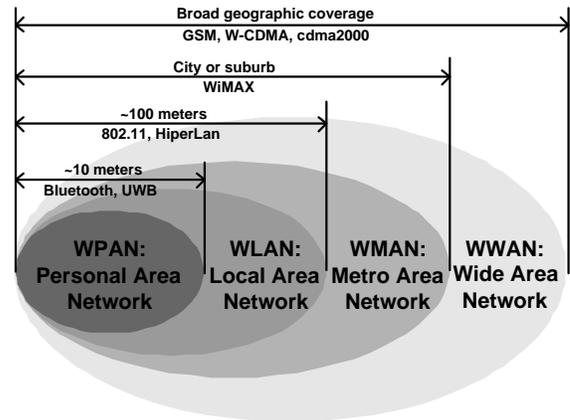


Figure 1: Categories of Wireless Networks

SDR.

We use two representative, but very different, wireless protocols to understand the architectural requirements of physical layer signal processing for SDR. Wideband code division multiple access (W-CDMA) [14] is one of the dominant third generation cellular protocols for multimedia services, including video telephony on a wireless link. W-CDMA improves over prior cellular protocols by increasing the data rate from 64Kbps to 2Mbps, and unifies a single service link for both voice and packet data. 802.11a [1] is a standard wireless local area network (WLAN) protocol that supports data rates from 6Mbps to 54Mbps. WLAN protocols are intended for indoor use with stationary terminals. We chose these two benchmarks since they are sufficiently different from each other algorithmically, and are representative of the large spectrum of algorithms that need to be supported by an SDR platform.

Software Defined Radio Challenges and Benefits.

The operation throughput requirements of W-CDMA and 802.11a are already an order of magnitude higher than the capabilities of modern DSP processors, a gap that is likely to grow in the future. Figure 2 shows the demands of these protocols in terms of throughput and power requirements. A comparison of the theoretical peak performance throughput and power consumption of SODA with other existing DSP, media, and general-purpose processor systems is also illustrated. The results are calculated for 16bit fixed point operations. We see that while most DSP processors operate at around 10 Mops/mW, most wireless protocols need 100Mops/mW. This is why most wireless protocols to date

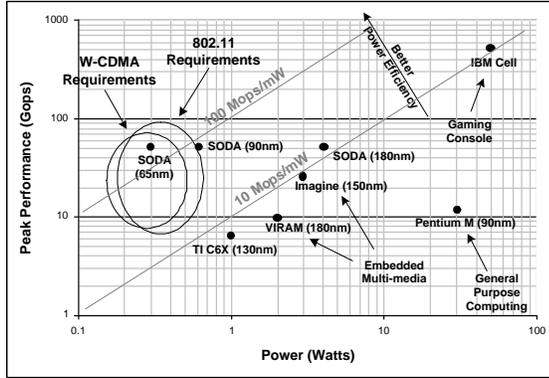


Figure 2: Throughput and power requirements of the W-CDMA and 802.11 wireless protocols, and computational efficiency of SODA and other DSP and general purpose processors

have been implemented with custom hardware. While custom hardware can meet the operational requirements, a programmable solution offers many potential advantages:

Multi-mode operation is enabled by running different protocols depending on the available wireless network — GSM in Europe, CDMA in the USA and some parts of Asia, and 802.11 in the coffee shop. This can be accomplished with less hardware than would be required by multiple custom hardware implementations.

Time to market of a protocol implementation is lower because the hardware can be reused. The hardware integration and software development tasks can progress in parallel.

Prototyping and bug fixes is enabled for next generation protocols on existing silicon with a software change. Continuing evolution of the specification can still be supported by changing the software after the chipset and even the device has been manufactured. Algorithmic improvements can be deployed without redesign.

Chip volumes are higher as the same chip can support multiple protocols without requiring hardware changes.

We believe that the use of programmable systems for wireless protocols is inevitable due to the need to support multiple increasingly complex protocols.

SODA Architecture. The proposed programmable architecture, SODA, can meet the extraordinarily high performance requirements of current and future wireless protocols, use reasonable hardware area, and operate within an embedded DSP processor’s power budget. The architecture is made up of four cores, each containing asymmetric dual pipelines that support scalar and 32-wide SIMD execution. The arithmetic units are customized for 16 bits, and the register files and software-controlled scratchpad memories need only a few ports. Our results show that in a 90nm implementation, our architecture meets the throughput requirements of the 2Mbps W-CDMA protocol and 24Mbps 802.11a running at only 400MHz. The area requirement is projected to be $6.7mm^2$. At the nominal operating voltage of 1V, this translates into a power consumption of less than 500mW. This number includes an ARM Cortex-M3 [2] control processor which is responsible for part of the protocol processing.

The main contributions of this paper are the following:

1. A description of the implementation and analysis of

two wireless protocols (W-CDMA and 802.11a) used for our benchmarking. Unlike other studies, we take the requirements of the entire protocol, not just a collection of stand-alone kernels, into account - Section 2.

2. A design study of a fully programmable wide-SIMD architecture, SODA, that can meet the power and performance requirements of high-end wireless protocols. The discussion outlines the architectural design choices and trade-offs required for an application-domain specific multiprocessor architecture for SDR - Sections 3 and 4.
3. An evaluation of the wireless algorithms on the proposed architecture which shows that high-end embedded programmable systems can meet the wireless protocols’ throughput requirements - Section 5.

2. ANALYSES OF WIRELESS PROTOCOLS

The majority of previous architectural studies on DSP applications have focused mainly on small DSP kernels in existing benchmark suites, such as MediaBench [18]. Such an approach cannot be used here since wireless protocols have complex inter-algorithm interactions that cannot be characterized by studying individual algorithms in isolation. Therefore, we have implemented the complete W-CDMA and 802.11a protocols’ physical layers in C to study the behavior of wireless protocol operations in [19]. A summary of our findings for W-CDMA and 802.11a is given in Table 2 in Section 5.3. Through the implementation of both protocols, we have found system-level challenges that have not been addressed in the literature, as well as many algorithmic-level implementation details that could not have been discovered through compiler analysis. Here we summarize our key observations into two categories: protocol system-level behavior and DSP algorithm-level behavior.

2.1 System-level Behavior

DSP Kernel Macro-Pipelining – Wireless protocols usually consist of multiple DSP algorithm kernels connected together in feed-forward pipelines. Data are streamed through kernels sequentially, as shown in Figure 3. With no data temporal locality, cache structures provide little additional benefits, in terms of power and performance, over software controlled scratchpad memories.

Low-throughput Inter-kernel Communication Traffic – Between DSP algorithm kernels, data are transferred as scalar variables. The traffic throughput of the protocol systems are not very high (as in 7.68MBps for W-CDMA receiving front-end). This implies that inter-kernel data traffic can be mapped onto low-throughput, low-power inter-connects with minimal performance degradation.

Heterogeneous Inter-kernel Communications – Some inter-kernel communications can be streamed, where the receiving kernel can process input data individually (i.e. filters). Other inter-kernel communications must be buffered, as the receiving kernels require blocks of the data (i.e. Interleaver and Turbo Decoder). Kernels with the same throughput, but different communication patterns will result in dramatically different hardware requirements. Streamed kernels need only small FIFO queues, but the buffered kernels require a large memory space.

Real-time Deadlines – W-CDMA has multiple periodic deadlines. Meeting these deadlines is one of the challenges

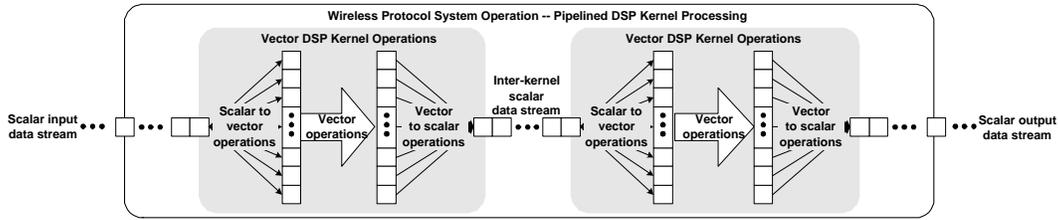


Figure 3: Wireless Protocol Characteristics: system-level macro-pipeline; algorithm-level vector operations

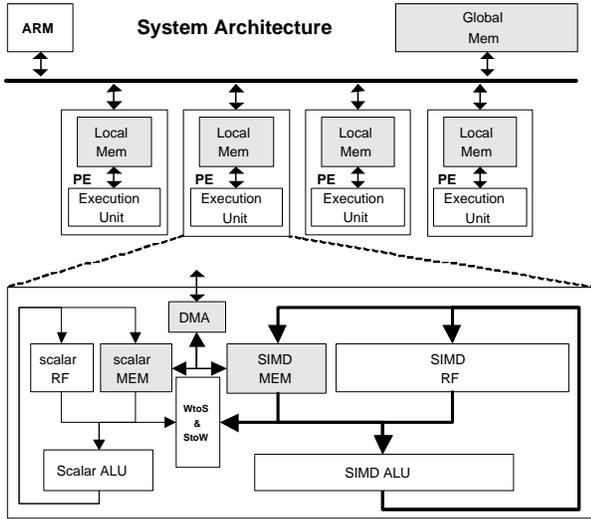


Figure 4: Overview of SODA multi-core DSP architecture

that has not been addressed in previous published DSP architectural studies. Meeting real-time deadlines requires concurrent execution management for multiple DSP algorithms.

2.2 Algorithm-level Behavior

High Data-Level Parallelism – Most of the computationally intensive DSP algorithms have abundant data level parallelism. For example, the searcher, the heaviest workload of the W-CDMA protocol, can be represented by 320-wide vectors. (See Table 2 in Section 5.3)

8 to 16bit Data Width – Most algorithms operate on variables with small values. Our analysis of W-CDMA and 802.11a suggests that the architecture should provide strong support for 8 and 16 bit fixed point operations. 32 bit fixed point and floating point support is not necessary.

Scalar-Vector operations – Because data are transferred as scalar data streams, but processed as vector variables, efficient scalar-vector conversion operations are needed to convert the inter-algorithm scalar variables into vector variables for intra-algorithm vector processing, as shown in Figure 3.

3. ARCHITECTURAL TRADEOFFS

In order to meet the high computational requirements of SDR with stringent power budgets, we propose a multiprocessor architecture shown in Figure 4. It consists of multiple processing elements (PEs), a scalar control processor, and global scratchpad memory, all connected through a shared bus. Each PE consists of a scalar unit and a wide SIMD unit. The wide SIMD unit runs most of the compute-intensive

algorithms, such as the searcher, FFT, filter, Viterbi, and Turbo decoder, while the scalar unit is used to support many of the DSP algorithms that are scalar in nature and cannot be parallelized. The ARM Cortex-M3 [2] core is mainly used as a system controller to handle the protocols’ control code and state transitions. There are no cache structures in this system: both the local memories and the global memory are managed by software through each PE’s DMA controller to which all the local and global memories are visible.

In Section 3.1, we first address the major challenges for supporting the wireless protocols in SODA. This includes managing concurrent DSP algorithm executions, controlling inter-algorithm communications, and meeting real-time deadlines. In Section 3.2, we address the architectural trade-off for supporting individual DSP algorithms, and in Section 3.3, we compare our design with existing industry and research solutions.

3.1 Multi-core System Design Tradeoff

Concurrent Algorithm Execution Support. Most commercial DSPs (i.e. TIC64x [3]) have a single execution context to avoid extra power consumption associated with supporting concurrent thread execution. However, periodic real-time deadlines require algorithms to compute with different data rates, and with different latencies. Realizing complex protocols such as W-CDMA with a single threaded system would require sophisticated operating system scheduling and complex context switching hardware—high-end micro-architecture techniques that are too expensive for wireless protocols. In order to fully support wireless protocols, it is necessary to support efficient concurrent DSP algorithm execution.

Static Multi-core Scheduling Vs. Multi-threading. Traditional micro-architectural techniques—such as simultaneous multithreading and cache coherency—that were originally developed for server-class multiprocessors provide a convenient abstraction to the programmer but may not be the most efficient for implementation in high-throughput embedded systems. Strict real-time requirements also require deterministic architectural behavior. This implies that micro-architectural features that trade-off good average-case performance for non-deterministic and poor worst-case performance (e.g., caching, multi-threading and prediction) are not well suited to these applications.

To reduce hardware complexity and produce efficient deterministic code behavior, we propose not to provide multi-threading nor cache coherency support. Instead, each protocol pipeline is broken up into kernels, and each kernel is assigned statically to a processing element which is then statically scheduled to execute according to the algorithm data flow. This model grew out of our observations that inter-kernel communication throughputs are low, and intra-kernel

computational throughputs are high. Therefore, the static scheduling approach results in less communication traffic compare to the case when kernels are split into threads.

Scratchpad Memory with Data Streaming. In addition to data computation, programmers also need to handle the data communications between algorithms. These inter-algorithm communications are usually data streaming buffers that are ideal for non-blocking DMA transfers. While the processors operate on the current data, the next batch of data are streamed between the memories and register files in the background. Streaming computations have been previously proposed for multi-media processor architectures, including the Imagine processor [6], and the IBM Cell processor [13]. They have shown that scratchpad memories, instead of cache, are best suited for streaming applications. We find that streaming computation also fits wireless protocol computations very well.

Macro-pipelined Message Passing Protocol. Traditional message passing protocol requires the sender and receiver to send synchronization semaphores to prevent queue over(under)flowing [10]. Because many of the inter-kernel communication channels are streaming data with frequent periodic real-time deadlines, this would require frequent synchronizations in order to multiplex between all of the different streaming channels. Our approach is to do message passing with global synchronization. We choose to use a real-time macro clock for communication and synchronization that is software controlled by the ARM controller. At every macro clock tick boundary, a list of PEs that needs to be synchronized is provided to the controller. This operation is analogous to the clocked scheme in a traditional pipelined processor, in which each PE behaves like a processor pipeline stage. An example of the W-CDMA execution using this approach is shown in Figure 10b in Section 5.2.

3.2 DSP Processor Design Tradeoff

Vector vs. SIMD vs. VLIW. Traditional embedded DSP processors typically fall under one of three categories: Vector, SIMD and VLIW. Vector architectures have native support for vector operations, with each instruction capable of computing arrays of data sequentially. However, due to the need to support computations on vectors with dynamically changing widths and strides, traditional vector processors are performance limited by their centralized register file's large number of access ports [17]. The SIMD approach separates the register file into clusters, reducing accessing ports' complexity. However, in most commercial solutions, SIMD width is conservatively limited between 4 to 8 wide, due to data array alignment difficulties in general purpose DSP computations [15]. VLIW architectures support ILP (Instruction Level Parallelism) very well, as memory operations can be done concurrently with multiple data computations. However, they are not very well suited for vector DLP (Data Level Parallelism), as each data computation requires its own instruction. Through our application analyses, we find that a hybrid combination of the three architectural styles provides a good match for the characteristics of most wireless protocols.

Wide SIMD Execution Unit. The computation intensive DSP algorithms in wireless protocols usually contain operations on very wide vectors. In addition, vector widths and strides are always known during compile time. Although vector architecture is a good fit for wide vector computation,

the extra hardware support for dynamic vector management is unnecessary, as all data operations can be statically scheduled. This favors a wide SIMD-styled clustered execution. Traditional SIMD architectures have a short SIMD width due to the difficulties in vector data alignment. In addition, general purpose SIMD accelerators usually need to support a large range of data sizes; (for example, Intel MMX [22] supports 8/16/32/64 bit SIMD operation). Therefore, the bottlenecks of a SIMD system are often the data movement and alignment operations, not data computation operations. Previous studies have addressed this problem through complex multi-ported memories and register files [15], or a full crossbar interconnect [11]. In the context of the power budgets for mobile devices, these are high-power solutions.

Wireless protocols' DSP algorithms have well-defined intra-vector data permutation patterns, and they operate on 8bit and 16bit data. These traits significantly simplify the data movement requirements. Therefore, we can afford to scale up the SIMD width to exploit DSP algorithms' very wide vector operations. In our design, the processor consists of 32 clusters. Each cluster contains 1 ALU and a simple register file with 2 read ports and 1 write port. The interconnect between the clusters is a relatively simple shuffle exchange network [24], which has been shown to support our target algorithms' data rearrangements efficiently.

Restricted Vector ISA. From our algorithm analysis, we find that all of the DSP algorithms can be implemented by vector operations with predicated execution and permutations. Both of these enhancements have been well studied, such as CRAY-1 [9] and the VIRAM SIMD instruction set [16]. However, unlike these previous vector ISAs, our design is more restricted — the width of the vector is set to the SIMD execution width, because there is no need for supporting unknown vector lengths and strides.

Asymmetric VLIW Instructions. In addition to the heavy vector computation, there are many small, yet equally important scalar DSP algorithms in wireless protocols. Wide SIMD execution units are too inefficient in supporting these scalar and narrow SIMD operations. Therefore, we provide a scalar execution unit for this purpose. Scalar operations can be executed concurrently with SIMD operations. This VLIW is asymmetric because instructions for SIMD pipeline cannot execute on the scalar pipeline, and vice versa. In addition, there is an AGU pipeline for memory accessing and address calculations.

3.3 SDR Architecture Survey

Recently, there has been tremendous interest in SDR in both the academia and industry, resulting in a wide range of proposed architectural solutions. These include highly parallel VLIW machines, fine-grained FPGA-like computation fabrics, coarse-grain MIMD architecture, SIMD, and vector architectures. A summary of SODA architecture and a few representative embedded processors from each category is shown in Table 1.

Hybrid-SIMD based SDR Solutions. The Sandblaster communication processor [12] is one of the most interesting SDR solutions. It is a complete solution that implements cellular and WLAN protocols. Similar to the SODA architecture, each Sandblaster processor consists of a scalar processor and a high throughput vector processing unit. However, the two architectures are very different at the system-level. Unlike the static multi-core scheduling used

	System-level Configuration	PE Configuration	PE Memory Organization	PE RegFile Organization	Fixed-point Data Precisions (bits)
SODA	1 ARM + 4 PE Static Scheduled	VLIW with SIMD ops	Clustered Scratchpad	Scalar+SIMD	16
Sandbridge Sandblaster	4 PEs Multi-threading	GPP+SIMD	Multi-core Cache	SIMD	16/32
Phillips EVP	Uni-processor	VLIW with SIMD ops	Scratchpad	VLIW+SIMD	8/16/32
XiRisc	Uni-processor	VLIW+embedded FPGA	Cache	Unified	8/16/32/64
QuickSilver	MIMD	Scalar+ASIP	Scratchpad	Distributed	8/16/32/64
TI C64x	Uni-processor	VLIW	Cache	Cluster	8/16/32
Imagine	Uni-processor	SIMD with VLIW op	Scratchpad	Stream+SIMD	8/16/32
VIRAM	Uni-processor	Vector	Scratchpad	Vector Cluster	8/16/32/64
IBM Cell	1 PPE + 8 SPE	Out-of-order SIMD	Unified Scratchpad	SIMD	1-128

Table 1: Architectural comparisons between SODA and other embedded DSP and multi-media processors

in SODA, Sandblaster supports concurrency through multi-threading. While this is a powerful technique, it requires additional hardware overhead, such as cache coherency. Phillips EVP processor [25] is also another SDR solution. Similar to SODA, it is also a VLIW architecture with wide SIMD execution units. However, EVP differs in its VLIW organization—it can execute multiple types of wide SIMD operations in parallel. While this provides support for ILP, it also requires a more complex register file. Because there is very limited ILP among vector computations, this extra level of parallelism does not add much to performance. In addition, while EVP is designed to be a stand-alone DSP processor, SODA is a multi-processor architecture designed for supporting system-level protocol execution.

Reconfigurable array based SDR Solutions. There have been numerous SDR solutions based on fine-grained reconfigurable FPGA-like computation fabrics. Examples of such solutions include picoArray [7], and the XiSystem’s XiRisc [20]. Some of these solution, such as the XiRisc, also include a scalar/VLIW processor, with the reconfigurable logic acting as an accelerator. One of the major drawbacks of the fine-grain computation fabrics is the high communication cost of data shuffling within the computation fabrics. In addition, programming such devices to meet real-time constraints is difficult.

Heterogeneous MIMD based SDR Solutions. These MIMD styled architectures contain a system of heterogeneous coarse-grained processing elements, with each type of PE tailored to a specific DSP algorithm group. Examples include Intel RCA [8], and QuickSilver [5]. Both RCA and QuickSilver have 3 or 4 different types of PEs, ranging from simple scalar processors to ASIP(Application Specific IP) Viterbi/Turbo accelerators. These heterogeneous SDR systems provide a trade-off between overall system flexibility and individual kernel computational efficiency. While a homogeneous processor system can distribute the system workload among PEs, a heterogeneous processor system must provide enough units for the worst case workloads of each type of PE. As shown in Table 2 in Section 5.3, W-CDMA and 802.11 require very different types of DSP algorithms. Therefore, heterogeneous systems are more-likely to under-utilize their hardware, resulting in less efficient overall system operations.

VLIW DSP Solutions. The TI TMS320C64x DSP processors [3] are highly parallel VLIW machines, that can achieve high performance. However, because data level parallelism is much more prevalent than instruction level par-

allelism, the benefits offered by VLIW are not utilized in wireless applications. The instruction execution power consumption is relatively higher than other solutions, and thus the overall computational efficiency is lower. These solutions typically cannot meet the SDR performance and power requirements by themselves. Therefore, they often include ASICs accelerators for performance enhancements.

Vector/SIMD based Multi-media Solutions. Vector and SIMD embedded processors have been very popular in the multi-media domain. Among them are the IBM Cell Processor [13], VIRAM Project [16], and Imagine Project [6]. IBM’s Cell processor is architecturally similar to our design at the system-level, with a controller (PPE) and multiple SIMD processors (SPE). However, the SPE is a generic SIMD-based processor, whereas ours is a domain-specific VLIW+SIMD processor targeted at SDR. Although the Cell processor has higher overall computational throughput than our processor, it was never designed to be a mobile solution, and its power consumption is 100x greater than the budget for a wireless protocol. Imagine [6] uses SIMD-based execution, where each instruction is a VLIW operation. The VIRAM [16] design is an improved vector processor designed for multi-media workload. Again, these processors were not designed specifically for wireless applications, but instead for general multimedia applications. The SODA architecture is designed specifically for wireless protocols and, as a result, can execute them much more efficiently.

4. SODA IMPLEMENTATION

In this section, we first discuss our PE design. We then discuss PE SIMD design trade-offs in Section 4.2. And finally, we discuss some of our architecture design techniques for low-power operations in Section 4.3.

4.1 PE Design

Overview. Figure 5 shows the architectural details of a single PE. There are three pipelines that run in lock-step: a scalar pipeline, an SIMD pipeline, and an AGU (Address Generation Unit) pipeline. The scalar pipeline consists of one 16bit datapath. Its main purpose is to support scalar DSP algorithms, as well as the DSP algorithms’ control code. In most DSP algorithms, the core kernels are made up of shallow nested loops (one or two levels). Because of this, we chose not to implement a branch predictor, but add loop counter-based branch instructions instead.

The SIMD pipeline consists of a 32-way 16bit datapath, with 32 16bit arithmetic units working in lock-step. It is de-

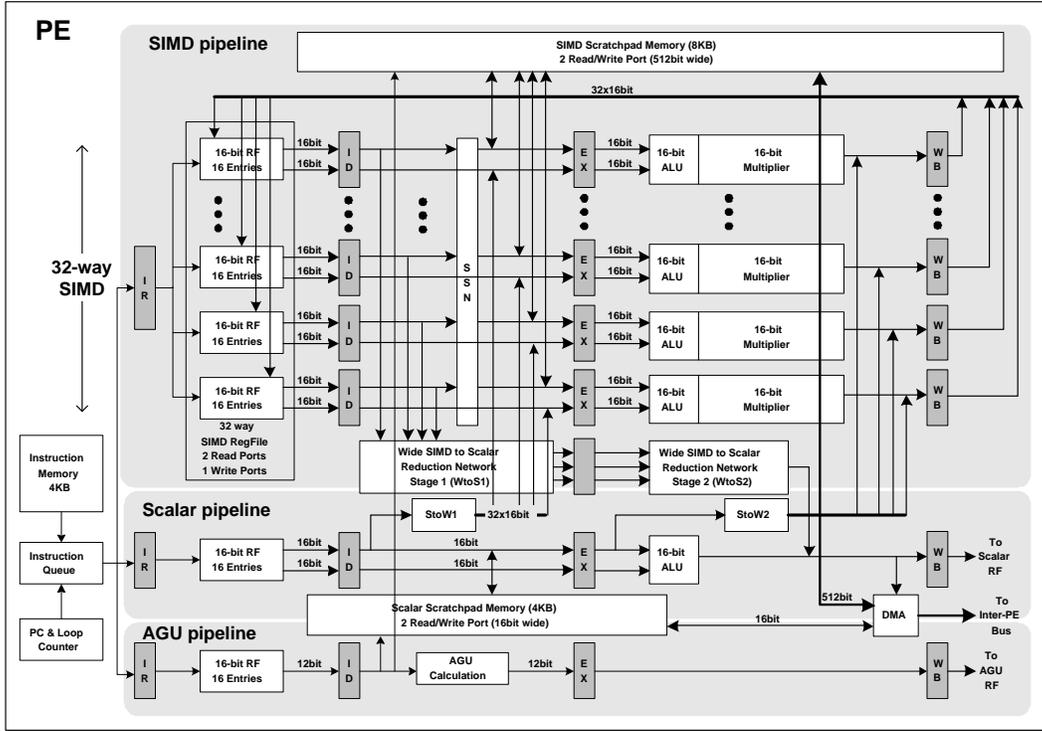


Figure 5: Processing Element Architectural Diagram

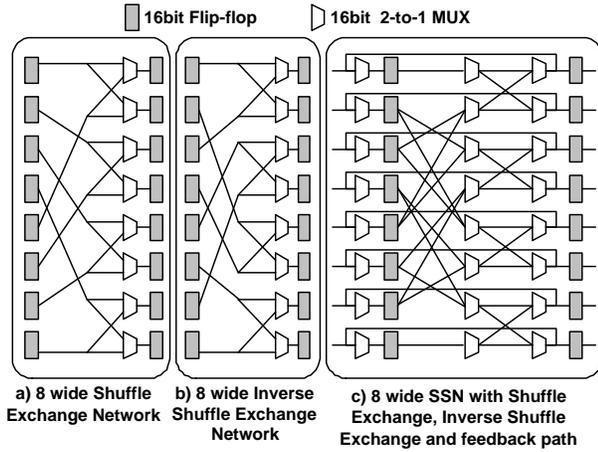


Figure 6: 8-wide SIMD Shuffle Network(SSN)

signed to handle computationally intensive DSP algorithms. Each datapath includes a 2 read-port, 1 write-port 16 entry register file, and one 16bit ALU with multiplier. The multiplier takes two execution cycles when running at the targeted 400MHz. Intra-processor data movements are supported through the SSN (SIMD Shuffle Network). Figure 6c shows a simplified 8-wide version of the network, whereas SODA’s SSN is actually 32-wide. SSN is consisted of a shuffle exchange (SE) network (shown in Figure 6a), an inverse shuffle exchange (ISE) network (shown in Figure 6b), and a feedback path. Previous work [26] has shown that any permutation of size N can be done with $2\log_2 N - 1$ iterations of either the SE or ISE network, where N is the SIMD width. For the permutation patterns of SDR algorithms, we found that we can reduce the number of iterations if we

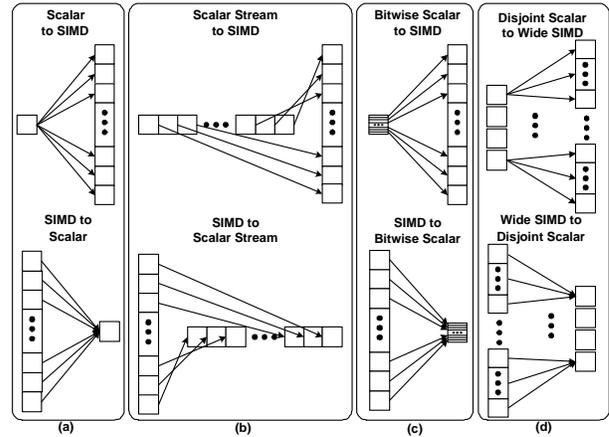


Figure 7: Scalar-SIMD Operations for Various DSP Algorithms

include both the SE and ISE networks. In addition to the SSN network, a straight-through connection is also provided for data that does not need to be permuted.

The AGU pipeline handles DMA (Direct Memory Access) transfers and memory address calculations for both scalar and SIMD pipelines. In wireless protocols, DSP kernels process data in streams through data queues, that is supported by the AGU’s DMA capability. In addition, it also handles local memory accesses for both the scalar and SIMD pipelines.

Asymmetric Dual SIMD Pipeline. As explained in Section 2, inter-kernel communications are via scalar streams, but intra-PE computations are vector operations. Therefore, support for a scalar-vector interface between the scalar

and SIMD pipeline is needed. In our architectural diagram, Figure 5, this is shown as the StoW and WtoS units. The StoW network spreads a scalar value into a wide vector, and WtoS network reduces a wide vector into a scalar value. The StoW1 unit is for spreading scalar register values into the SIMD pipeline, and the StoW2 is for spreading scalar memory values into the SIMD pipeline. The WtoS network is a 32-to-1 reduction tree. It is pipelined into two stages, WtoS1 and WtoS2 units.

From our benchmark analysis, we found four types of scalar-vector operations, shown in Figure 7.

Scalar-SIMD Operations – These operations spread one scalar variable into a vector, and reduce a vector down to one scalar variable, as shown in Figure 7a. Filter, Viterbi Decoder, and Turbo Decoder fall into this category.

ScalarStream-SIMD Operations – These operations transfer an array of scalar variables directly into a vector, as shown in Figure 7b. If implemented directly through the scalar-SIMD StoW network, the transfer operations would take up to 32 cycles, as each scalar value is spread sequentially from the scalar to SIMD pipeline. However, with the AGU unit, we can redirect our DMA to transfer data directly into the SIMD memory, bypassing the scalar pipeline. The algorithms in this category include FFT and the RAKE receiver.

BitwiseScalar-SIMD Operations – These operations spread the bits of a scalar value into a vector, as shown in Figure 7c. Vectors with 1bit elements are common in DSP. One example is the searcher, which correlates vectors of individual bits that make up the received signal stream. This special expansion/reduction functionality is supported in the StoW and WtoS networks.

DisjointScalar-SIMD Operations – These operations support expansion/reduction operations between multiple disjoint scalar values and wide SIMD vectors, as shown in Figure 7d. This feature is useful for algorithms with native vector width less than the SIMD’s width. An example is the Turbo decoder. It has a native vector width of 8 elements. Our SIMD datapath can operate on 32 elements at once. Running a sequential version of the Turbo Decoder only utilizes 25% of the SIMD pipeline. Using the sliding window technique, Turbo decoder can be parallelized to process 4 data streams in lock-step SIMD-style execution. However, the 4 data streams require 4 separate scalar values. In order to handle expansion and reduction of multiple disjoint scalar values, the StoW and WtoS networks are modified to include a 4-wide 16bit disjoint scalar (DS) register. For StoW expansion, four separate values are first read sequentially from the scalar pipeline into the DS register, and then 4 8-wide expansions are performed. For WtoS reduction, the SIMD vector is first reduced to 4 16bit values, stored in the DS register. The scalar pipeline then processes these 4 values sequentially. Although the scalar operations are still processed sequentially, because the majority of the computations are vector operations, this scalar overhead has minimal effect on the overall performance.

Special DSP Instructions. Implementing customized complex instructions is very common in DSP processors. Typical examples are Multiply-Accumulate (MAC) and saturated arithmetic instructions. The first type of custom or intrinsic operations includes special vector permutations supported by the SSN. The Vector Compare & Select (VCS) operation, needed in Viterbi and Turbo decoders, compares

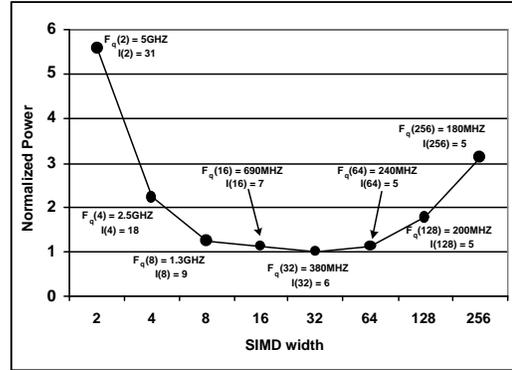


Figure 8: Average normalized power of a 4-PE configuration for achieving the computational requirements of W-CDMA and 802.11a in 180nm technology

and selects between two adjacent vector elements. The butterfly operation is implemented to enhance the FFT performance. These two types of permutation operations are micro-coded into the controller of the SSN network.

The second type of intrinsic instruction is used to convert a vector to a single scalar value. This instruction is heavily used in the synchronization and modulation kernels. An example, the Vector Max instruction, is crucial for Viterbi and Turbo since calculating the maximum value of a vector without special hardware support is very inefficient.

The last type of intrinsic instruction is predicated negation. This instruction uses two 1bit vectors to control the sign of the two ALU operands, and conditionally negates the operands before ALU execution. This is equivalent to a multiplication of the operands with a 1bit number that can be either +1 or -1. It is used for auto-correlation, modulation and FFT operations.

4.2 SIMD Design and Tradeoffs

To justify the SODA system configuration with wide SIMD pipelines, we have done a study on the first-order power consumption trade-offs SIMD width and frequency. This study was done using 180nm technology. We estimate that 40GOPS would be required in order to meet the realtime computation requirements of W-CDMA and 802.11a. Furthermore, we find that both W-CDMA and 802.11a can be partitioned into 4 major task groups that are relatively balanced (see Figure 10c for the W-CDMA implementation). So, in this paper, we examine the power consumption of a 4 PE system. In a 4 PE system, each PE will need to supply approximately 10GOPS. On one end of the spectrum, the PE can be a 2-wide SIMD running at 5GHZ; and on the other end, it can be a 256-wide SIMD running at 180MHZ. Intuitively, the optimal SIMD configuration should lie within these two extremes. Figure 8 summarizes our findings to determine the power consumption of a PE.

The following is our methodology for calculating the first-order power consumption P for a single PE with a workload requirement of T GOPS, and an SIMD width of w . We assume a homogeneous system with PEs having the same SIMD configuration and frequency. If N is the number of algorithms running on the SIMD pipeline, then $T = \sum_{i=1}^N T_i$, where T_i is the workload of the i th algorithm in the protocol.

The required frequency of a PE, $F(w)$, as a function of w , can be calculated by summing up the frequencies of each individual algorithm.

$$F(w) = \sum_{i=1}^N \left(\frac{T_i}{V_i} \left\lceil \frac{V_i}{w} \right\rceil \right) \quad (1)$$

Where V_i is the native vector width of algorithm i . The first term in the summation, $\frac{T_i}{V_i}$, calculates the frequency for meeting the computational requirement of T_i GOPS in terms of number of vector operations per second. The second term, $\lceil \frac{V_i}{w} \rceil$, calculates the number of cycles to perform a vector operation of size V_i on a SIMD processor of width w . In the case of $w > V_i$, the SIMD is under utilized since the vector is narrower than its SIMD width. The exception is the Turbo decoder, where we can use the sliding window technique to compute multiple windows in parallel and thereby exploit the wider SIMD width. In our analysis, T_i and V_i are calculated based on the W-CDMA and 802.11a protocol profiling results shown in Table 2.

Given the limitation of silicon technology, there is an upper bound on the achievable frequency. PEs with narrow SIMD width that require ultra high frequency will have to implement deeper pipelines. In this study, we scale up the pipeline depth based on SODA’s 5 stage pipeline organization. There are four architectural components that contribute to the overall SIMD pipeline depth: the register file (R), ALU (A), SIMD memory (M), and SSN (S). We ignored the WtoS network, because it is not a part of the SIMD pipeline. Equation 2 expresses the overall pipeline depth I , as the sum of the register file pipelines(I_r), the ALU pipelines(I_a), and the maximum of the memory pipeline(I_m) and SSN pipeline(I_s).

$$I(w) = 2I_r(w) + I_a(w) + \max(I_m(w), I_s(w)) \quad (2)$$

In Equation 2, I_r is scaled by 2 due to the two separate pipeline stages for register read and write. Also the maximum of I_m and I_s is used because SSN and memory share the same pipeline stage. The pipeline depth of each component is obtained from synthesized Verilog, for frequency $F(w)$.

Let $E(w)$ be the energy consumption of one cycle of operation for one PE, and let $P(w)$ be PE’s power consumption.

$$E(w) = C + w(L_e I + R_e U_r + A_e U_a) + M_e(w)U_m + S_e(w)U_s + D_e(w)U_d \quad (3)$$

$$P(w) = E(w) \cdot F(w) \quad (4)$$

In Equation 3, C is the constant power overhead, due to the scalar and AGU pipeline, the instruction memory, and the DMA controller. L_e is one 16bit datapath pipeline’s flip-flop energy, R_e is one 16bit register file’s energy, A_e is one 16bit ALU’s energy, M_e is the SIMD memory energy, S_e is the SSN energy, and D_e is the WtoS reduction network energy. All of the above energy results are for one cycle of operation. Because memory, SSN, and reduction network do not scale linearly with the SIMD width, we model them empirically by synthesizing each configuration in Verilog. U_x represents the average utilization factor for component x , gathered from behavioral simulations on the SODA simulator.

Figure 8 shows the normalized power as a function of SIMD width for average W-CDMA and 802.11a workloads. Each point is annotated with its operating frequency, $F(w)$, and the number of pipeline stages, $I(w)$. We see that smaller

SIMD width configurations consume less power per cycle, but require unrealistically deep pipelines. For example, the 4-wide SIMD configuration requires 18 pipeline stages. Wider SIMD configurations have shorter pipelines with low operating frequencies, but suffer from underutilization. Figure 8 shows that the 32-wide SIMD consumes the lowest power. The 8, 16, and 64 wide SIMD also achieve similar power consumption, and would be acceptable design points. The power numbers have been derived assuming that underutilized SIMD processors still consume dynamic power for the unused SIMD units. However, we can employ simple clock-gating techniques to turn off the unused units, thereby reducing wide SIMD’s power consumptions. In addition, frequency and voltage do not scale linearly with nanometer CMOS technologies. In sub-90nm implementations, narrow width SIMD will result in deeper pipelines. Consequently, the optimal power consumption point is likely to shift to higher SIMD width in future technologies, if leakage can be contained.

4.3 Embedded Low-power Design

In order to achieve the high computational requirements, while maintaining low power requirements, SODA utilizes the following techniques:

Intrinsic Operations – Traditional DSP processors rely heavily on MAC operations to achieve efficiency. We found that many of these multiplication operations are with 1 bit values, and can be implemented by vector logic operations (Predicated Negation) that consume significantly lower power.

Clustered Register Files with 2 Read Ports and 1 Write Port – Each PE’s register file is a cluster of three separate files: an SIMD RF, scalar RF, and AGU RF. For a 32-way vector operation, each RF in our PEs requires only 2 ports. As shown in previous studies [23], increasing the number of register file ports increases the register file’s power consumption quadratically. By using less ports, our implementation saves register file power.

Fewer Memory Read/Write Ports – Each PE’s local memory is a cluster of 2 memories—4KB for the scalar memory and 8KB for the SIMD memory. In general, two memories consume lower power than one unified memory. In addition, our SIMD memory requires a 512bit read/write port, but our scalar memory only requires a 16bit read/write port. Separate memories allow us to further optimize each for its intended use.

Smaller Instruction Fetch Logic – Our VLIW instructions use a fixed sized instruction width of 64bits split into three fields—Scalar, AGU, and SIMD. Commercial DSP solutions often have variable length instruction widths of 96-128bits. In addition, they do not support SIMD instructions or a vector ISA that allows us to efficiently express long vector operations, effectively reducing an algorithm’s code size.

5. EXPERIMENTAL EVALUATIONS

5.1 Methodology

SODA Behavioral Analysis. In order to test the performance of SODA, we first developed W-CDMA and 802.11a physical layer system implementations in C. This approach enables full system performance including memory requirements, synchronization schemes, and non-parallelizable bottlenecks to be evaluated. Block diagrams of the major com-

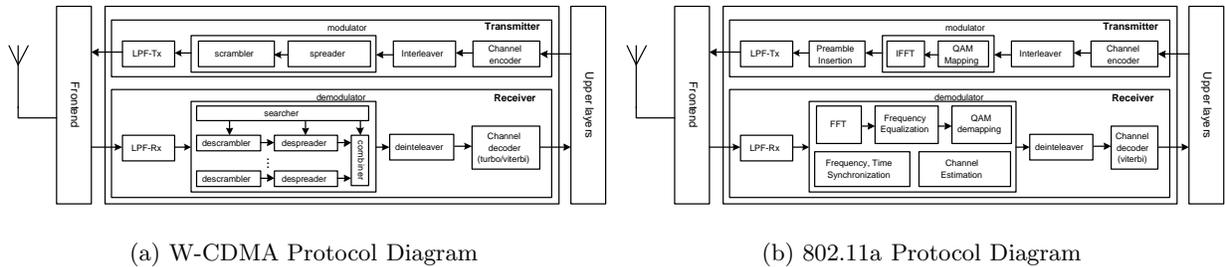


Figure 9: Physical Layer Operations of W-CDMA and 802.11a Wireless Protocols

ponents for each protocol are given in Figure 9. Next, we hand-coded the benchmarks into the SODA instruction set. To evaluate intra-kernel synchronization and data flow characteristics, we built an inter-processor network simulator based on our PE’s cycle-accurate processor simulator, and DMA transfers and bus synchronization schemes.

SODA Area And Power Model. Area estimation of our architecture was calculated using our RTL Verilog model of the SODA architecture. We synthesized our Verilog model using Synopsys Physical Compiler and Cadence Silicon Ensemble for 400MHz using the TSMC 180nm library. The memories were generated with the Artisan SRAM memory generator. The estimated area for 90nm and 65nm processes were calculated using a quadratic scaling factor.

Dynamic power was estimated using utilization factors of each PE derived from our behavioral simulations of the W-CDMA and 802.11a protocols on our system simulator. For each PE we then took the dynamic power results from Physical Compiler and used the utilization factors to calculate the dynamic power of that PE. The dynamic power of the memories were derived from the Artisan SRAM memory generator. Using the synthesized model of the PE, we extracted the interconnect power and added it to the dynamic power then summed up the PEs to calculate the total dynamic power of the system.

To scale to 90nm and 65nm, we used the Predictive Technology Models (PTM) [4] and simulated in SPICE based on a delay of 20 F04 in 180nm at 1.8V, 90nm at 1V, and 65nm at 0.8V.

The leakage power was estimated at 30% of the total power. This is in accordance with ITRS specifications for 90nm technology. We used the more conservative approach, since the PTM leakage results were lower than industry trends.

5.2 Software Defined Radio Implementations

Protocol System Implementations. For this discussion we will focus on W-CDMA instead of 802.11a, because its behavior is more complex. Figure 10 shows the real-time W-CDMA 2Mbps DCH (Dedicate CHannel) execution trace on SODA: Figure 10a shows the system execution of 1 frame of data; Figure 10b shows one slot of execution using the macro-pipelining message passing protocol; and Figure 10c shows the functional mapping of W-CDMA onto SODA. DCH is a full duplex channel consisting of the DPDCH (Dedicated Physical Data CHannel) for uplink and the DPCH (Dedicated Physical CHannel) for downlink. In the W-CDMA specification, DCH also includes the uplink DPCCCH (Dedicated Physical Control CHannel), which is not mod-

eled in this study. The uplink and downlink channels are mapped onto their own PEs. This assignment achieves a relatively balanced workload across the 4 PEs.

To better understand W-CDMA execution, consider Figure 10a. The horizontal axis is time, and the vertical axis lists the SODA’s PEs, and their real-time processing utilization. The utilization of PE1, PE2, PE3, and PE4 are 60%, 50%, 100% and 94% respectively. One W-CDMA frame contains 15 slots. There are two hard real-time deadlines that have to be met in W-CDMA. The first one is the power control critical path that controls the transmission power based on received signal quality. It needs to update periodically once per slot (0.67 msec). The critical path is the channel between the FIR Rx filter, Demodulation and Power Control. This is a streaming channel with minimal memory storage requirements. The other real-time critical path is the channel from the FIR filter to the searcher. This needs to complete within 5 msec and requires a large amount of data buffering. Figure 10b shows the multi-PE execution using our macro-pipelined message passing protocol. Data is streamed from one PE to the next, synchronizing only on the macro-clock boundary.

From the above analysis, we see that while throughput is essential, fast intra-processor kernel switching, and efficient inter-processor communication are also essential. Because wireless protocols have static run-time characteristics, compile-time scheduling of the kernels is sufficient to reduce unnecessary context switching overhead. Our macro-pipelining message passing technique reduces the inter-PE synchronization overhead by pipelining data transfers, and exploiting the streaming nature of inter-kernel communication.

5.3 Performance and Power Results

Performance Results. Table 2 provides a characterization of each kernel algorithm in W-CDMA and 802.11a in terms of extent of vectorization, vector width, bit width, etc. This characterization was used to define the SODA architecture, as described in Section 2. Table 2 also lists the throughput and latency of each kernel algorithm when implemented on SODA. The raw computations are measured in terms of number of execution cycles on a general purpose Alpha processor. The SODA computation, on the other hand, is the number of execution cycles required by the SODA vector ISA. It can be seen that large speedups are possible in many cases. For instance, W-CDMA’s searcher algorithm, which requires 26.5 Gops on a general purpose processor, is reduced to 200 Mops on SODA. Part of the speedup is due to SODA’s wide SIMD execution, and part of it is due to

Algorithms	Configurations	General Purpose Processor (Alpha)				SODA
		Vector Comp.	Vector Width	Bit Width	Raw Comp. Mcycles/sec	Comp. Mcycles/sec
W-CDMA (2Mbps)						
Scrambler	Defined in W-CDMA standard	yes	2560	1,1	240	9
Descrambler*	12 fingers, 3 base stations	yes	2560	1,8	2,600	23
Spreader	spreading factor = 4	yes	512	8	300	5
Despreader*	12 fingers, 3 base stations	yes	512	8	3,600	11
PN Code (Rx)	3 base stations	no	1	8	30	23
PN Code (Tx)	Defined in W-CDMA standard	no	1	8	10	8
Combiner*	2 Mbps Data Rate	partial	12	8	100	3
FIR (Tx)	4 filters x 65 coeff. x 3.84MSps	yes	64	1,16	7,900	307
FIR (Rx)	2 filters x 65 coeff. x 7.68MSps	yes	64	8,8	3,900	182
Searcher*	3 base stations, 320 windows	yes	320	1,8	26,500	200
Interleaver	1 frame	no	1	8	10	2
Deinterleaver	1 frame	no	1	8	10	2
Turbo Encoder	K=4	partial	3	1,1	100	2
Turbo Decoder*	K=4, 2 SOVA, 5 Iterations	partial	8	8,8	17,500	540
*These algorithms have dynamically changing workloads that are dependent on channel conditions.						
802.11a (24Mbps)						
FFT	64 points	yes	64	16	15,600	240
IFFT	64 points	yes	64	16	15,600	240
Equalizer	64 points	yes	54	16	960	120
QAM	64 constellation points	no	1	4	1	2
DQAM	64 constellation points	no	1	4	3	2
FIR (Tx)	1 filter x 33 tap x 20MSps x 2	yes	33	16	3,040	160
FIR (Rx)	1 filter x 33 tap x 40MSps x 2	yes	33	16	6,080	320
Freq. Sync.	Defined in 802.11 standard	partial	16	16	190	10
Time Sync.	Defined in 802.11 standard	partial	16	16	190	10
Interpolator	Farrow structure cubic 8 taps	yes	2048	16	4,800	250
Interleaver	1 frame	no	1	1	290	60
Deinterleaver	1 frame	no	1	16	290	60
Conv. Enc.	K=7, Rate = 3/4	partial	6	1	100	40
Viterbi Dec.	K=7, Soft Input	partial	64	8	35,000	398
Scrambler	Defined in 802.11 standard	partial	7	1	340	34
Descrambler	Defined in 802.11 standard	partial	7	16	340	34

Table 2: Kernel Algorithms in W-CDMA and 802.11a and their performance on a GPP and SODA.

	Components	Units	Area		W-CDMA 2Mbps		802.11a 24Mbps	
			Area mm^2	Area %	Power mW	Power %	Power mW	Power %
PE	SIMD+scalar Data Mem (8KB+4KB)	4	6.1	23%	87	3%	67	2%
	SIMD Register File (16x512bit)	4	1.9	7%	1077	37%	874	27%
	SIMD ALUs and Multipliers	4	6.7	25%	314	11%	609	19%
	SIMD Pipeline+Clock+Routing	4	1.5	6%	1127	38%	1157	36%
	Intra-processor Interconnect	4	1.1	4%	53	2%	53	2%
	Scalar Pipeline+Inst. Mem+Inst. Fetch	4	3.1	11%	274	9%	329	10%
System	ARM (Cortex-M3)	1	0.6	3%	5	< 1%	10	< 1%
	Global Scratchpad Memory (64KB)	1	3.6	14%	10	< 1%	80	2%
	Inter-processor Bus with DMA	1	2.0	7%	3	< 1%	26	1%
Total	180nm (1.8V @400MHZ)		26.6	100%	2950	100%	3206	100%
Est.	90nm (1V @400MHZ)		6.7		447		486	
	65nm (0.8V @400MHZ)		3.5		236		257	

Table 3: System Area and Power Summary

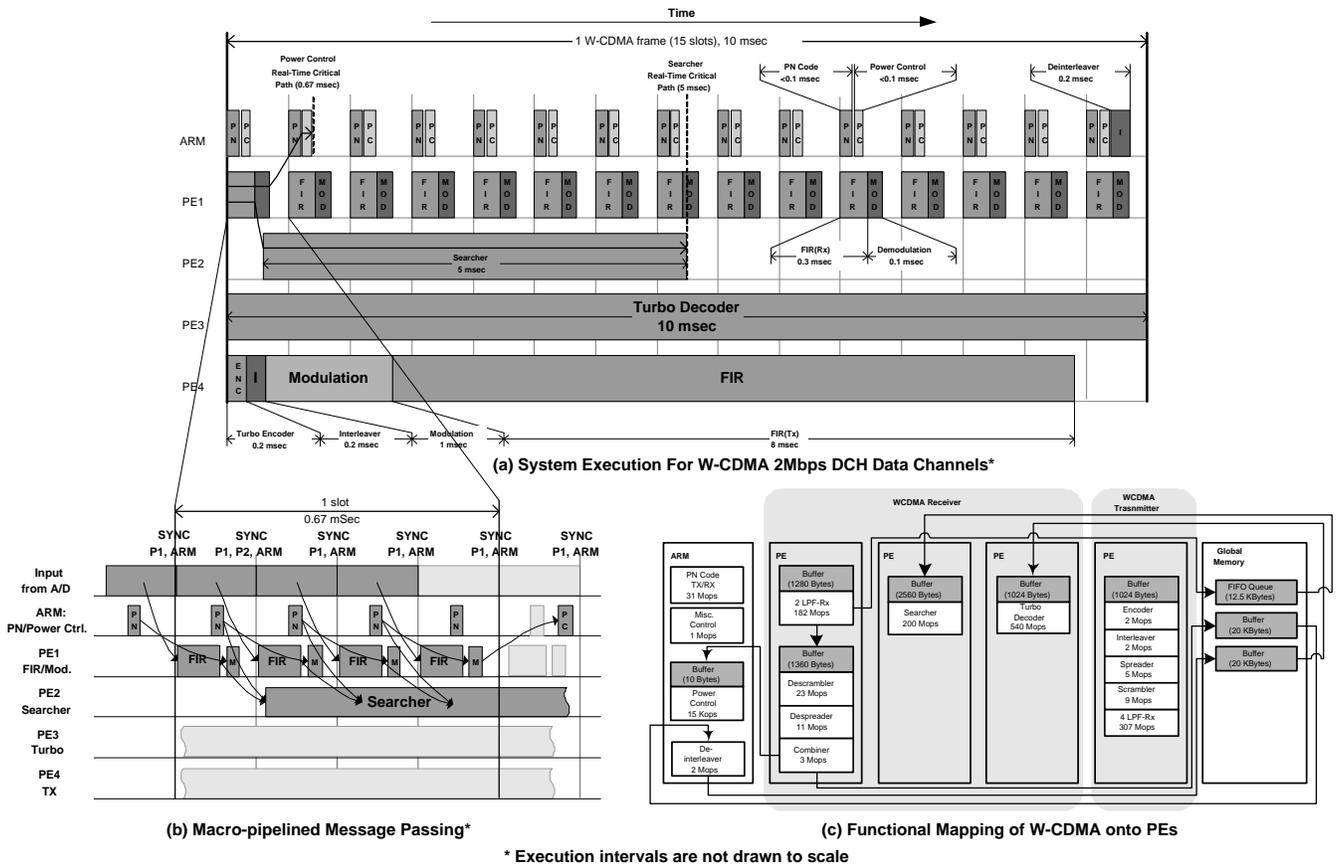


Figure 10: W-CDMA Protocol Implementation

the fact that SODA assembly code is hand-optimized.

Because W-CDMA is designed to support mobile communications, its workload is highly dependent on the environment conditions. In this study, the descrambler, despreader, combiner, and searcher are benchmarked with the worst case environment condition, because they include real-time deadlines that must be met under the heaviest workload. The Turbo decoder is benchmarked with the average case workload because it has flexible deadlines that allow its inputs to be buffered. This is why it is acceptable for the decoder to take 540Mcycles(1.35 seconds) to finish one second of computation.

Power and Area Results. Table 3 lists the area and power breakdowns of the SODA system. The wide SIMD design means the SIMD pipeline and clock logic consumes the largest amount of power. The SIMD register file is also one of the major power consumers (37% in W-CDMA and 27% in 802.11a), due to heavy utilizations during vector computations. SIMD memory power is higher for W-CDMA (87mW) than for 802.11a (67mW). This is because most 802.11a algorithms have vector width less or equal to 64, so the SIMD register values do not spill into the memory. In contrast, W-CDMA has more algorithms with long vectors that need to be buffered in memory. The SIMD ALU power consumption is significantly higher for 802.11a than for W-CDMA, because 802.11a's FFTs requires many 16-bit multiplications, whereas the majority of the W-CDMA computations are additions. In our synthesized design, a 16-bit multipliers consumes approximately 10x more power than

an 16-bit adder. This is the principal reason why 802.11a consumes more power than W-CDMA. The intra-processor interconnect consumes very little power for both 802.11a and W-CDMA. Finally, low inter-PE communications implies that the bus power consumption is also not a concern (3mW for W-CDMA and 26mW for 802.11a). Overall, the results show that a power efficient wide SIMD multi-PE architecture can be designed using simple register files, partially connected intra-processor interconnect, and a low power bus-based inter-processor network.

The area results, shown in Table 3, indicate that the ALUs with multipliers and the scratchpad memories take the largest area in the PE. In addition, the PE's local memories (48KB) occupy a larger area ($6.1mm^2$) than the 64KB global memory ($3.6mm^2$), because the local memories are dual-ported, with one port dedicated to the DMA, whereas the global memory is a 32bit single ported memory. The intra-processor interconnect, including the SSN and the WtoS reduction network, is only 4% of the total area. This means that the interconnect network is not a limitation for 32-wide SIMD systems. Of course, if this number were scaled to hundreds, then the interconnect network may start to become a limitation.

Technology Scaling and Power Optimizations. At 180nm, SODA's power consumption is 3W. This is too high for embedded mobile devices. A typical cellular phone's power budget for the physical layer is around 200mW [21]. To see if this constraint can be met, we have estimated the power and area of SODA at state-of-the-art technology

nodes of 90nm and 65nm using the Predictive Technology Models. Designs in both technologies fall within the range of acceptable power consumption — 450mW and 250mW respectively. There are other factors that we have ignored that would further reduce power consumption. These include a greater use of custom design, and the observation that many of the W-CDMA algorithms need only 8bit arithmetic. Our studies were based on unoptimized synthesis. In a volume production setting, much of the datapath would be implemented with custom designs to significantly reduce space and power. We previously synthesized an 8bit 32 wide version of SODA, and its power consumption in 90nm was about 300mW. However, 802.11 and many next generation protocols use 16bit algorithms, thus an 8bit solution will not meet future demands. There are still many important 8bit algorithms, such as Viterbi decoder. This means that power optimization techniques such as clock-gating, dynamic precision and voltage scaling can be used to reduce power consumption by dynamically adjusting between 8bit or 16bit computations and between different SIMD widths. We are investigating these issues.

6. CONCLUSION

Process technology scaling has made the transition from custom hardware to programmable architectures possible even for performance-hungry and power-limited workloads typified by wireless protocols. In this paper we describe and discuss architectural trade-offs for designing a domain specific processor for Software Defined Radio. We describe and motivate our multiprocessor, programmable wide SIMD architecture. We show that our architecture is capable of meeting the processing requirements of two widely differing protocols (W-CDMA and 802.11a) within acceptable power budgets when using a state-of-the-art technology node. Our choice of these two dissimilar protocols was to stress the flexibility of our solution. Further process scaling will enable the support of even more demanding protocols (such as UWB) in a power-efficient manner.

7. ACKNOWLEDGMENT

Yuan Lin is supported by a Motorola University Partnership in Research Grant. This research is also supported by ARM Ltd., the National Science Foundation grant NSF-ITR CCR-0325898 and CCR-0325761.

8. REFERENCES

- [1] ANSI/IEEE Std 802.11, 1999 Edition, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
- [2] ARM Cortex-M3: <http://www.arm.com/products/CPU/ARM-Cortex-M3.html>.
- [3] DSP Developers' Village, Texas Instruments: <http://dspvillage.ti.com>.
- [4] Predictive Technology Model: <http://www.eas.asu.edu/ptm/>.
- [5] QuickSilver Technology: <http://www.qstech.com/>.
- [6] J. H. Ahn et al. Evaluating the Imagine Stream Architecture. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.
- [7] R. Baines and D. Pulley. Software defined baseband processing for 3G base stations. In *4th International Conference on 3G Mobile Communication Technologies (Conf. Publ. No. 494)*, pages 123–127, June 2003.
- [8] I. Chen, A. Chun, E. Tsui, H. Honary, and V. Tsai. Overview of Intel's Reconfigurable Communication Architecture. In *3rd Workshop on Application Specific Processors*, pages 95–102, Sept. 2004.
- [9] Cray Research Inc. *The Cray-1 Computer System*, Publication No.2240008b 1976.
- [10] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.
- [11] K. Diefendorff, P. Dubey, R. Hochsprung, and H. Scales. AltiVec Extension to PowerPC Accelerates Media Processing. In *IEEE Micro*, volume 20, no. 2, pages 85–95, Mar./Apr. 2000.
- [12] J. Glossner, E. Hokenek, and M. Moudgill. The Sandbridge Sandblaster Communications Processor. In *3rd Workshop on Application Specific Processors*, pages 53–58, Sept. 2004.
- [13] P. H. Hofstee. All About the Cell Processor. In *IEEE Symposium on Low-Power and High-Speed Chips(COOL Chips VIII)*, April 2005.
- [14] H. Holma and A. Toskala. *WCDMA for UMTS: Radio Access For Third Generation Mobile Communications*. John Wiley and Sons, LTD, New York, New York, 2001.
- [15] H. C. Hunter and J. H. Moreno. A New Look at Exploiting Data Parallelism in Embedded System. In *Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 159–169, 2003.
- [16] C. Kozyrakis and D. Patterson. Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 283–293, Nov. 2002.
- [17] C. Kozyrakis and D. Patterson. Overcoming the Limitations of Conventional Vector Processors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 399–409, 2003.
- [18] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 330–335, 1997.
- [19] H. Lee et al. Software Defined Radio - A High Performance Embedded Challenge. In *Proc. 2005 Intl. Conference on High Performance Embedded Architectures and Compilers (HiPEAC)*, Nov. 2005.
- [20] A. Lodi et al. XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module. In *IEEE Journal of Solid-State Circuits*, volume 41, No. 1, pages 85–96, Jan. 2006.
- [21] Y. Neuvo. Cellular Phones as Embedded Systems. In *IEEE International Solid-State Circuits Conference*, 2004.
- [22] A. Peleg and U. Weiser. MMX Technology Extension to the Intel Architecture. In *IEEE Micro*, volume 16, no. 4, Aug. 1996.
- [23] S. Rixner et al. Register Organization for Media Processing. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 375–386, Jan. 2000.
- [24] H. Stone. Parallel Processing with the Perfect Shuffle. In *IEEE Transactions on Computers*, volume 20, Feb.
- [25] C. van Berkel et al. Vector Processing as an Enabler for Software-Defined Radio in Handsets From 3G+WLAN Onwards. In *Proc. 2004 Software Defined Radio Technical Conference*, Nov. 2004.
- [26] A. Waksman. A Permutation Network. In *Journal of the ACM*, volume 15, No. 1, pages 159–163, Jan. 1968.