A Low-Power DSP for Wireless Communications

Hyunseok Lee, *Member, IEEE*, Chaitali Chakrabarti, *Senior Member, IEEE*, and Trevor Mudge, *Fellow, IEEE*

***Abstract*—This paper proposes a low-power high-throughput digital signal processor (DSP) for baseband processing in wireless terminals. It builds on our earlier architecture—Signal processing On Demand Architecture (SODA)—which is a four-processor, 32-lane SIMD machine that was optimized for WCDMA 2 Mbps and IEEE 802.11a. SODA has several shortcomings including large register file power, wasted cycles for data alignment, etc., and cannot satisfy the higher throughput and lower power requirements of emerging standards. We propose SODA-II, which addresses these problems by deploying the following schemes: operation chaining, pipelined execution of SIMD units, staggered memory access, and multicycling of computation units. Operation chaining involves chaining the primitive instructions, thereby eliminating unnecessary register file accesses and saving power. Pipelined execution of the vector instructions through the SIMD units improves the system throughput. Staggered execution of computation units helps simplify the data alignment networks. It is implemented in conjunction with multicycling so that the computation units are busy most of the time. The proposed architecture is evaluated with an in-house architecture emulator which uses component-level area and power models built with Synopsys and Artisan tools. Our results show that for WCDMA 2 Mbps, the proposed architecture uses two processors and consumes only 120 mW while SODA uses four processors and consumes 210 mW when implemented in 0.13-μm technology and clocked at 300 MHz.**

***Index Terms*—Baseband processor, digital signal processing (DP), low power, programmable, SIMD, software-defined radio (SDR).**

I. Introduction

RECENT years have seen an emergence of a large number of protocols that cater to different types of wireless communication networks. In these protocols, baseband processing is one of the most computationally demanding parts and is usually realized with ASICs for power efficiency. But ASIC-based hardwired solutions are costly because of the large number of existing and upcoming wireless standards. Software-defined radio (SDR) provides a cost-effective and flexible solution for implementing multiple wireless protocols in software. In this paper, we present a programmable, high-throughput, low-power processor for baseband processing. Designing such a processor is challenging, because programmability is usually achieved at the cost of energy efficiency.

Several programmable architectures have been proposed for baseband processing that achieve high energy efficiency by exploiting the inherent parallelism in many of the algorithms. Data-level parallelism (DLP) has been exploited efficiently in a number of the existing SIMD-based architectures such as Infineon's MUSIC [1], Analog Devices' TigerSHARC [2], Icera's DXP [3], NXP's EVP [4], SandBridge's SandBlaster [5], Linköping's SIMT [6], and Michigan's SODA [7]. There also exists some degree of thread-level parallelism (TLP) which has been supported by SandBlaster [5], SIMT [6], and SODA [7]. In addition, there are heterogeneous SDR architectures that support fine-grained parallelism such as PicoArray [8] and XiRisc [9], or coarse-grain parallelism such as Intel RCA [10], QuickSilver [11], Montium [12], and ADRES [13]. In this paper we present a programmable baseband processor, SODA-II, that takes advantage of the inherent parallelism in novel ways to further increase energy efficiency for current and emerging standards.

Previously we developed a low-power programmable multiprocessor architecture, Signal processing On Demand Architecture (SODA) [7], for supporting wireless baseband processing protocols such as W-CDMA 2 Mbps [14] and IEEE 802.11a [15]. Each processor consists of a 16-bit and 32-lane SIMD datapath to handle the vector computations, a scalar datapath, scratch pad memories, address generation unit, and DMA support. The main drawbacks of SODA are that it spends a significant amount of time on noncomputational operations such as realigning data through a shuffle network and performing a large number of register file accesses. This is quite typical of wide SIMD architectures, and similar observations have been made in [16] and [17].

In order to design the next generation of low-power signal processors with higher throughput and lower power, we need to reevaluate the interaction between algorithms and architecture. Our analysis of the workload characteristics of the wireless protocols revealed that the majority of baseband processing is implemented by the following vector algorithm kernels: finite impulse response (FIR) filters, pattern matching, minimum/maximum finding, fast Fourier transforms (FFT), and the Viterbi operations of branch metric computation (BMC) and add-compare-select (ACS). These kernels can be further decomposed into five primitive vector operations of data load, vector alignment, vector computation, vector reduction, and data store. These characteristics guided the design of the proposed architecture, SODA-II. While SODA-II retained the multiprocessor wide SIMD architecture of SODA, it significantly improved upon the performance and energy efficiency of SODA by employing the following schemes, namely, operation chaining, pipelined execution of SIMD units, staggered

Manuscript received September 18, 2008; revised May 14, 2009; accepted May 14, 2009. First published October 09, 2009; current version published August 25, 2010.

H. Lee is with the Department of Electronics and Communications Engineering, Kwangwoon University, 139-701 Seoul, Korea (e-mail: hyunseok@kw.ac.kr).

C. Chakrabarti is with the Department of Electrical Engineering, Arizona State University, Tempe, AZ 85287-5706 USA (e-mail: chaitali@asu.edu).

T. Mudge is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48105-2122 USA (e-mail: tnm@eecs.umich.edu).

Digital Object Identifier 10.1109/TVLSI.2009.2023547

TABLE I
MAJOR ALGORITHM KERNELS IN THE BASEBAND PROCESSING WORKLOAD

Key Kernels	Vector/Scalar	Vector Width	System Type	Function Block
FIR filter	vector	6-320	TDMA,CDMA,OFDMA	pulse shaper, channel estimator
Pattern matching	vector	16	CDMA	synchronization
min/max finding	vector	32-10248	TDMA,CDMA,OFDMA	channel decoder
Viterbi-BMC/ACS	vector	64-256	TDMA,CDMA,OFDMA	channel decoder, channel estimator
FFT	vector	64-2048	OFDMA	demodulation
Viterbi-TB	scalar	-	TDMA,CDMA,OFDMA	channel decoder, channel estimator
Interleaving	scalar	-	TDMA,CDMA,OFDMA	interleaver, deinterleaver, channel decoder
Symbol mapping	scalar	-	TDMA,CDMA,OFDMA	modulator, demodulator
Channel encoding	scalar	-	TDMA,CDMA,OFDMA	channel encoder
Sliding window	scalar	-	TDMA,CDMA,OFDMA	frame detection
Code generation	scalar	-	CDMA	modulator, demodulator
Interpolation	scalar	-	OFDMA	demodulator
Frequency tracking	scalar	-	OFDMA	demodulator

memory access, and multicycling in computation units. Operation chaining involves chaining the primitive instructions and improves the throughput and power performance by reducing the number of register file accesses. Pipelined execution of SIMD units supports parallel execution of vector operations and increases the system throughput. It is efficient for vector algorithm kernels with simple data alignment patterns such as FIR filter, pattern matching, and min/max finding. For algorithms that have complex data alignment and require multiple cycles for computation such as FFT and Viterbi, a combination of staggered memory access and multicycling of the computation units results in lower power and higher throughput.

To evaluate the effectiveness of these schemes, we implemented a hardware model of SODA-II. At the component level, we used Verilog and Synopsys' Physical Compiler for characterization and used PrimePower to generate power estimates. The component-level information was used in an architecture emulation program to generate kernel-level and system-level power estimates. We compared the performance of the proposed architecture with the original SODA [7], termed the reference architecture in this paper. We found that operation chaining and pipelined execution of SIMD units improved the system throughput the most, and operation chaining and staggered memory accesses enhanced the system energy efficiency the most. As an application example, we showed that for WCDMA 2-Mbps packet service, the proposed architecture requires two processors (PEs) and consumes an estimated 120 mW compared to SODA which requires four PEs and consumes an estimated 210 mW when implemented in 0.13-μm technology and clocked at 300 MHz.

The rest of the paper is organized as follows. Section II briefly describes the key characteristics of the wireless protocols. Section III describes the limitations of SODA and then proposes three architectural schemes to improve its performance. Section IV describes the details of the SODA-II architecture. Section V provides an analysis of its power and throughput performance. Section VI includes a brief survey of existing baseband processor architectures. Section VII concludes with some remarks.

II. WORKLOAD CHARACTERIZATION

There are three main categories of wireless communication systems. These are based on time division multiple access (TDMA), code division multiple access (CDMA), and orthogonal frequency division multiple access (OFDMA). Despite the differences in their mode of operation, baseband processing shows many common characteristics: 1) the number of major algorithm kernels is few; 2) baseband operation is a mixture of vector and sequential algorithm kernels; 3) vector algorithm kernels dominate baseband workload; and 4) vector algorithm kernels can be decomposed into five primitive vector operations. We elaborate on these characteristics and show how they are exploited in the development of the proposed DSP architecture.

TABLE II
DISTRIBUTION OF WORKLOAD FOR THE 2-Mbps WCDMA PACKET DATA SERVICE. THE VECTOR ALGORITHMS CONSTITUTE MORE THAN 90% OF THE TOTAL WORKLOAD

FIR filter	Viterbi BMC/ACS	Min/Max Finding	Scalar workload
66.7%	27.7%	0.03%	5.6%

Table I shows that the majority of baseband processing can be modeled as a combination of five vector algorithm kernels and eight scalar algorithm kernels. The five vector kernels are FIR filtering, pattern matching, min/max finding, Viterbi-BMC/ACS, and FFT, and the eight scalar kernels are Viterbi-TB (trace-back), interleaving, symbol mapping, channel encoding, sliding window, code generation, interpolation, and frequency tracking.

An analysis of the wireless protocols shows that the vector kernels account for more than 90% of the baseband processing workload of protocols based on TDMA, CDMA, or OFDMA. Thus, the power efficiency and throughput of a baseband processor is determined by how efficiently the vector kernels can be computed. As an example, Table II shows the distribution of workload for the 2-Mbps WCDMA packet data service before parallelizing. To avoid diverging from the main theme of the paper, the detailed workload characterization for the 2-Mbps WCDMA packet data service has been included in the Appendix.

Further analysis showed that the vector algorithm kernels can be decomposed into five vector operations: data load, vector alignment, computation, vector reduction, and data store. Table III summarizes the decomposition of the five vector kernels. Data load involves reading input operands from memory; vector alignment shuffles or permutes the loaded

TABLE III
DECOMPOSITION OF MAJOR VECTOR ALGORITHM KERNELS INTO FIVE VECTOR OPERATIONS

Vector Kernels	Load	Alignment	Computation	Reduction	Store
FIR Filter	scalar	vector shift	conditional complement, real mult.	summation	scalar
Pattern Matching	single-vector	no shuffle	conditional complement, real mult.	summation	scalar
Min/Max Finding	multi-vector	no shuffle	compare-and-select	finding min or max	scalar
Viterbi-BMC/ACS	multi-vector	Viterbi trellis	add, sub, compare, conditional comp.	-	vector
FFT	multi-vector	FFT butterfly	add, complement, complex mult.	-	vector

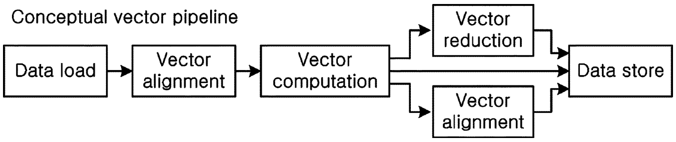


Fig. 1. Pipeline of vector operations in baseband processing—a conceptual diagram.

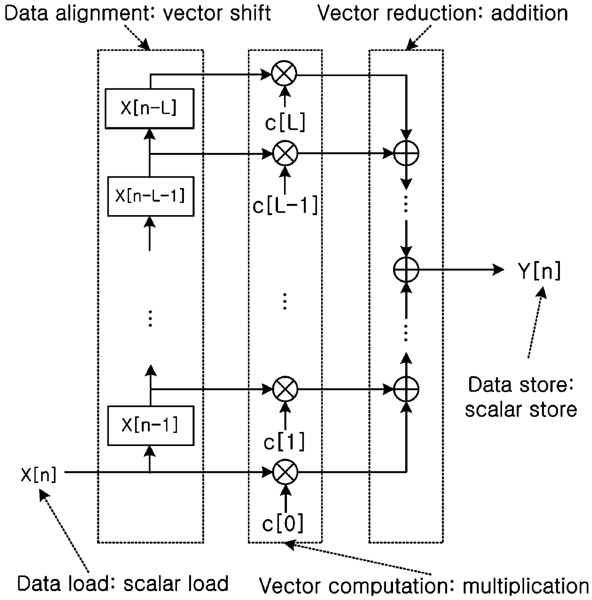


Fig. 2. Decomposition of FIR kernel $Y[n] = \sum_{i=0}^{L} X[n-i]c[i]$ into scalar load, vector shift, multiplication, summation, and scalar store.

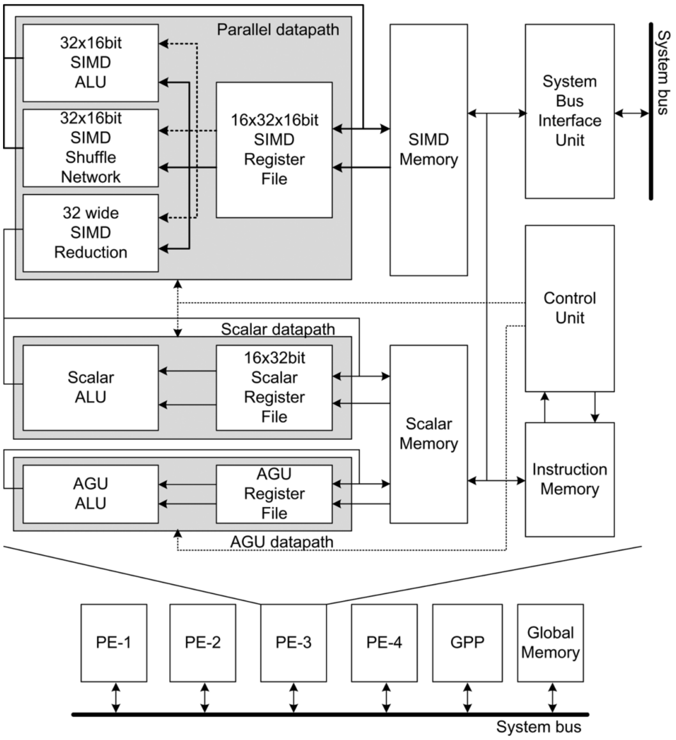


Fig. 3. High-level description of SODA.

data before or after vector computation; vector computation performs arithmetic or logical operations in parallel; vector reduction converts the output of the vector computation into a scalar result; data store saves the results of a vector computation or vector reduction to data memory.

The five vector operations can be used to construct a vector pipeline, the conceptual diagram is shown in Fig. 1. Fig. 2 illustrates the decomposition for the FIR filter example. This figure shows that the FIR filter can be formed by pipelining data load (which corresponds to the scalar input $X[n]$ being input), vector alignment (which corresponds to a vector shift), multiplication (which corresponds to vector computation), addition (which corresponds to vector reduction), and finally data store (which corresponds to storing the scalar output $Y[n]$). This type of decomposition is also possible for other vector kernels listed in Table III. Note, that in this pipeline, data dependencies between vector operations are unidirectional. Since there is sufficient vector workload to keep the pipeline busy for several hundred cycles, all of the vector kernels can be mapped onto a hardware vector pipeline with only occasional reconfiguration.

III. EVOLUTION FROM SODA TO SODA-II

Earlier we had proposed a wide SIMD architecture, SODA [7], that fully exploited the vector parallelism of baseband processing. While SODA satisfied the throughput requirements of 2G and 3G terminals, and was indeed a low-power solution (450 mW), it did not have the capability to support higher throughput applications. In this section, we first describe SODA (Section III-A), its limitations (Section III-B), and then introduce three architectural schemes to improve SODA performance significantly (Section III-C).

A. *Previous Work: SODA*

The SODA architecture is shown in Fig. 3. It is a chip multiprocessor architecture with four PEs, one general-purpose processor (GPP), and a shared global scratchpad memory. Each PE is sufficiently large and implements an entire kernel algorithm such as FIR or Viterbi. Such a mapping results in low traffic between PEs, and a low-speed bus is enough to support inter-PE communication. The GPP controls the four PEs.

TABLE IV
POWER PROFILE OF A SODA PE WHEN IT PERFORMS VITERBI DECODING WITH $K = 7, R = 1/3$, AND FFT OF 64 POINTS

	D-mem.	V-reg.	V-shuffle	V-ALU	control
Viterbi	7.45%	**70.66%**	1.19%	9.77%	10.91%
FFT	17.68%	**57.62%**	4.24%	9.6%	6.3%

TABLE V
OPERATION CYCLE PROFILE OF A SODA PE WHEN IT PERFORMS VITERBI DECODING WITH $K = 7, R = 1/3$, AND FFT OF 64 POINTS

	load/store	alignment	computation	control
Viterbi	37.2%	4.8%	**48.0%**	9.9%
FFT	24.3%	25.2%	**43.7%**	6.8%

Each PE in SODA consists of six units: parallel datapath, scalar datapath, instruction memory, data memory, control unit, and system bus interface unit. The parallel datapath executes the vector operations, and the scalar datapath executes the sequential operations. The control unit consists of an instruction decoder, interrupt handler, and program counter. It controls the scalar datapath and the parallel datapath. The system bus interface unit manages the communication between PEs.

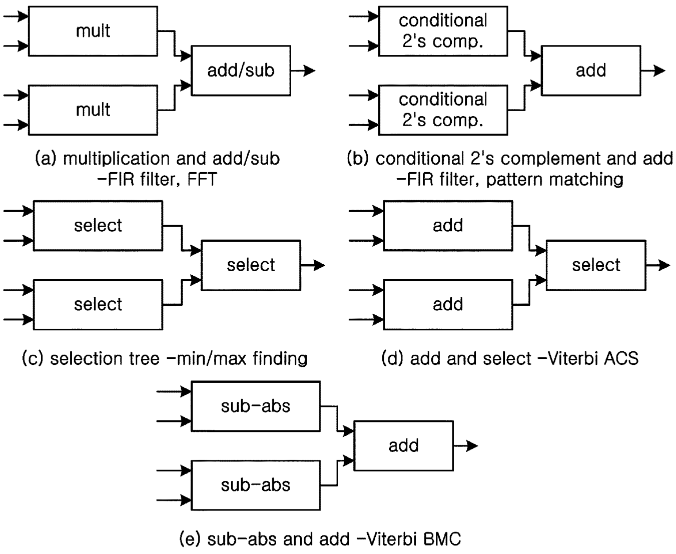
The parallel datapath of a SODA PE consists of an SIMD ALU, SIMD register file, SIMD shuffle network, and an SIMD reduction unit. The SIMD ALU has 32 lanes, where each lane consists of a 16-bit datapath and performs vector computations. The SIMD register file provides input vector operands to the SIMD ALU, shuffle network, and reduction unit. It also stores temporary computation results like a conventional register file. The SIMD shuffle network performs data alignment that is required to support data movement operation between computations. It is implemented by a multiple stage shuffle exchange network. The SIMD reduction logic converts vector data into a scalar value. For example, it is used for finding the maximum of 32 numbers or adding 32 numbers.

SODA is based on a RISC style instruction set where each instruction describes a primitive operation: data load/store from/to memory, and simple arithmetic operations such as addition, multiplication and comparison. An instruction can describe the operation in only one unit—either an SIMD ALU, or an SIMD shuffle network, or an SIMD reduction unit. While this reduces the length of the instruction, it results in significant increase in the number of cycles as well as register power consumption.

B. Limitations of SODA

The SODA architecture has some limitations. First, $\sim$ 60–70% of the SODA parallel datapath power is consumed by the SIMD register file (denoted by V-reg) as shown in Table IV. This is because the number of accesses to register files is high in the RISC-style of operation adopted by SODA—each instruction loads two input operands from the register file and stores one operation result on the register file.

Second, even though the vector operations that constitute a vector kernel can be computed in a pipelined fashion (as shown in Fig. 1), they are executed as a sequence of RISC instructions in SODA. As a result, SODA spends a substantial portion of its operation time on noncomputational operations such data load/store and vector alignment. Table V shows that SODA spends $\sim$40–50% of its operation time on noncomputational operations while executing Viterbi and FFT.



(a) multiplication and add/sub –FIR filter, FFT

(b) conditional 2's complement and add –FIR filter, pattern matching

(c) selection tree –min/max finding

(d) add and select –Viterbi ACS

(e) sub-abs and add –Viterbi BMC

Fig. 4. Operations in major vector kernels which can be chained by concatenating arithmetic units.

In Section III-C, we describe techniques that have been adopted in SODA-II to reduce the time spent in noncomputation operations and to reduce the number of register file accesses.

C. Performance Enhancement Schemes in SODA-II

SODA-II enhances the performance of SODA by implementing operation chaining (Section III-C1), pipelined execution of SIMD units (Section IIIC2), staggered memory access, and multicycling (Section III-C3).

1) Operation Chaining: Operation chaining is a technique where a set of primitive operations linked by unidirectional dependencies are computed one after the other without writing intermediate results to a register file. It is implemented by concatenating arithmetic units so that the result of one arithmetic unit can be directly forwarded to the next one without accessing the register file. In fact, this technique eliminates about $\sim$40–50% of register file accesses and helps reduce the high register file power consumption problem of SODA. Operation chaining is applicable to many of the major kernels listed in Table III. The five frequent computation patterns which can be chained are shown in Fig. 4. These can be mapped onto hardware that consists of two parallel arithmetic units (first stage) which feed into a third arithmetic unit (second stage). This is the basic structure of the SIMD computation unit (CU) that is described in Section IV-B and illustrated in Fig. 8. The concatenation of arithmetic units results in a reduced cycle count and higher throughput. All vector kernels benefit by operation chaining; the amount of performance improvement varies according to the characteristics of the vector kernel.

Table VI lists the instructions that are used for implementing the five vector kernels. Note that 9 out of the 11 instructions are chained instructions. For instance, instruction "DBMCA" is obtained by chaining two subtract/absolute value operations

TABLE VI
CHAINED AND PRIMITIVE INSTRUCTIONS USED FOR IMPLEMENTING THE VECTOR KERNELS

Chained Instructions	
Instruction	**Description**
DCCAD RD,R1,R2,R3,R4	Double conditional complements and addition : T_1 = R1 ? R2:-R2; T_2 = R3 ? R4:-R4; RD = T_1+T_2
DMLAD RD,R1,R2,R3,R4	Double multiplications and addition: T_1 = $R1\times R2$; T_2 = $R3\times R4$; RD = T_1+T_2
DMLSB RD,R1,R2,R3,R4	Double multiplications and subtraction : T_1 = $R1\times R2$; T_2 = $R3\times R4$; RD = T_1-T_2
TMIN RD,R1,R2,R3,R4	Triple min. value search: T_1 = (R2>R1) ? R1:R2; T_2 = (R4>R3) ? R3:R4; RD = (T_2>T_1) ? T_1:T_2
TMAN RD,R1,R2,R3,R4	Triple max. value search: T_1 = (R1>R2) ? R1:R2; T_2 = (R3>R4) ? R3:R4; RD = (T_1>T_2) ? T_1:T_2
DBMCA RD,R1,R2,R3,R4	Double branch metric computations and addition: T_1 = abs(R1-R2); T_2 = abs(R3-R4); RD = T_1+T_2
BMCA RD,R1,R2,R3	Branch metric computation and addition: T_1 = abs(R1-R2); RD = T_1+R3
BMC RD,R1,R2	Branch metric computation, RD = abs(R1-R2)
ACS RD,R1,R2,R3,R4	Add, compare, and select: T_1=R1+R2; T_2=R3+R4; RD=(T_1>T_2) ? T_1:T_2
Primitive Instructions	
Instruction	**Description**
ADD RD,R1,R2	Addition, RD = R1+R2
SUB RD,R1,R2	Subtraction, RD = R1-R2

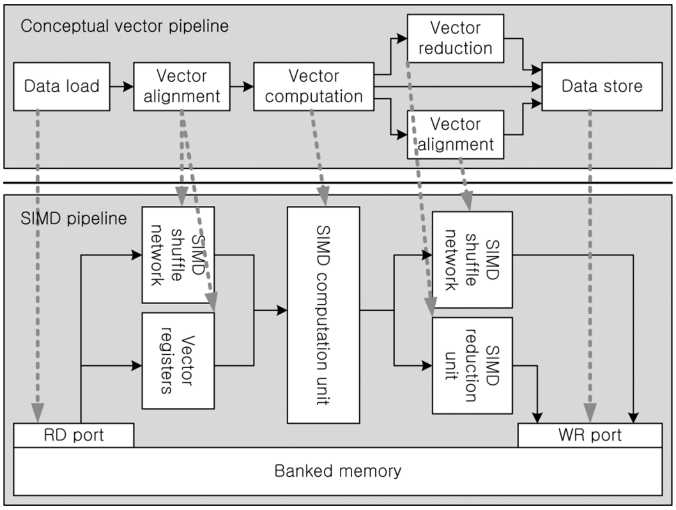


Fig. 5. Mapping of conceptual vector pipeline onto a pipeline of SIMD execution units.

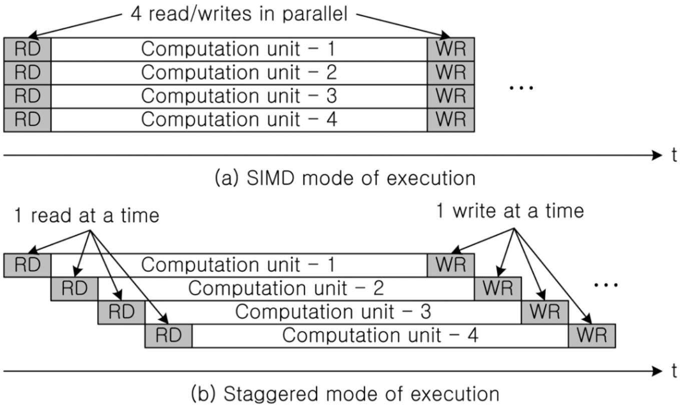


Fig. 6. Execution of computation units using (a) SIMD mode and (b) staggered mode.

followed by an addition corresponding to the structure shown in Fig. 4(e).

2) Pipelined Execution of SIMD Units: In baseband processing, noncomputational operations such as data load/store and vector alignment occupy a substantial portion of the total cycle count (see Table V). So significant gain can be achieved if the vector operations can be executed in a pipelined fashion resulting in reduced number of load/store and alignment operations.

Fig. 5 shows the mapping of the vector operations onto a pipeline of SIMD execution units. The data dependencies between the vector operations are unidirectional, and so there is no complex data forwarding between the corresponding hardware units. The net result is a significant reduction in the number of data load/store instructions. For this scheme to be effective, the pipeline has to be highly utilized. Our workload analysis shows that the vector kernel algorithms keep the pipeline busy for long periods of time, resulting in significant performance enhancement.

The pipelined execution scheme is certainly an efficient way of implementing vector kernels with simple data alignment patterns such as FIR filtering, min/max finding, etc. However, its application to vector kernels with complex data alignment and multicycle computations (such as Viterbi and FFT) results in function unit underutilization and the use of power-hungry data alignment networks. In Section III-C3, we describe how to alleviate these problems by using a combination of staggered execution of computation units and multicycling.

3) Staggered Memory Access and Multicycling in Computation Units: As mentioned earlier, data alignment is fairly simple for FIR filtering, pattern matching, and min/max finding. For instance, for FIR filtering, one scalar element can be read from the data memory and shifted into the vector registers. As a result, a vector load from data memory can be avoided. For kernels with complex data alignment such as Viterbi and FFT, vector load can be avoided by distributing data in multiple memory banks. However the data have to be aligned using $N \times N$ switches before the data can be fed to the computation units. In an architecture that supports operation chaining, four sets of $N \times N$ switches would be required, resulting in a fairly large area overhead.

Now, if the operations in the SIMD computation unit could be staggered, as in time division multiplexing, then each $N \times N$ switch could be replaced with a single $N \times 1$ switch. The data alignment problem could now be handled by the memory address generators. Fig. 6 illustrates the SIMD mode and staggered mode of operation in a datapath with four computation units. For more details, please refer to [18].

One potential problem with such a design is that in many cases, the computation unit is underutilized. If the architecture

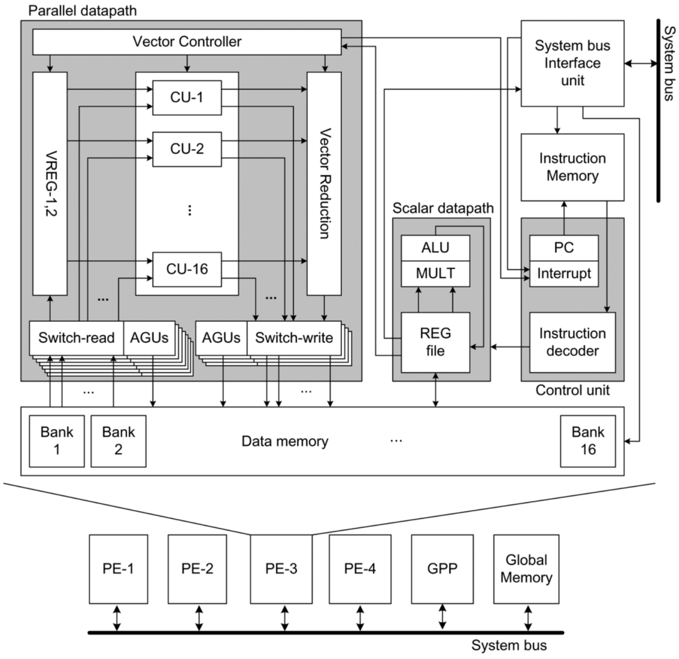


Fig. 7. Architecture of the proposed SODA-II processor.

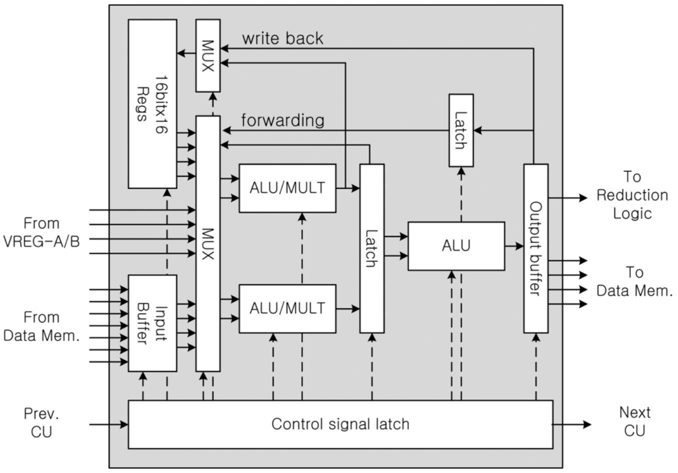


Fig. 8. Architecture of the computation unit.

has 16 computation units, then the computation unit has to have a delay of 16 cycles to be 100% utilized. To keep the computation unit busy for 16 cycles most of the time, it has to be assigned a more complex workload. This is exactly what is done in SODA II. The computation unit now behaves like a small microengine. It reads in multiple data and operates on them over multiple cycles—the process is referred to as *multicycling*. As a consequence, the number of read and write switches must be increased. For instance, for Viterbi and FFT, seven inputs have to be fed to the computation unit to keep it utilized all the time, and the number of read switches increases from four to seven. However all these switches are much simpler than the $N \times N$ switches. For instance, the seven $N \times 1$ switches consume 1/12 of the area and 1/2 of the power compared to the four $N \times N$ switches.

IV. Proposed Architecture: SODA-II

A. High-Level Architecture of SODA-II

Fig. 7 shows the proposed architecture, SODA-II. Like SODA [7], it consists of four PEs, one GPP, and one global memory. Each PE consists of a parallel datapath, a scalar datapath, data memory, instruction memory, control unit, and system bus interface unit. While at the high-level, SODA-II is similar to SODA, it differs from SODA in the way the kernel operations are implemented in the parallel datapath. Specifically, it incorporates operation chaining and pipelined execution of SIMD units to enhance the performance of all kernels. In addition, it implements staggered execution and multicycling to enhance the performance of Viterbi- and FFT-type operations.

B. Detailed Architecture of Parallel Datapath Components

The parallel datapath of a PE consists of computation units, address generators and switches for data read and data write, vector registers, reduction logic, and vector controller (see Fig. 7). The scalar datapath is similar to the SODA scalar datapath and is not described here.

1) Computation Units: The computation unit is like a small microengine that is capable of supporting single-cycle operations such as FIR and multicycle operations such as Viterbi and FFT. The CU supports operation chaining. There are 16 CUs; the number 16 was chosen after conducting a detailed analysis of hardware cost, including the cost of the switches, energy, and throughput of all the kernels [18]. We found that in 0.13-μm technology, the architecture sweet spot was 16 when optimized for throughput/watts.

The detailed architecture of the computation unit is shown in Fig. 8. It consists of three arithmetic units, an input buffer, an output buffer, register file, multiplexer, and latch. The arithmetic units are connected in a way to support the computation patterns of the vector kernels depicted in Fig. 4. For additional pipelining, the outputs of the first stage are latched. The outputs can be written into the register file. To support efficient multicycling, the outputs can also be forwarded to the ALUs via the data-forwarding paths.

The multiplexer reads input operands from three different sources: the input buffer, vector registers, and register file. In the SIMD mode (which is employed for the FIR-type of operations), the input operands come from the vector registers. In the staggered mode (which is employed for the FFT- or Viterbi-type of operations), the intermediate outputs are cycled back for further computation. The inputs to the multiplexer then come from the input buffer and register file. Furthermore, in the staggered mode, the control signals of one CU are sent to the next one in the following cycle, thereby reducing the controller complexity.

2) Vector Reduction Unit: The vector reduction unit converts the vector data generated by the computation units to scalar data. There are only two kinds of reduction operations, summation and minimum/maximum value searching. Summation can be represented by $y = \sum_{i=0}^{N-1} x_i$, where the input vector $X = (x_0, \ldots, x_{(N-1)})$ and the scalar output is y. Minimum/maximum value searching can be represented by $y = \min_{i=0}^{N-1} \; x_i$. Both $\min(\cdot)$ and $\max(\cdot)$ functions are based on subtraction and selection. Thus, the summation and minimum/maximum value

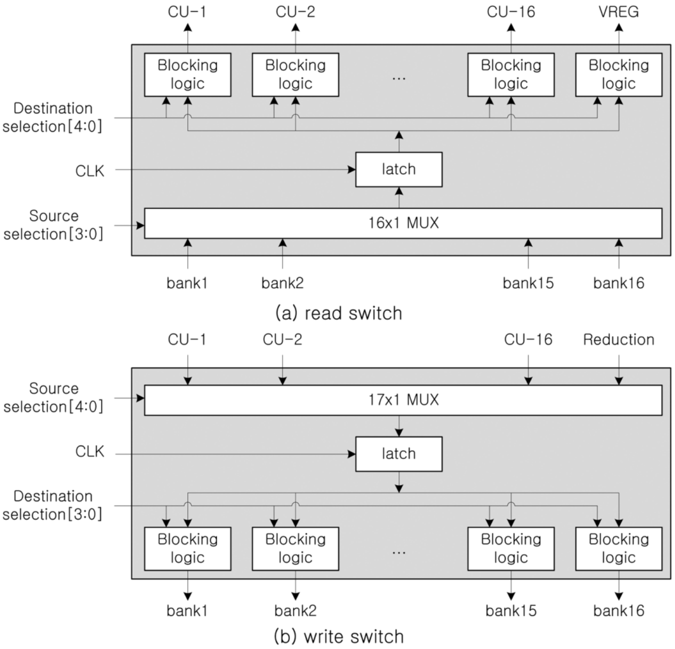


Fig. 9. Block diagram of (a) read switch and (b) write switch.

searching operations can easily be mapped onto the same hardware with only minor modifications. The reduction logic is organized as a $\log_2 N$ depth tree.

3) Switches for Data Read/Write: The role of the switches is to route data between data memory banks and the computation units. In the proposed architecture, we have 11 switches, seven for read and four for write. The large number was necessary to support multicycling for Viterbi- and FFT-type of operations. As shown in Fig. 9(a), each switch for data read consists of a 16×1 multiplexer, a latch, and 17 blocking logic units. The 16×1 multiplexer is used to select one data from 16 memory subbanks. The output of the multiplexer is routed through a latch to blocking logic units placed in front of the 16 CUs and one vector register. The blocking logic helps in reducing glitch propagation; the latch allows read and write pipelining. The switch for a data write is similar to the switch for a data read and shown in Fig. 9(b). It consists of 17×1 multiplexer (16 input ports for CUs and one input port for vector reduction), a latch, and 16 blocking logic units. In order to provide multiple data to the CU, several switches are active at a time.

4) Address Generators: Address generators (AGU) are used to automatically generate the next address of the data memory (see Fig. 7). One address generator is designated per read/write switch, and so there are 11 address generators in the parallel datapath. Because of the deterministic address generation patterns of the vector kernels, hardware required for automatic address generation is simple. There are two address generation modes, linear and paged. In the linear mode, the next data memory address is given by a linear equation, $y[n] = \alpha + \beta \cdot n$, where α is an initial address and β is the difference between two adjacent data points. The linear operation mode is used for FIR filter and Viterbi-BMC/ACS.

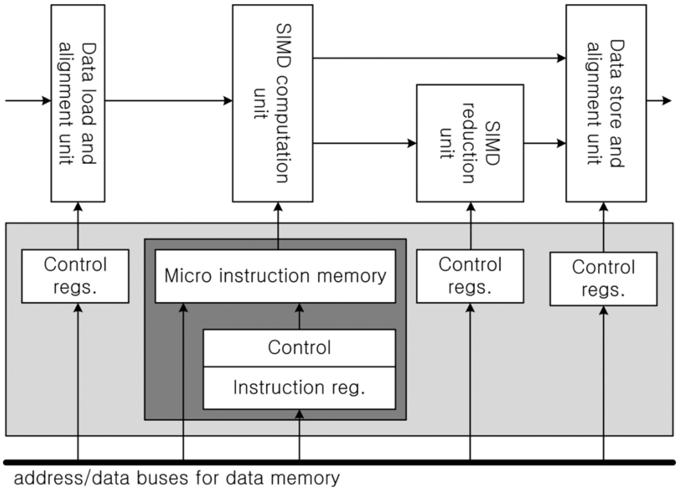


Fig. 10. Control scheme of the parallel datapath.

The address generation pattern in the paged mode can be represented by the following equation:

$$y[n] = \begin{cases} y[n-1] + \alpha, & \text{if } (n \bmod M) \neq M-1 \\ y[n-1] + \beta, & \text{otherwise} \end{cases} \tag{1}$$

where M is the number of data elements in a page, α is the displacement within a page, and β is the displacement when entering a new page. The paged mode is used for data write in the Viterbi-BMC/ACS and FFT kernels.

5) SIMD Control Logic: To support pipelined execution through the SIMD units, an efficient controller has to be designed. We make use of control registers, which are mapped onto data memory space, for controlling the data load/store, vector alignment, and vector reduction units. Fig. 10 depicts the overall control scheme. The control of each of the individual 16 CUs uses the same control registers. In staggered mode each CU views a shifted version of the same control registers. We refer to this as streaming control.

Control of the computation units is more involved since these units support multicycling. A complex instruction set computer (CISC)-style control scheme is utilized here. Five CISC instructions are defined in order to describe the operation of the vector kernels. The instructions shown in Table VI are used for building the microcode of these CISC instructions.

V. Power and Throughput Analysis

A. Experimental Environment and Methodology

The dynamic power and throughput of the proposed architecture is estimated at the component, kernel, and system levels. For component-level estimation, a hardware model is built in Verilog and synthesized using Synopsys' Physical compiler. TSMC-13 standard cell library based on 0.13-μm technology is used. To reduce dynamic power, a low voltage (1 V) and low dielectric constant library are used. Wire capacitance for interconnection-dominated modules such as switches is considered by generating the layout using Cadence's Silicon ensemble.

TABLE VII
COMPONENT-LEVEL BREAKDOWN OF THE POWER CONSUMPTION AND AREA OF SODA-II. THE AVERAGE POWER IS FOR THE WCDMA 2-MbpsPACKET DATA SERVICE WHEN ONLY TWO PEs ARE ACTIVE. THE LEAKAGE POWER AND AREA CORRESPOND TO THE FOUR-PROCESSOR ARCHITECTURE

		Average power (mW)	**Leakage power (mW)**	**Area** (mm^2)
Memory	Data mem.	13.8	1.41	3.7
	R/W switch	17.4	0.18	0.1
	Addr. gen.	7.4	0.10	0.1
Vector CU	REGs	7.0	0.25	0.4
	MULTs	5.5	0.42	0.5
	ALUs	8.6	0.43	0.4
	MUX	2.2	0.10	0.0
Vector registers		0.4	0.05	0.0
Vector reduction		0.4	0.04	0.2
Vector control		6.6	0.01	0.2
SCALAR	REG	2.4	0.06	0.1
	MULT	2.4	0.04	0.1
	ALU	1.8	0.00	0.1
	Control	1.1	0.01	0.1
	I-MEM	12.0	0.70	1.8
BUS I/F		0.4	0.10	0.1
Miscellaneous		34.3	-	2.4
Total(rounded)		120	4	11

Additionally, Artisan's memory compiler is used for the generation of storage components. The memory compiler provides HDL models which are used for generating the timing and power information. The gate-level dynamic power is evaluated using Synopsys' PrimePower. Overall, the component-level power estimates are fairly accurate.

The kernel-level power estimates are obtained by using the component-level models in an architecture emulation program that was developed in house. The emulation program calculates the activity level of each component during the execution of the assembly program. This is used to estimate the cycle count and power consumption of each kernel. In order to consider the effect of global clocking and interconnection, we assume 30% power and area overhead. Finally, the system-level throughput and power estimate is calculated by aggregating kernel-level estimation. We expect our estimates to be within 20% of those obtained by fully synthesized designs.

All comparisons are with respect to the reference architecture, SODA. Since SODA was implemented in 0.18-μm technology, operated at 1.8 V, and was clocked at 400 MHz, it was resynthesized using the same parameters as the proposed architecture, namely, 0.13-μm technology, 1 V, low-K material, and 300-MHz operation frequency. The operation frequency of 300 MHz was chosen by evaluating the energy/cycle versus operation frequency (in the range 100–800 MHz) for WCDMA 2 Mbps and then choosing the frequency with the lowest energy/cycle.

B. Component-Level Analysis of SODA-II

1) Power: Table VII shows the component-level power breakdown of SODA-II. The average power is the power consumed while executing WCDMA 2-Mbps packet data service by only two PEs; the other two PEs are inactive. From Table VII, we see that the data memory and instruction memory are the most power consuming blocks in a PE. Thus, use of low-power memory will be essential for further power reduction. The power consumption of the registers inside each CU is 7.0 mW, which is only 80% of the ALU power. This is significantly small compared to SODA where the register file consumed about ~6–7 times more power than the ALU. Finally, the 30% power overhead due to global clocking and interconnection is presented under "Miscellaneous."

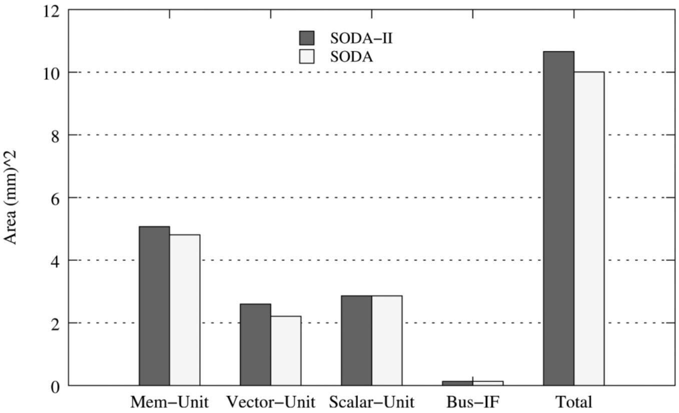


Fig. 11. Area comparison between SODA and SODA-II.

TABLE VIII
COMPARISON OF THE THROUGHPUT AND ENERGY CONSUMPTION OF SODA-II WITH SODA WHEN THEY PERFORM MAJOR VECTOR KERNELS FOR BASEBAND PROCESSING. THE NUMBER OF PEs IN BOTH ARCHITECTURES IS 4

Vector Kernels	**Energy** (pJoul/Output)		
	SODA-II	SODA	Ratio
FIR filter	530	745	71%
Pattern matching	200	430	47%
Min/Max finding	600	900	67%
Viterbi-BMC/ACS	38530	52780	73%
FFT	1090	1380	79%
	Throughput (Output/Sec)		
	SODA-II	SODA	Ratio
FIR filter	2.7E8	0.7E8	400%
Pattern matching	2.7E8	0.7E8	395%
Min/Max finding	6.8E8	5.4E8	125%
Viterbi-BMC/ACS	2.1E6	1.1E6	195%
FFT	1.1E8	4.2E7	255%

2) Area: The four-PE SODA-II architecture occupies 11 mm^2. The data and instruction memories are dominant, occupying about 50% of the total area. The area of the parallel SIMD datapath is about 20%. Thus, a wide datapath does not contribute much to the total area. The impact of the scalar datapath on the total area is negligible.

Fig. 11 compares the area of the two competing four-PE architectures. We see that the memory and vector units of SODA-II are slightly larger than SODA. Overall, the area of SODA-II is about 10% larger than that of SODA.

C. Kernel-Level Analysis

We compare the throughput and energy consumption of SODA-II and SODA while processing five key vector kernels in Table VIII. We use typical operation scenarios. For instance, for FIR filter, we assume 32 taps, down sampling rate of 2, 16-bit filter coefficients, and 16-bit input data, which is the configuration of the WCDMA pulse shaping filter.

TABLE IX
POWER COMPARISON BETWEEN THE REFERENCE ARCHITECTURE (4 PEs) AND THE PROPOSED ARCHITECTURE (2 PEs) FOR WCDMA 2-Mbps PACKET DATA SERVICE

Power consumption [mW]	**Reference (SODA)**	**Proposed) (SODA-II)**
Memory	28.5	40.6
Vector computation	124.1	31.1
Scalar	19.7	19.7
Vector control	35.44	6.7
Bus I/F	0.4	0.4
Total	210	120

Table VIII shows that SODA-II consumes about ∼20–50% less energy than SODA. Most of the energy savings come from the following factors. The first is the use of operation chaining, which minimizes the number of power-consuming register file accesses. The second is the low control overhead because there is no additional decoding after the initial configuration—streaming control is used to coordinate the pipelined execution in each SIMD lane and across lanes. The third is the simplified data alignment where the $N \times N$ switches are replaced with $N \times 1$ switches during staggered mode operation.

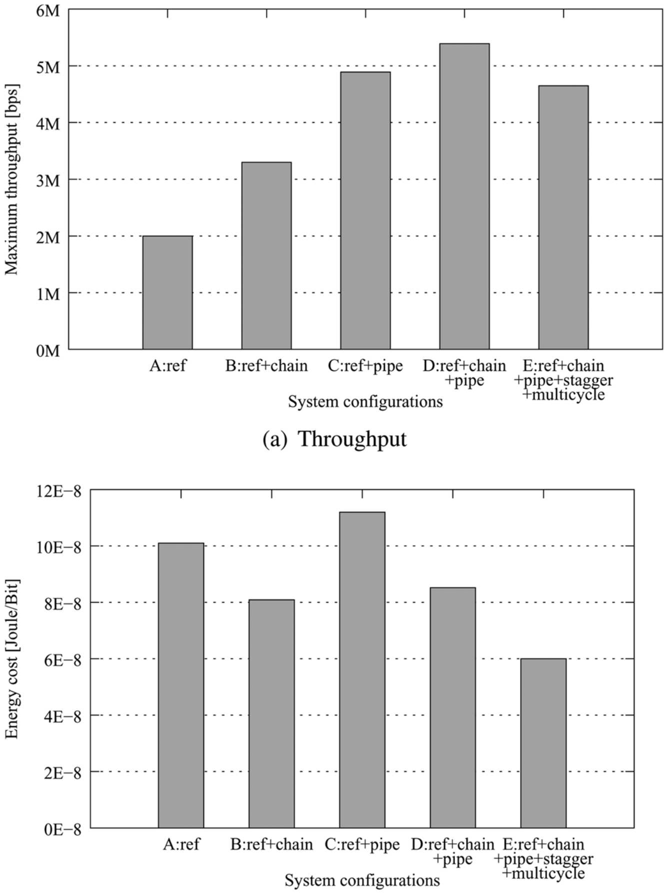
Table VIII also shows that SODA-II processes about ∼1.3–4 times more input data than SODA. This is primarily because of operation chaining and pipelined execution of SIMD units. The performance of the min/max finding kernel is not as impressive. Here the data load takes comparatively more time and does not utilize pipelined execution through the SIMD units effectively.

D. System-Level Analysis: WCDMA 2-Mbps Workload

For processing WCDMA2-Mbps packet data workload, SODA utilizes all four PEs and consumes 210 mW while SODA-II utilizes only two PEs and consumes 120 mW. In comparison, the SB3010 processor from Sandbridge, which is implemented in 90 nm, consumes 600 mW, which is significantly higher compared to both SODA and SODA-II.

Table IX compares the power performance of SODA and SODA-II. The column "memory" represents the power dissipated for data load/store from/to memory. The column "vector computation" represents the power for vector computations that includes the power consumption of arithmetic units and register files. The column "scalar" represents the power consumption for the scalar workload. The column "vector control" represents the power dissipated for vector control signal generation. The two architectures show identical power consumption for system bus interface and scalar workloads.

From Table IX, we see that SODA-II dissipates less power for the vector computations and the vector control signal generation. The reduction in the vector computation power is because of operation chaining. The reduction in the control signal generation power is because control instructions are not decoded in every cycle and the control signals flow in a streaming fashion. While accessing memory, the proposed architecture dissipates more energy than SODA. This is because the data load/store operations are performed through switches which are power consuming. However, the power gains of the vector computation and vector control generation operations more than compensate for this overhead.



(a) Throughput

(b) Energy cost

Fig. 12. Impact of three schemes on system throughput (bits per second) and energy cost (Joules per bit).

E. Impact of Three Schemes

In order to analyze the impact of the three schemes, we studied five system configurations with respect to both maximum throughput and energy cost, represented by energy consumption per bit: A) reference architecture SODA, B) SODA with operation chaining; C) SODA with pipelined execution of SIMD units referred to as SIMD pipelining; D) SODA with operation chaining and SIMD pipelining; and E) SODA with operation chaining, SIMD pipelining, staggered execution and multicycling (SODA-II). For evaluation, we build kernel-level and system-level models corresponding to all configurations. Fig. 12 illustrates the impact of each of these schemes.

We see that operation chaining (configuration B) improves system throughput by about 70% and reduces energy cost by about 20% by decreasing the number of data load/store instructions and thus the number of register file accesses. Pipelined execution of SIMD units (configuration C) increases the system throughput by about 140% because multiple vector operations can now be processed at the same time. Vector kernels with simple data alignment patterns such as FIR filter, pattern matching, min-max can be processed very efficiently because of this enhancement. The energy cost, however, increases by about 15% due to more complex control that is required to manage the pipelined flow during the execution of kernels such as FFT and Viterbi that require multiple cycles.

TABLE X
COMPARISON OF ARCHITECTURAL FEATURES OF BASEBAND PROCESSORS

	SODA-II	SODA	Infineon MuSIC	ADI TigerSHARC	Icera DXP	NXP EVP	Sandbridge Sandblaster	Linköping SIMT
# DSP cores	4	4	4	8	N/A	1	4	1
PE freq. (MHz)	300	400	300	250	1000	300	600	300
# SIMD lanes	32	32	4	2×4	4	16	2×4	4
VLIW on SIMD	no	no	yes	yes	yes	yes	yes	no
Scalar datapath	yes	yes	no	no	no	yes	yes	yes
Coprocessor	no	no	yes	no	no	yes	no	yes
Scratch pad mem.	yes	yes	yes	yes	yes	yes	yes	no
Shared global mem.	yes	yes	yes	no	N/A	no	no	yes

Fig. 12 also shows how a combination of these schemes can enhance system throughput and energy performance. By deploying a combination of operation chaining and pipelined execution of SIMD units (configuration D), the system throughput increases by 170% which is about 10% higher than that of configuration C (SIMD pipelining alone). The energy cost of configuration D drops by about 35% compared to configuration C because of fewer register file accesses due to operation chaining. When staggered memory access and multicycling are added to configuration D, the energy cost is further reduced by 30% (configuration E). This is due to use of $N \times 1$ switches (instead of $N \times N$ switches), fewer register file accesses and streaming-style control scheme. Note that the throughput of configuration E is 10% lower than that of configuration D. This is because the computation units could not be completely utilized during processing of all the kernels.

While throughput is certainly an important performance metric, energy consumption per bit is a more relevant measure for baseband processing, provided performance goals are met. Under this constraint, configuration E, which corresponds to SODA-II, is the best design choice for baseband processing.

VI. Wireless Baseband Processor Survey

In recent years, a large number of architectures have been proposed for baseband processing. Most of these architectures are SIMD-based and consist of one or few high-performance DSP processors. The DSP processors are connected through a shared bus and managed through a general-purpose control processor. Many of them have a shared global memory connected to the bus. Table X compares the different SIMD-based architectures with respect to SIMD width, VLIW support, frequency, and memory organization.

The DSP processors differ in the width of the SIMD datapath. While SODA and EVP [4] support a wide SIMD datapath (16 and 32 lanes respectively), TigerSHARC [2], DXP [3], Sandblaster [5], and SIMT [6] have narrow SIMD datapaths (four lanes). In general, wider SIMD datapaths have higher power efficiency but require higher levels of data-level parallelism to be effective. Since the majority of baseband computations are on wide vector arithmetics, a wide SIMD can be utilized fairly well. In addition, most of the SIMD-based DSP processors support VLIW execution by allowing concurrent memory and SIMD arithmetic operations. TigerSHARC and SIMT go one step further and provide concurrent SIMD arithmetic operations by having two four-lane SIMD ALU units that are controlled with different instructions. Sandblaster also supports multhreading in order to avoid overheads caused by use of long instruction words in VLIW.

All the SIMD-based DSP processors use software-managed scratchpad data memories instead of caches. This is because data are accessed in a regular pattern in baseband processing and use of scratch pad memories helps reduce power. Most of the processors operate at relatively low frequencies. The exception is ICERA DXP [3] which implements a deeply pipelined high-frequency design. Its SIMD ALUs are chained so that a sequence of vector operations can be performed before the data are written back to the register file. Finally, some of these architectures have hardware accelerators for error correction algorithms such as Viterbi and Turbo coding.

Apart from the SIMD-based architectures, there are a few reconfigurable architectures such as the fine-grained architectures PicoArray [8] and XiRisc [9], and the coarse-grained architectures including Intel RCA [10], QuickSilver [11], Montium [12], and ADRES [13]. These architectures have different types of PEs, ranging from simplescalar processors to application-specific processors which typically serve as hardware accelerators for error correction. These heterogeneous systems provide a trade-off between system flexibility and computational efficiency of the individual kernels.

VII. Conclusion

This paper presents a low-power, high-performance programmable DSP architecture that is optimized for baseband processing of wireless terminals. It builds on our previously proposed architecture SODA which supported WCDMA 2 Mbps and IEEE 802.11a efficiently. The power and throughput performance of the proposed architecture, SODA-II is significantly better than SODA. This is because of: 1) chaining operations in the computation units, which reduces the number of load/store instructions; 2) pipelined execution of vector operations in the SIMD pipeline which allows for multiple vector operations to be processed in parallel; and 3) staggered access of shared memory and multicycling in the computation units, which reduces the overhead of data alignment.

The performance of the proposed architecture is evaluated with an in-house emulator which uses accurate component-level models that were developed using Synopsys and Artisan tools. For WCDMA2-Mbps packet data service, SODA-II uses only two PEs and is estimated to consume 120 mW when operating at 300 MHz compared to SODA which uses all four PEs and is estimated to consume 210 mW. SODA-II has the capability of supporting higher throughput applications; it can support 9-Mbps

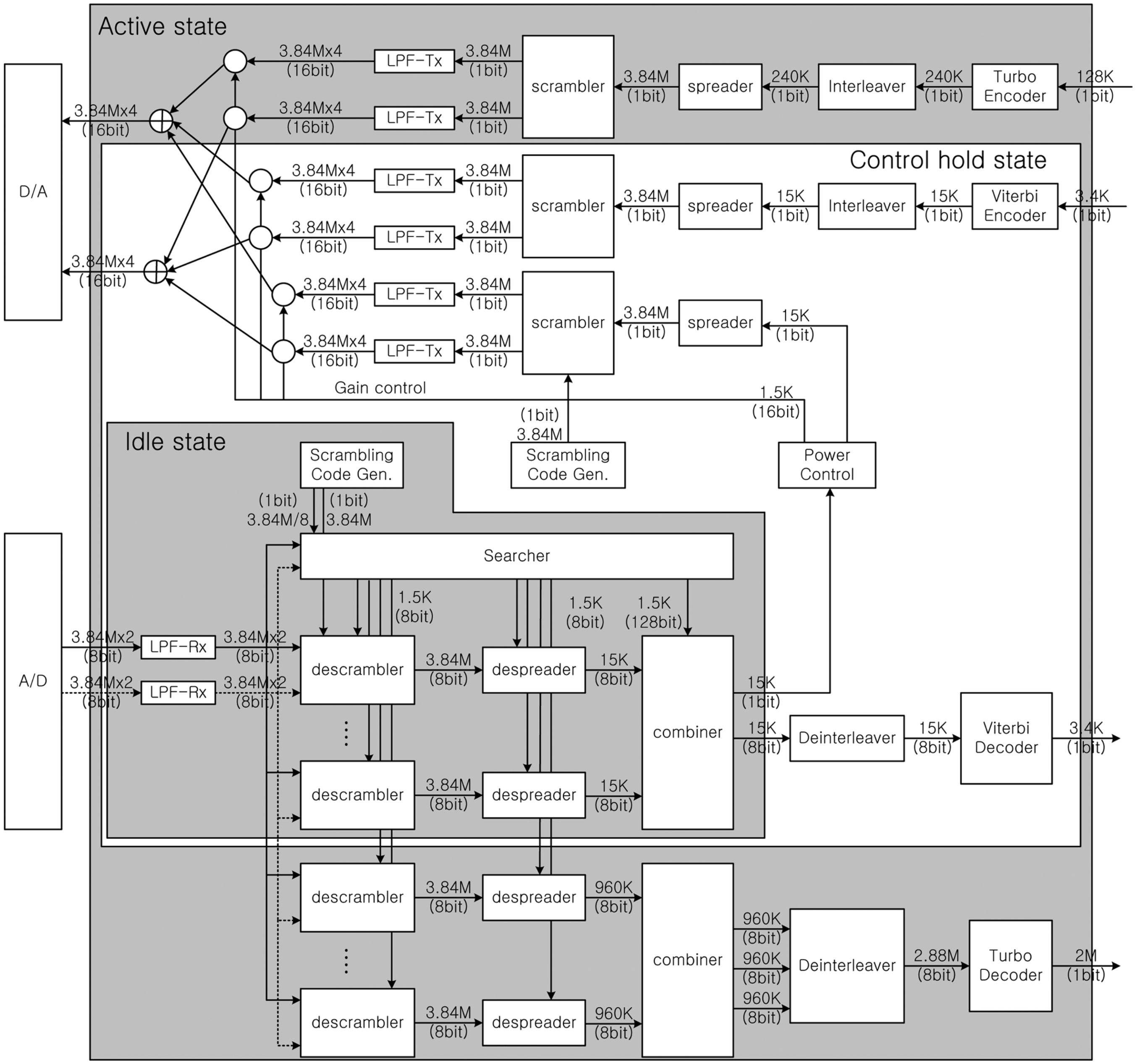


Fig. 13. Detailed block diagram of the WCDMA physical layer when it provides2-Mbps packet data service.

(WCDMA) throughput at 730 mW when operating four PEs at 700 MHz.

APPENDIX
CHARACTERIZATION OF WCDMA PHYSICAL LAYER

Fig. 13 shows a detailed block diagram of the WCDMA physical layer. It supports three modes of operation: idle mode, control hold mode, and active mode. In the idle mode, only a part of the reception path is active, specifically the low-pass filter (LPF)-Rx, demodulator (descrambler, despreader, and combiner), and multipath searcher. In the control hold mode, a bidirectional 3.4-Kbps signaling link is established with the base stations and both the transmission and receiver paths are switched on. The convolutional encoder/Viterbi decoder, LPF-Rx/Tx, modulator/demodulator, multipath searcher, and power control are all activated. In the active mode, a terminal additionally establishes a bidirectional high-speed data link where the error control is done by Turbo codes. Turbo decoding is computationally very intensive and significantly increases the terminal's workload. Here we assume that Turbo decoding has been implemented by the soft output Viterbi algorithm (SOVA).

Fig. 13 also describes the interface between the kernel algorithms. The number at the top of each arrow represents the number of samples per second, and the number at the bottom represents its bit width. We see that the size of most input/output data in the transmission path is 1 bit, but, in the reception path, the size of input/output data is 8 or 16 bit because of the precision requirements for channel decoding. The data rate undergoes significant jumps at the spreader and despreader. For in-

TABLE XI
PEAK WORKLOAD PROFILE OF THE WCDMA PHYSICAL LAYER FOR THE THREE OPERATION MODES

	Active		**Control Hold**		**Idle**	
	(MOPS)	%	**(MOPS)**	%	**(MOPS)**	%
Searcher	**26,538.0**	**42.1**	**26,358.0**	**58.4**	**3,317.3**	**37.7**
Interleaver	2.2	0.0	2.2	0.0	-	-
Deinterleaver	0.2	0.0	0.2	0.0	-	-
Conv. encoder	0.0	0.0	0.0	0.0	-	-
Viterbi Decoder	200.0	0.3	200.0	0.4	-	-
Turbo encoder	0.0	0.0	0.0	0.0	-	-
Turbo decoder	**17,500.0**	**27.8**	**0.0**	**0.0**	-	-
Scrambler	245.3	0.4	245.3	0.5	-	-
Descrambler	**2,621.4**	**4.2**	**2,621.4**	**5.8**	**889.2**	**10.1**
Spreader	297.5	0.5	297.5	0.7	-	0.0
Despreader	**3,642.5**	**5.8**	**3,642.5**	**8.0**	**607.1**	**6.9**
LPF-Rx	**3,993.6**	**6.3**	**3,993.6**	**8.8**	**3,993.6**	**45.3**
LPF-Tx	**7,897.2**	**12.6**	**7,897.2**	**17.4**	-	-
Power control	0.0	0.0	0.0	0.0	-	-
Total	62,937.0	-	45,272.9	-	8,807.2	-

stance, in the transmission path, the data rate is upconverted from kilosamples per second into megasamples per second after the spreading operation.

The detailed peak workload profile of the WCDMA physical layer is shown in Table XI. This was generated by compiling the W-CDMA benchmark (written in C) with an Alpha gcc compiler, and executing it on the M5 architecture simulator [19]. The instruction count required to finish each algorithm was taken and the peak workload of each algorithm calculated by dividing the instruction count by the tightest processing time requirement of the algorithm.

The first thing to note in Table XI is that the total workload varies significantly with the operation mode. For instance, the total workload in the idle mode is only 14% of that in the active mode. Second, kernel algorithms and the workload assigned to them also vary according to the operation mode. For instance, the multipath searcher has the same workload in the active and control hold modes but significantly lower workload in the idle mode. In general, the workload in the transmission path is much less than that in the receiver path.

Next we present the parallelism in each of the W-CDMA algorithms in Table XII. The second and third columns in the table present the ratio between the run time of the scalar code and the vector code. The fourth column represents the maximum possible data-level parallelism, DLP, defined as the maximum SIMD vector width. The fifth column provides the bit width of the two vector operands in the fifth column. The last column shows the thread-level parallelism, TLP.

From Table XII, we can see that the searcher, LPF, scrambler, descrambler, and the BMC/ACS of the Viterbi decoder exhibit considerable DLP and TLP. The DLP of the scrambler and descrambler can be converted into TLP by subdividing large vectors into smaller ones. The Turbo decoder contains limited DLP because the allowed maximum vector length of the ACS operation of the Turbo decoder is 8. It is possible to increase the DLP by decoding multiple blocks simultaneously.

TABLE XII
PARALLELISM AVAILABLE IN THE ALGORITHMS OF THE WCDMA PHYSICAL LAYER

		Scalar Load (%)	**Vector Load** (%)	**Vector Width**	**Data Width** (bit)	**Max Task**
Searcher		3	97	320	1,8	5120
Interleaver		100	0	-	-	-
Deinterleaver		100	0	-	-	-
Viterbi encoder		60	40	8	1,1	1
Viterbi	BMC	1	99	256	8,8	45
	ACS	1	99	256	8,8	45
	TB	100	0	-	-	-
Turbo encoder		60	40	4	1,1	2
Turbo	BMC	1	99	16	8,8	20
	ACS	1	99	16	8,8	20
	TB	100	0	-	-	-
Scrambler		1	99	2560	1,1	1
Descrambler		1	99	2560	1,8	1
Spreader		100	0	-	-	-
Despreader		100	0	-	-	-
Combiner		100	0	-	-	-
LPF-Tx		1	99	32	1,16	6
LPF-Tx		1	99	32	8,8	2
Power control		100	0	-	-	-

Table XII also shows that there are several algorithms that cannot be parallelized. Examples include the interleaver, deinterleaver, spreader, despreader, and combiner. However, the workload of these algorithms is not significant as shown in Table XI. Therefore we can easily increase system throughput and power efficiency by exploiting the inherent DLP and TLP shown in Table XII. From this analysis, we conclude that the architecture should support both scalar and vector operations and that the architecture should especially be optimized for vector operations.

REFERENCES

[1] H.-M. Bluethgen, C. Grassmann, W. Raab, and U. Ramacher, "A programmable baseband platform for software-defined radio," presented at the SDR Tech. Conf. and Product Exposition, Nov. 2004.

[2] J. Fridman and Z. Greenfield, "The TigerSHARC DSP architecture," *IEEE Micro*, vol. 20, no. 1, pp. 66–76, Jan./Feb. 2000.

[3] S. Knowles, "The SOC future is soft," in *IEE Cambridge Branch Seminar*, Dec. 2005.

[4] K. v. Berkel, F. Heinle, P. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Appl. Signal Process.*, vol. 2005, no. 16, pp. 2613–2625, 2005.

[5] J. Glossner, D. Lancu, L. Jin, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," *IEEE Commun. Mag.*, vol. 41, no. 1, pp. 120–128, Jan. 2003.

[6] A. Nilsson, E. Tell, and D. Liu, "An 11 mm^2 70 mW fully-programmable baseband processor for mobile WiMAX and DVB-T/H in 0.12 μm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2008, pp. 266–267.

[7] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A low-power architecture for software radio," in *Proc. Int. Symp. Comput. Architecture*, Jun. 2006, pp. 89–101.

[8] A. Duller, G. Panesar, and D. Towner, "Parallel processing—The PicoChip way!," *Commun. Process Arch.*, pp. 125–138, Sep. 2003.

[9] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. D. Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, and R. Guerrieri, "XiSystem: A XiRisc-based SoC with reconfigurable I/O module," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 85–96, Jan. 2006.

[10] A. Chun, E. Tsui, I. Chen, H. Honary, and J. Lin, "Application of Intel reconfigurable architecture to 802.11a, 3G and 4G standards," presented at the Circuit and System Symp. Emerging Technologies, Frontiers of Mobile and Wireless Communications, May 2004.

[11] QuickSilver Technology [Online]. Available: http://www.qstech.com

[12] G. Smit, A. Kokkeler, P. Wolkotte, P. Hölzenspies, M. Burgwal, and P. Heysters, "The chameleon architecture for streaming DSP applications," *EURASIP J. Embedded Syst.*, vol. 2007, Art. no. 78082.

[13] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereins, "Design methodology for a tightly coupled VLIW/reconfigurable matrix architecture: A case study," in *Design Automation and Test in Europe*, 2004, vol. 2, pp. 21 224–21 229.

[14] *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*, H. Holma and A. Toskala, Eds. Hoboken, NJ: Wiley , 2000.

[15] P. Roshan and J. Leary*, 802.11 Wireless LAN Fundamentals*. Indianapolis, IN: Cisco Press, 2003.

[16] D. Talla, L. John, and D. Burger, "Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements," *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 1015–1031, Aug. 2003.

[17] H. Hunter and J. Moreno, "A new look at exploiting data parallelism in embedded systems," in *Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Syst.*, Oct. 2003, pp. 159–169.

[18] H. Lee, "A baseband processor for software defined radio terminals," Ph.D. thesis, Univ. Michigan, Ann Arbor, MI, 2007.

[19] N. L. Binkert *et al.*, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul./Aug. 2006.

Hyunseok Lee (M'08) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2007.

He is currently an Assistant Professor in the Department of Electronics and Communications Engineering, Kwangwoon University, Seoul, Korea. From 1992 to 2008, he participated in the development of IS-95, cdma2000, WCDMA, and mobile WiMAX systems at Samsung Electronics, Suwon, Korea. His research interest includes low-power signal processing architectures and embedded systems for wireless communications.

Chaitali Chakrabarti (S'86–M'89–SM'02) is a Professor of electrical engineering at Arizona State University, Tempe. Her research interests include all aspects of low-power embedded systems design and VLSI architectures and algorithms for signal processing, image processing, and communications.

Dr. Chakrabarti served as the Technical Committee Chair of the DISPS subcommittee, IEEE Signal Processing Society (2006–2007). She is currently an Associate Editor of the *Journal of VLSI Signal Processing Systems* and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS.

Trevor Mudge (S'74–M'77–SM'84–F'95) received the Ph.D. degree in computer science from the University of Illinois, Urbana-Champaign.

Since then, he has been at the University of Michigan, Ann Arbor. He was named the Bredt Professor of Engineering after a ten-year term as Director of the Advanced Computer Architecture Laboratory—a group of a dozen faculty and 80 graduate students. He is an author of numerous papers on computer architecture, programming languages, VLSI design, and computer vision. He has also chaired 40 theses in these areas.

Dr. Mudge is a Fellow of the IEEE, a member of the ACM, the IET, and the British Computer Society.