Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 56467, 16 pages
doi:10.1155/2007/56467

Research Article

The Sandbridge SB3011 Platform

John Glossner, Daniel Iancu, Mayan Moudgill, Gary Nacer, Sanjay Jinturkar, Stuart Stanley, and Michael Schulte

Sandbridge Technologies, Inc., 1 North Lexington Avenue, White Plains, NY 10601, USA

Received 1 August 2006; Revised 18 January 2007; Accepted 20 February 2007

Recommended by Jarmo Henrik Takala

This paper describes the Sandbridge Sandblaster real-time software-defined radio platform. Specifically, we describe the SB3011 system-on-a-chip multiprocessor. We describe the software development system that enables real-time execution of communications and multimedia applications. We provide results for a number of interesting communications and multimedia systems including UMTS, DVB-H, WiMAX, WiFi, and NTSC video decoding. Each processor core achieves 600 MHz at 0.9 V operation while typically dissipating 75 mW in 90 nm technology. The entire chip typically dissipates less than 500 mW at 0.9 V.

1. INTRODUCTION

Performance requirements for mobile wireless communication devices have expanded dramatically since their inception as mobile telephones. Recent carrier offerings with multiple communication systems and handover from cellular to broadband suggest that some consumers are requesting convergence devices with full data and voice integration. The proliferation of cameras and Internet access in cell phones also suggests a variety of computationally intense features and applications such as web browsing, MP3 audio, and MPEG4 video are needed. Moreover, consumers want these wireless subscriber services to be accessible at all times anywhere in the world. Such complex functionality requires high computing capability at low power consumption; adding new features requires adding computing capacity.

The technologies necessary to realize true broadband wireless handsets and systems presenting unique design challenges if extremely power efficient, yet high-performance, broadband wireless terminals are to be realized. The design tradeoffs and implementation options inherent in meeting such demands highlight the extremely onerous requirements for next generation baseband processors. Tremendous hardware and software challenges exist to realize convergence devices.

The increasing complexities of mobile terminals and a desire to generate multiple versions with increasing features for handsets have led to the consideration of a software-defined radio- (SDR-) based approach in the wireless industry. The previous generation of mobile terminals was primarily designed for use in geographically restricted areas where growth of the wireless industry was dependant upon signing up new users. The penetration levels in European and Asian countries are high and new revenue streams (from technologies such as 3G) have been slow to materialize for a variety of complex reasons. True convergence of multimedia, cellular, location and connectivity technologies is expensive, time consuming, and complex at all levels of development—not only mobile terminals, but infrastructure as well. Moreover, the standards themselves have failed to converge, which has led to multiple market segments. In order to maintain market share, a handset development company must use multiple platforms each of which may be geographically specific supporting multiple combinations of communications systems. This requires some handset companies to support multiple platforms and multiple hardware solutions from multiple technology suppliers.

1.1. SDR-based approach

Building large parallel processing systems is a difficult task. Programming them efficiently is even more challenging. When nonassociative digital signal processing (DSP) arithmetic is included, the challenge of automated software development for a complex chip multiprocessor (CMP) system is amplified.

Early software-defined radio (SDR) platforms were often built out of discrete processors and FPGAs that were integrated on a card. More recently a trend has been to integrate multiple processors on a single chip creating SDR CMP systems. The SDR Forum [1] defines five tiers of solutions. Tier-0 is a traditional radio implementation in hardware. Tier-1, software-controlled radio (SCR), implements the control features for multiple hardware elements in software. Tier-2, software-defined radio (SDR), implements modulation and baseband processing in software but allows for multiple frequency fixed function RF hardware. Tier-3, ideal software radio (ISR), extends programmability through the RF with analog conversion at the antenna. Tier-4, ultimate software radio (USR), provides for fast (millisecond) transitions between communications protocols in addition to digital processing capability.

The advantages of reconfigurable SDR solutions versus hardware solutions are significant. First, reconfigurable solutions are more flexible allowing multiple communication protocols to dynamically execute on the same transistors thereby reducing hardware costs. Specific functions such as filters, modulation schemes, encoders/decoders can be reconfigured adaptively at run time. Second, several communication protocols can be efficiently stored in memory and coexist or execute concurrently. This significantly reduces the cost of the system for both the end user and the service provider. Third, remote reconfiguration provides simple and inexpensive maintenance and feature upgrades. This also allows service providers to differentiate products after the product is deployed. Fourth, the development time of new and existing communications protocols is significantly reduced providing an accelerated time to market. Development cycles are not limited by long and laborious hardware design cycles. With SDR, new protocols are quickly added as soon as the software is available for deployment. Fifth, SDR provides an attractive method of dealing with new standards releases while assuring backward compatibility with existing standards.

SDR enabling technologies also have significant advantages from the consumer perspective. First, mobile terminal independence with the ability to “choose” desired feature sets is provided. As an example, the same terminal may be capable of supporting a superset of features but the consumer only pays for features that they are interested in using. Second, global connectivity with the ability to roam across operators using different communications protocols can be provided. Third, future scalability and upgradeability provide for longer handset lifetimes.

1.2. Processor background

In this section we define a number of terms and provide background information on general purpose processors, digital signal processors, and some of the workload differences between general purpose computers and real-time embedded systems.

The *architecture* of a computer system is the minimal set of properties that determine what programs will run and what results they will produce [2]. It is the contract between the programmer and the hardware. Every computer is an interpreter of its *machine language*—that representation of programs that resides in memory and is interpreted (executed) directly by the (host) hardware.

The logical organization of a computer’s dataflow and controls is called the *implementation or microarchitecture*. The physical structure embodying the implementation is called the *realization*. The architecture describes what happens while the implementation describes how it is made to happen. Programs of the same architecture should run unchanged on different implementations. An architectural function is *transparent* if its implementation does not produce any architecturally visible side effects. An example of a nontransparent function is the load delay slot made visible due to pipeline effects. Generally, it is desirable to have transparent implementations. Most DSP and VLIW implementations are not transparent and therefore the implementation affects the architecture [3].

Execution predictability in DSP systems often precludes the use of many general-purpose design techniques (e.g., speculation, branch prediction, data caches, etc.). Instead, classical DSP architectures have developed a unique set of performance-enhancing techniques that are optimized for their intended market. These techniques are characterized by hardware that supports efficient filtering, such as the ability to sustain three memory accesses per cycle (one instruction, one coefficient, and one data access). Sophisticated addressing modes such as bit-reversed and modulo addressing may also be provided. Multiple address units operate in parallel with the datapath to sustain the execution of the inner kernel.

In classical DSP architectures, the execution pipelines were visible to the programmer and necessarily shallow to allow assembly language optimization. This programming restriction encumbered implementations with tight timing constraints for both arithmetic execution and memory access. The key characteristic that separates modern DSP architectures from classical DSP architectures is the focus on compilability. Once the decision was made to focus the DSP design on programmer productivity, other constraining decisions could be relaxed. As a result, significantly longer pipelines with multiple cycles to access memory and multiple cycles to compute arithmetic operations could be utilized. This has yielded higher clock frequencies and higher performance DSPs.

In an attempt to exploit instruction level parallelism inherent in DSP applications, modern DSPs tend to use VLIW-like execution packets. This is partly driven by real-time requirements which require the worst-case execution time to be minimized. This is in contrast with general purpose CPUs which tend to minimize average execution times. With long pipelines and multiple instruction issue, the difficulties of attempting assembly language programming become apparent. Controlling dependencies between upwards of 100 in-flight instructions is not an easy task for a programmer. This is exactly the area where a compiler excels.

One challenge of using some VLIW processors is large program executables (code bloat) that result from independently specifying every operation with a single instruction. As an example, a VLIW processor with a 32-bit basic instruction width may require 4 instructions, 128 bits, to specify 4 operations. A vector encoding may compute many more operations in as few as 21 bits (e.g., multiply two 4-element vectors, saturate, accumulate, and saturate).

Another challenge of some VLIW implementations is that they may have excessive register file write ports. Because each instruction may specify a unique destination address and all the instructions are independent, a separate port may be provided for the target of each instruction. This can result in high power dissipation, which is unacceptable for handset applications.

To help overcome problems with code bloat and excessive write ports, recent VLIW DSP architectures, such as OnDSP [4], the embedded vector processor (EVP) [5], and the synchronous transfer architecture (STA) [6], provide vector operations, specialized instructions for multimedia and wireless communications, and multiple register files.

A challenge of visible pipeline machines (e.g., most DSPs and VLIW processors) is interrupt response latency. It is desirable for computational datapaths to remain fully utilized. Loading new data while simultaneously operating on current data is required to maintain execution throughput. However, visible memory pipeline effects in these highly parallel inner loops (e.g., a load instruction followed by another load instruction) are not typically interruptible because the processor state cannot be restored. This requires programmers to break apart loops so that worst-case timings and maximum system latencies may be acceptable.

Signal processing applications often require both computations and control processing. Control processing is often amenable to RISC-style architectures and is typically compiled directly from C code. Signal processing computations are characterized by multiply-accumulate intensive functions executed on fixed point vectors of moderate length. Therefore, a DSP requires support for such fixed point saturating computations. This has traditionally been implemented using one or more multiply accumulate (MAC) units. In addition, as the saturating arithmetic is nonassociative, parallel operations on multiple data elements may result in different results from serial execution. This creates a challenge for high-level language implementations that specify integer modulo arithmetic. Therefore, most DSPs have been programmed using assembly language.

Multimedia adds additional requirements. A processor which executes general purpose programs, signal processing programs, and multimedia programs (which may also be considered to be signal processing programs) is termed a convergence processor. Video, in particular, requires high performance to allow the display of movies in real-time. An additional trend for multimedia applications is Java execution. Java provides a user-friendly interface, support for productivity tools and games on the convergence device.

The problems associated with previous approaches require a new architecture to facilitate efficient convergence applications processing. Sandbridge Technologies has developed a new approach that reduces both hardware and software design challenges inherent in real-time applications like SDR and processing of streaming data in convergence services.

In the subsequent sections, we describe the Sandbridge SB3011 low-power platform, the architecture and implementation, the programming tools including an automatically multithreading compiler, and SDR results.

2. THE SB3011 SDR PLATFORM

Motivated by the convergence of communications and multimedia processing, the Sandbridge SB3011 was designed for efficient software execution of physical layer, protocol stacks, and multimedia applications. The Sandbridge SDR platform is a Tier-2 implementation as defined by the SDR Forum. Figure 1 shows the SB3011 implementation. It is intended for handset markets. The main processing complex includes four DSPs [7] each running at a minimum of 600 MHz at 0.9 V. The chip is fabricated in 90 nm technology. Each DSP is capable of issuing multiple operations per cycle including data parallel vector operations. The microarchitecture of each DSP is 8-way multithreaded allowing the SB3011 to simultaneously execute up to 32 independent instruction streams each of which may issue vector operations.

2.1. *DSP complex*

Each DSP has a level-1 (L1) 32 KB set-associative instruction cache and an independent L1 64 KB data memory which is not cached. In addition a noncached global level-2 (L2) 1 MB memory is shared among all processors. The implementation guarantees no pipeline stalls for L1 memory accesses (see Section 4). For external memory accesses or L2 accesses only the thread that issued the load request stalls. All other threads continue executing independent of which processor issued the memory request.

The Sandblaster DSP is a true architecture in the sense that from the programmer's perspective each instruction completes prior to the next instruction issuing—on a per thread basis.

The processors are interconnected through a deterministic and opportunistic unidirectional ring network. The interconnection network typically runs at half the processor speed. The ring is time-division multiplexed and each processor may request a slot based on a proprietary algorithm. Communications between processors is primarily through shared memory.

The processor's instruction set provides synchronization primitives such as load locked and store conditional. Since all data memory is noncached, there are no coherence issues.

2.2. *ARM and ARM peripherals*

In addition to the parallel multithreaded DSP complex, there is an entire ARM complex with all the peripherals necessary to implement input/output (I/O) devices in a smart phone. The processor is an ARM926EJ-S running at up to 300 MHz. The ARM has 32 kB instruction and 32 kB data cache memories. There is an additional 64 kB of scratch

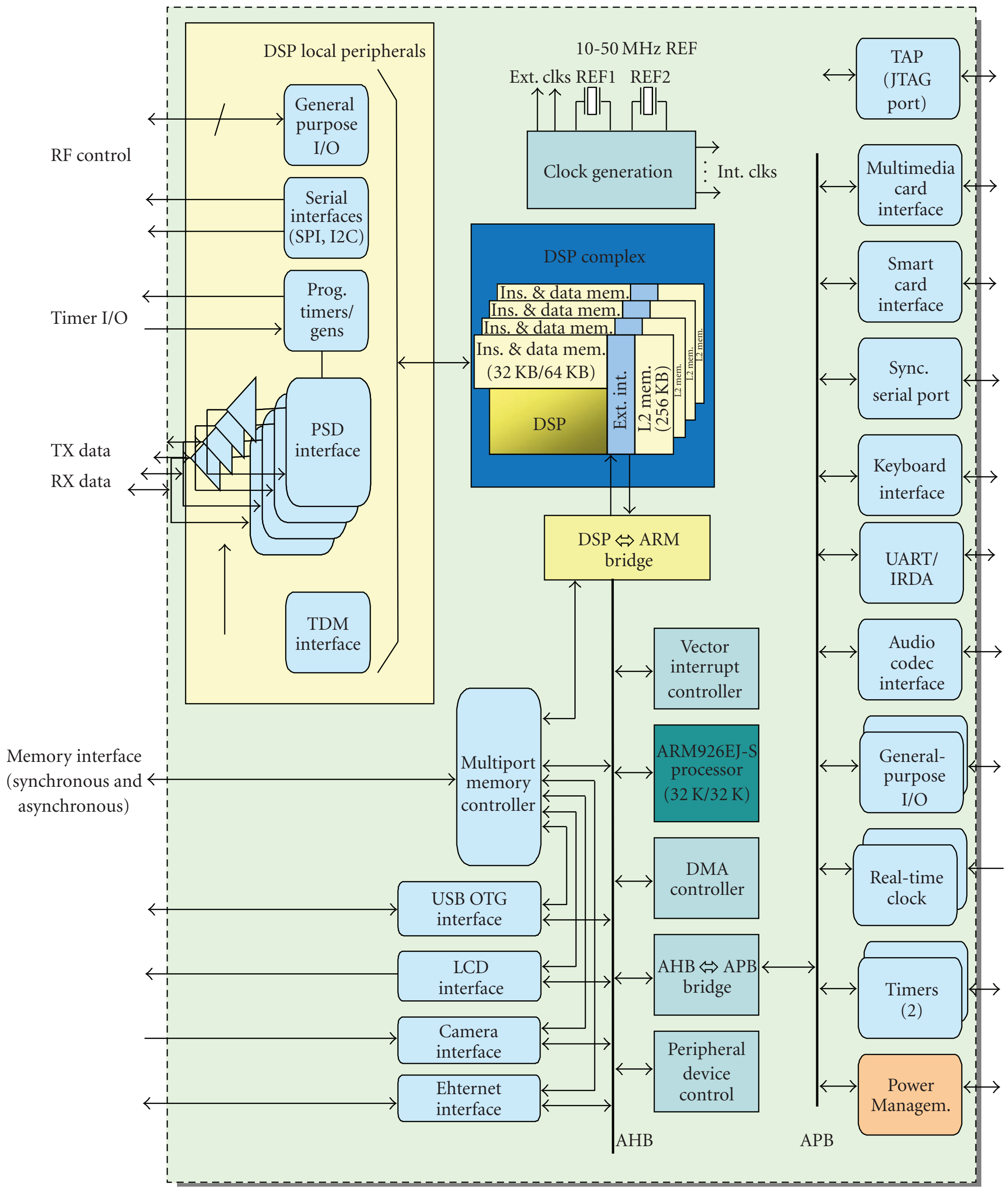

Figure 1: Sandblaster SB3011 chip.

memory partitioned as 32 kB instruction and 32 kB data. Sandbridge has ported Linux to this platform and the processor functions as the user interface (UI or sometimes MMI) for smart phone applications.

Using an AMBA advanced high-speed bus (AHB) and advanced peripheral bus (APB), the system is able to support the processing of multiple concurrent data interfaces. Attached to the APB is a multimedia card (MMC) and secure digital card (SD Card) interface for connecting external Flash storage. Keyboard and mouse interfaces are included along with multiple UARTs and an IRDA interface. Audio and microphone PCM data is supported through an AC-97 interface which connects directly to an external codec. A number of other general peripherals are also supported on the APB including a real-time clock and general purpose timers, which are used to keep system time.

The AHB is used to move high-speed data such as memory requests and video into the chip for processing or out of the chip for display or storage. A direct connection to an LCD display is provided and can be accessed from either the ARM or DSP processors. Similarly, a high-speed camera interface is provided to capture raw video data that may be encoded or processed in the DSP or ARM processors.

The SB3011 includes a full USB 2.0 on-the-go (OTG) implementation for seamless connection to printers, cameras, storage, or other USB devices. An Ethernet interface is also included on the chip for wired local area network (LAN) connections.

2.3. *External memory*

External memory requests which both the ARM and DSPs can initiate are routed through a multiport memory controller attached to the AHB. The external memory can be synchronous or asynchronous. Typical memories include Flash, SDRAM, DRAM, and SRAM. The controller supports multiple simultaneous requests whether generated through direct memory access (DMA) devices (both for the ARM and DSP) or by a direct address from the processors. External memory requests are managed by an arbitration controller which ensures priority and fairness for the bus transactions. All external memory is mapped into a 32-bit global address space and is shared among all processors.

Device processors are booted from external memory in a two-step sequence: ARM followed by the DSPs. Once the device is released from reset, the ARM processor begins execution from one of the memory controller's memory ports (typically the port connected to Flash memory on the card). The ARM then executes a device initialization routine, which configures the necessary internal device peripherals and the execution vectors for the DSPs. Once complete, the DSPs are enabled and each processor begins executing the Sandbridge Operating System (SBOS), which may be in Flash or other memory.

2.4. *DSP peripherals*

In addition to the ARM peripherals, there are a number of DSP specific peripherals primarily intended for moving data to and from external radio frequency (RF) devices, time division multiplexed (TDM) voice and data devices (e.g., T1/E1), and other peripherals. These peripherals interface directly to the DSPs' L2 memory subsystem through the multiple parallel streaming data (PSD) or TDM interfaces. Four half-duplex PSD interfaces are provided, each supporting up to 16-bit data samples. PSD data is latched or transmitted by the device on both edges of its respective clock, thus realizing two data streams per interface (typically I and Q streams to and from an RF's analog-front-end device). Four serial TDM interfaces are provided, each of which capable of up to 1024 channels, for an aggregate 32 k samples per second throughput. Support for synchronization of transmitted or received data bursts is accomplished through the use of dedicated I/O timers. When configured, these timers can be operated with an external (system) clock source and are internally used to gate the DMA transfers on the PSD interfaces. This feature is important for slot-based communications systems such as GSM.

A number of other interfaces are provided for general purpose control of external components typically found in smart phones. These include general purpose timers which can be used as external clocks, SPI, and I2C buses which are common in RF control logic, and general purpose I/O. The SPI and I2C peripherals allow the DSPs to compute in software functions such as automatic gain control (AGC) and send the information seamlessly to the RF control interface. The DSP computes the changed values and the SPI or I2C bus delivers the information to the external chip(s).

2.5. *Power management*

To facilitate flexible system-level power management, the Sandblaster SB3011 incorporates thirteen independent power domains. Each processor core is isolated by a separate domain thus 5 domains encapsulate the ARM plus 4 DSPs. An additional domain is used for L2 memories. The other power domains are used to isolate specific logic portions of the chip.

Each domain is independently controllable by the Device Power Management Unit (DPMU) which is itself isolated within an independent power domain. The DPMU is a programmable peripheral that allows for the following options: (1) the ability to place the device in power down where all data and internal state is not maintained and (2) the ability to place each processor independently in power down where each core does not maintain state but the L2 memories are back-biased and thus retain state.

In addition to the voltage control features, clock management is also included in two forms: (1) instruction-based automatic clock enable/disable operation where the hardware dynamically controls clocks to internal sub-blocks when inactivity is detected and (2) operating System (OS) or application-based clock enable/disable which can control DSP cores, AHB peripherals, LCD, Camera, USB, Ethernet, and APB peripherals.

While not a comprehensive list, some typical profiles of low power configurations include the following. (1) Device Deep Sleep where the all the SB3011 functional blocks are powered off with the exception of the Device Power Management Unit. No state is retained in this mode. In this state only the DPMU is powered since it is required to wake the rest of the chip. (2) Processing Unit Deep Sleep Mode where all the processor cores are shut down without state retention. However, L2 memories and peripherals retain state and may function. (3) Device Standby where all DSP cores and the ARM processor clocks are disabled but full state is retained.

The subsequent sections discuss the Sandblaster DSP architecture and programming tools that enable real-time implementation of the parallel SDR system.

3. SANDBLASTER LOW-POWER ARCHITECTURE

3.1. *Compound instructions*

The Sandblaster architecture is a compound instruction set architecture [7]. Historically, DSPs have used compound

```
L0: lvu %vr0, %r3, 8

||                    vmulreds
%ac0,%vr0,%vr0,%ac0

|| loop %lc0,L0
```

Figure 2: Compound instruction for sum of squares inner loop.

instruction set architectures to conserve instruction space encoding bits. In contrast, some VLIW architectures contain full orthogonality, but only encode a single operation per instruction field, such that a single VLIW is composed of multiple instruction fields. This has the disadvantage of requiring many instruction bits to be fetched per cycle, as well as significant write ports for register files. Both these effects contribute heavily to power dissipation. Recent VLIW DSP architectures, such as STA, overcome these limitations by providing complex operations, vector operations, and multiple register files.

In the Sandblaster architecture, specific fields within the instruction format may issue multiple suboperations including data parallel vector operations. Most classical DSP instruction set architectures are compound but impose restrictions depending upon the particular operations selected. In contrast, some VLIW ISAs allow complete orthogonality of specification and then fill in any unused issue slots by inserting no operation instructions (NOPs) either in hardware or software.

3.2. *Vector encoding*

In addition to compound instructions, the Sandblaster architecture also contains vector operations that perform multiple compound operations. As an example, Figure 2 shows a single compound instruction with three compound operations. The first compound operation, lvu, loads the vector register vr0 with four 16-bit elements and updates the address pointer r3 to the next element. The vmulreds operation reads four fixed point (fractional) 16-bit elements from vr0, multiplies each element by itself, saturates each product, adds all four saturated products plus an accumulator register, ac0, with saturation after each addition, and stores the result back in ac0. The vector architecture guarantees Global System for Mobile communication (GSM) semantics (e.g., bit-exact results) even though the arithmetic performed is nonassociative [8]. The loop operation decrements the loop count register lc0, compares it to zero, and branches to address L0 if the result is not zero.

3.3. *Simple instruction formats*

Simple and orthogonal instruction formats are used for all instructions. The type of operation is encoded to allow simple decoding and execution unit control. Multiple operation fields are grouped within the same bit locations. All operand fields within an operation are uniformly placed in the same bit locations whether they are register-based or immediate values. As in VLIW processors, this significantly simplifies the decoding logic.

3.4. *Low-power idle instructions*

Architecturally, it is possible to turn off an entire processor. All clocks may be disabled or the processor may idle with clocks running. Each hardware thread unit may also be disabled to minimize toggling of transistors in the processor.

3.5. *Fully interlocked*

Unlike some VLIW processors, our architecture is fully interlocked and transparent. In addition to the benefit of code compatibility, this ensures that many admissible and application-dependent implementations may be derived from the same basic architecture.

4. LOW-POWER MICROARCHITECTURE

4.1. *Multithreading*

Figure 3 shows the microarchitecture of the Sandblaster processor. In a multithreaded processor, multiple threads of execution operate simultaneously. An important point is that multiple copies (e.g., banks and/or modules) of memory are available for each thread to access. The Sandblaster architecture supports multiple concurrent program execution by the use of hardware thread units (called contexts). The architecture supports up to eight concurrent hardware contexts. The architecture also supports multiple operations being issued from each context. The Sandblaster processor uses a new form of multithreading called token triggered threading (T^3) [9].

With T^3, all hardware contexts may be simultaneously executing instructions, but only one context may issue an instruction each cycle. This constraint is also imposed on round-robin threading. What distinguishes T^3 is that each clock cycle, a token indicates the next context that is to be executed. Tokens may cause the order in which threads issue instructions to be sequential (e.g., round-robin), even/odd, or based on other communications patterns. Figure 4 shows an example of T^3 instruction issue, in which an instruction first issues from Thread 0, then Thread 3, then Thread 2, and so forth. After eight cycles, the sequence repeats with Thread 0 issuing its next instruction. Compared to SMT, T^3 has much less hardware complexity and power dissipation, since the method for selecting threads is simplified, only a single compound instruction issues each clock cycle, and most dependency checking and bypass hardware is not needed.

4.2. *Decoupled logic and memory*

As technology improves, processors are capable of executing at very fast cycle times. Current state-of-the-art 0.13 um technologies can produce processors faster than 3 GHz. Unfortunately, current high-performance processors consume

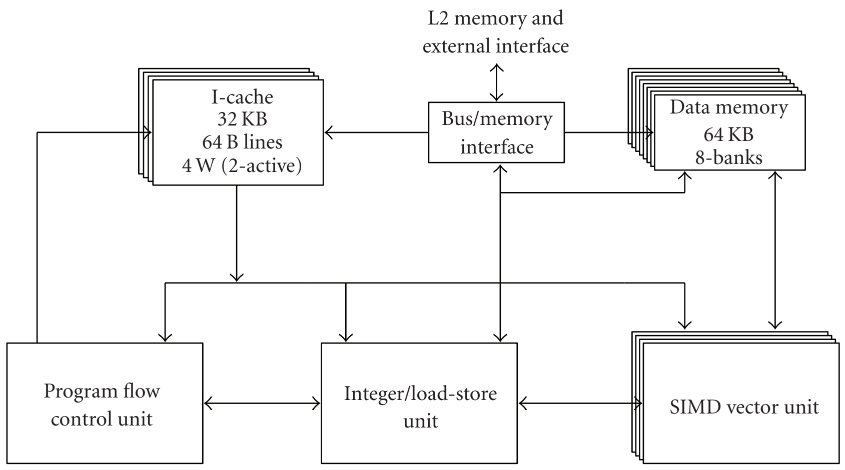
L2 memory and external interface
I-cache 32 KB 64 B lines 4 W (2-active)
Bus/memory interface
Data memory 64 KB 8-banks
Program flow control unit
Integer/load-store unit
SIMD vector unit

FIGURE 3: Multithreaded microarchitecture.

$$\begin{tikzcd}
T0 \arrow[r] & T3 \arrow[r] & T2 \arrow[r] & T1 \arrow[r] & T6 \arrow[r] & T5 \arrow[r] & T4 \arrow[r] & T7 \arrow[lllllll, bend left]
\end{tikzcd}$$

FIGURE 4: Token triggered threading, with even/odd sequencing.

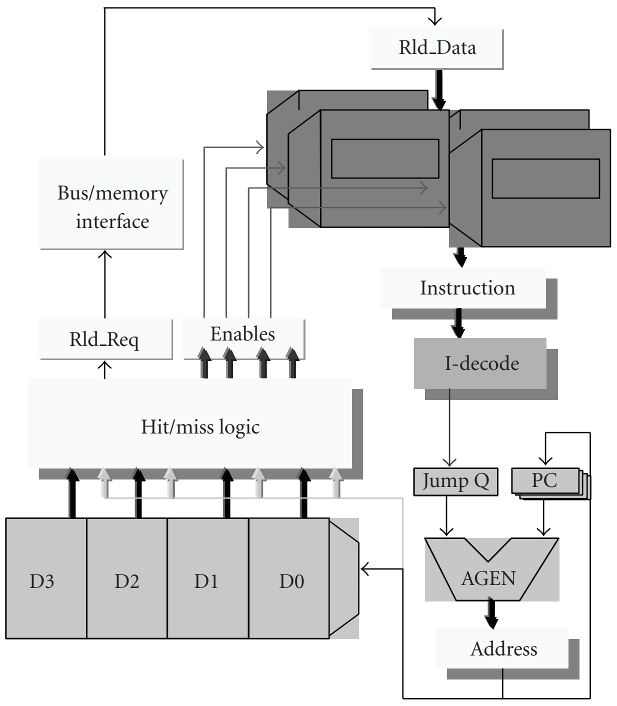

FIGURE 5: Cache memory design.

significant power. If power-performance curves are considered for both memory and logic within a technology, there is a region in which you get approximately linear increase in power for linear increase in performance. Above a specific threshold, there is an exponential increase in power for a linear increase in performance. Even more significant, memory and logic do not have the same threshold.

For 0.13 um technology, the logic power-performance curve may be in the linear range until approximately 600 MHz. Unfortunately, memory power-performance curves are at best linear to about 300 MHz. This presents a dilemma as to whether to optimize for performance or power. Fortunately, multithreading alleviates the power-performance trade-off. The Sandblaster implementation of multithreading allows the processor cycle time to be decoupled from the memory access time. This allows both logic and memory to operate in the linear region, thereby significantly reducing power dissipation. The decoupled execution does not induce pipeline stalls due to the unique pipeline design.

4.3. *Caches*

An instruction cache unit (ICU) stores instructions to be fetched for each thread unit. A cache memory works on the principle of locality. Locality can refer to spatial, temporal, or sequential locality [2]. We use set associative caches to alleviate multiple contexts evicting another context's active program. In our implementation, shown in Figure 5, there are four directory entries (D0–D3) and banked storage entries. A thread identifier register (not shown) is used to select whether the cache line in the left or right bank will be evicted. This effectively reduces the complexity of the cache line selection logic. In a 4-way set associative cache, only one additional least recently used (LRU) bit is needed to select which of the two lines should be evicted. This method of using thread information and banked memory accesses significantly reduces the complexity of the cache logic. In our implementation, a unique feature is the use of a read associativity of 4 and a write associativity of 2, which further reduces the cache logic complexity.

Ld/St	Inst. Dec.	RF read	Agen.	XFer	Int. ext.	Mem. 0	Mem. 1	Mem. 2	WB
ALU	Inst. Dec.	Wait	RF read	Exec. 1	Exec. 2	XFer	WB	—	—
I_Mul	Inst. Dec.	Wait	RF read	Exec. 1	Exec. 2	Exec. 3	XFer	WB	—
V_Mul	Inst. Dec.	VRF read	Mpy1	Mpy2	Add1	Add2	XFer	VRF WB	—

Figure 6: Processor pipeline.

4.4. Pipeline

The pipeline for one particular implementation of the Sandblaster DSP is shown in Figure 6. The execution pipelines are different for various functions. The Load/Store (Ld/St) pipeline is shown to have 9 stages. It is assumed that the instruction is already in the cache. The first stage decodes the instruction (Inst. Dec.). This is followed by a read from the general purpose register file. The next stage generates the address to perform the Load or Store. Five cycles are used to access data memory. Finally, the result for a Load instruction is written back (WB) to the referenced register file location. Once an instruction from a particular context enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction issuing from the same thread tries to use the result.

Similarly, there are multiple (variable) stages for other execution pipelines. The integer unit has three execute stages for multiplication (I_MUL) and two execute stages for addition (ALU). The vector unit has four execute stages, two for multiplication and two for addition.

4.5. Interlock checking hardware

Most interlocked architectures require significant interlock checking hardware and bypass logic for both correctness and performance reasons. Multithreading mitigates this effect. With the carefully designed pipeline shown in Figure 6, there is only one interlock that must actually be checked for in hardware, a long memory load or store. All other operations are guaranteed to complete prior to the same thread issuing a new instruction. This completely removes the power consuming interlock checks associated with most interlocked architectures.

5. LOW-POWER LOGIC DESIGN

5.1. Single write-port register files

Having multithreading to cover the latency associated with long pipeline implementations allows the use of single write-port register files even though more than one write may occur within an instruction cycle. An important point is that the write back stages are staggered. This allows a single write port to be implemented but provides the same functionality as multiple write ports [10].

An example is loading the integer register file while performing an integer multiply. From the processor pipeline shown in Figure 6, it is apparent that the reads and writes from the register file are staggered in time. In addition, separate architected register spaces for vector, integer, and accumulate operations enable reduced ports. A VLIW implementation of the instruction shown in Figure 2 may take many write ports for sustained single cycle throughput. Comparatively, our solution requires at most a single combined R/W port and an additional read port per register file.

5.2. Banked register files

Token triggered threading which follows a permutation of even and odd thread issue policies along with the pipeline implementation enables the use of banked register files. This allows the register files to run at half the processor clock, but never stall awaiting data.

5.3. Single-ported memories

The same characteristics that allow banked register file operation also enable the use of single ported L1 memories that may also be banked and run at half the processor clock. Since decoupled memories are highly desirable to reduce power, this provides significant overall savings.

5.4. Minimal control signals

A combination of architectural and microarchitectural techniques allows the processor to be implemented with very few control signals. Since control signals often propagate to many units, they become not only a source of bugs but also may dissipate significant power.

5.5. Clock gating

Because the architecture is modular and the pipeline is deep, there is time to compute which functional units will be active for each instruction. If a functional unit is not active, the clocks may be gated to that unit and suspend it on a unit-by-unit basis. As an example, if there are no vector operations on a given cycle, the vector unit is disabled. Even within a unit it is possible to gate the clocks. For example, if a vector multiply operation is executed but it does not need to be reduced, the reduce unit within the vector unit is gated off.

6. LOW-POWER CIRCUIT DESIGN

The average power consumption in a CMOS circuit can be modeled as

$$P_{\text{avg}} = \alpha C V_{\text{dd}}^2 f + V_{\text{dd}} I_{\text{mean}} + V_{dd} I_{\text{leak}}, \tag{1}$$

where α is the average gate switching activity, C is the total capacitance seen by the gates' outputs, V_{dd} is the supply voltage, f is the circuit's operating frequency, I_{mean} is the average current drawn during input transition, and I_{leak} is the average leakage current. The first term, $\alpha C V_{\text{dd}}^2 f$, which represents the dynamic switching power consumed by charging and discharging the capacitive load on the gates' outputs, often dominates power consumption in high-speed microprocessors [11]. The second term, $V_{\text{dd}} I_{\text{mean}}$, which represents the average dynamic power due to short-circuit current flowing when both the PMOS and NMOS transistors conduct during input signal transitions, typically contributes 10% to 20% of the overall dynamic power [12]. This is also a function of frequency but is simplified in this analysis. The third term, $V_{\text{dd}} I_{\text{leak}}$, represents the power consumed due to leakage current and occurs even in devices that are not switching. Consequently, for systems that are frequently in standby mode, the leakage power may be a dominate factor in determining the overall battery life. Since the leakage power increases exponentially with a linear decrease in device threshold voltage, leakage power is also a concern in systems that use power supply voltage scaling to reduce power.

6.1. Low-voltage operation

Since the dynamic switching power, $\alpha C V_{\text{dd}}^2 f$, is proportional to the supply voltage squared, an effective technique for reducing power consumption is to use a lower supply voltage. Unfortunately, however, decreasing the supply voltage also decreases the maximum operating frequency. To achieve high performance with a low-supply voltage, our arithmetic circuits are heavily pipelined. For example, our multiply-accumulate unit uses four pipeline stages. Our unique form of multithreading helps mask long pipeline latencies, so that high performance is achieved.

6.2. Minimum dimension transistors

Minimum dimension transistors help to further reduce power consumption, since they reduce circuit capacitance [13]. Throughout the processor, we use minimum dimension transistors, unless other considerations preclude their use. For example, transistors that are on critical delay paths often need to have larger dimensions to reduce delay [14]. Larger dimension transistors are also used to drive nodes with high fan-out and to balance circuit delays.

6.3. Delay balancing

Gates with unbalanced input delays can experience glitches, which increase dynamic switching power and dynamic short-circuit power [15]. To reduce glitches, we balance gate input delays in our circuits through a combination of gate-level delay balancing techniques (i.e., designing the circuits so that inputs to a particular gate go through roughly the same number of logic levels) and judicious transistor sizing. Glitches are further reduced by having a relatively small number of logic levels between pipeline registers.

6.4. Logic combining and input ordering

Dynamic and static power consumptions are also reduced by utilizing a variety of specially designed complex logic cells. Our circuits include efficient complex logic cells, such as 3-input AndOrInvert (AOI), 3-input OrAndInvert (OAI), half adder, and full adder cells. Providing a wide variety of complex gates with different drive strengths, functionality, and optionally inverted inputs gives circuit designers and synthesis tools greater flexibility to optimize for power consumption. Keeping nodes with a high probability of switching inside of complex gates and reordering the inputs to complex gates can help further reduce power. In general, inputs that are likely to be off are placed closer to gate output nodes, while inputs that are likely to be on are placed closer to the supply voltage [15].

7. SANDBLASTER SOFTWARE TOOLS

A *simulator* is an interpreter of a machine language where the representation of programs resides in memory but is not directly executed by host hardware. Historically, three types of architectural simulators have been identified. An *interpreted simulator* consists of a program executing on a computer where each machine language instruction is executed on a model of a target architecture running on the host computer. Because interpreted simulators tend to execute slowly, compiled simulators have been developed. A *statically compiled simulator* first translates both the program and the architecture model into the host computer's machine language. A *dynamically compiled* (or just-in-time) *simulator* either starts execution as an interpreter, but judiciously chooses functions that may be translated during execution into a directly executable host program, or begins by translating at the start of the host execution.

7.1. Interpreted execution

Instructions set simulators commonly used for application code development are cycle-count accurate in nature. They use an architecture description of the underlying processor and provide close-to-accurate cycle counts, but typically do not model external memories, peripherals, or asynchronous interrupts. However, the information provided by them is generally sufficient to develop the prototype application.

Figure 7 shows an interpreted simulation system. Executable code is generated for a target platform. During the execution phase, a software interpreter running on the host interprets (simulates) the target platform executable. The simulator models the target architecture, may mimic the implementation pipeline, and has data structures to reflect the

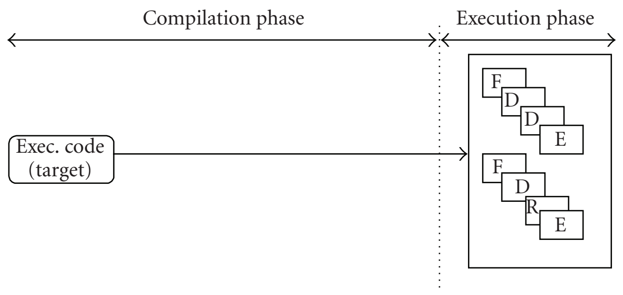

Figure 7: Interpreted simulation.

machine resources such as registers. The simulator contains a main driver loop, which performs the *fetch, decode, data read, execute, and write back* operations for each instruction in the target executable code.

An interpreted simulator has performance limitations. Actions such as instruction fetch, decode, and operand fetch are repeated for every execution of the target instruction. The instruction decode is implemented with a number of conditional statements within the main driver loop of the simulator. This adds significant simulation overhead because all combinations of opcodes and operands must be distinguished. In addition, the execution of the target instruction requires the update of several data structures that mimic the target resources, such as registers, in the simulator.

7.2. *Statically compiled simulation*

Figure 8 shows a statically compiled simulation system. In this technique, the simulator takes advantage of the any a priori knowledge of the target executable and performs some of the activities at compile time instead of execution time. Using this approach, a simulation compiler generates host code for instruction fetch, decode, and operand reads at compile time. As an end product, it generates an application-specific host binary in which only the execute phase of the target processor is unresolved at compile time. This binary is expected to execute faster, as repetitive actions have been taken care of at compile time.

While this approach addresses some of the issues with interpretive simulators, there are other limitations. First, the simulation compilers typically generate C code, which is then converted to object code using the standard *compile → assemble → link* path. Depending on the size of the generated C code, the file I/O needed to scan and parse the program could well reduce the benefits gained by taking the compiled simulation approach. The approach is also limited by the idiosyncrasies of the host compiler such as the number of labels allowed in a source file, size of switch statements and so forth. Some of these could be addressed by directly generating object code—however, the overhead of writing the application-specific executable file to the disc and then rereading it during the execution phase still exists. In addition, depending on the underlying host, the application-specific executable (which is visible to the user) may not be portable to another host due to different libraries, instruction sets and so forth.

7.3. *Dynamically compiled simulation*

Figure 9 shows the dynamically compiled simulation approach. This is the approach used in the Sandbridge simulator. In this approach, target instructions are translated into equivalent host instructions (executable code) at the beginning of execution time. The host instructions are then executed at the end of the translation phase. This approach eliminates the overhead of repetitive target instruction fetch, decode, and operand read in the interpretive simulation model. By directly generating host executable code, it eliminates the overhead of the compile, assemble, and link path and the associated file I/O that is present in the compiled simulation approach. This approach also ensures that the target executable file remains portable, as it is the only executable file visible to the user and the responsibility of converting it to host binary has been transferred to the simulator.

7.4. *Multithreaded programming model*

Obtaining full utilization of parallel processor resources has historically been a difficult challenge. Much of the programming effort can be spent determining which processors should receive data from other processors. Often execution cycles may be wasted for data transfers. Statically scheduled machines such as Very Long Instruction Word architectures and visible pipeline machines with wide execution resources complicate programming and may reduce programmer productivity by requiring manual tracking of up to 100 in-flight instruction dependencies. When nonassociative DSP arithmetic is present, nearly all compilers are ineffective and the resulting burden falls upon the assembly language programmer. A number of these issues have been discussed in [8].

A good programming model should adequately abstract most of the programming complexity so that 20% of the effort may result in 80% of the platform utilization [16]. While there are still some objections to a multithreaded programming model [9], to-date it is widely adopted particularly with the introduction of the Java programming language [17].

With hardware that is multithreaded with concurrent execution and adopting a multithreaded software programming model, it is possible for a kernel to be developed that automatically schedules software threads onto hardware threads. It should be noted that while the hardware scheduling is fixed and uses a technique called token triggered threading (T^3) [18], the software is free to use any scheduling policy desired.

The Sandblaster kernel has been designed to use the POSIX pthreads open standard [19]. This provides cross platform capability as the library is compilable across a number of systems including Unix, Linux, and Windows.

7.5. *Compiler technology*

There are many challenges faced when trying to develop efficient compilers for parallel DSP technologies. At each level of processor design, Sandbridge has endeavored to alleviate these issues through abstraction. First and foremost,

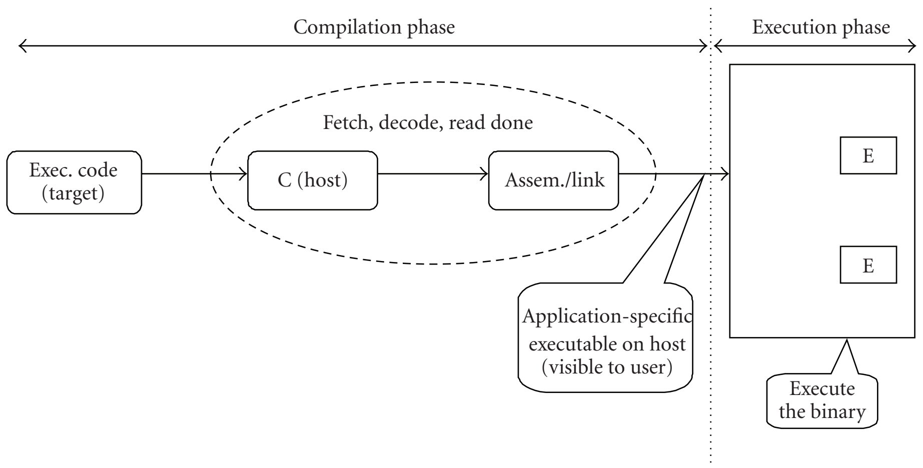

Figure 8: Statically compiled simulation.

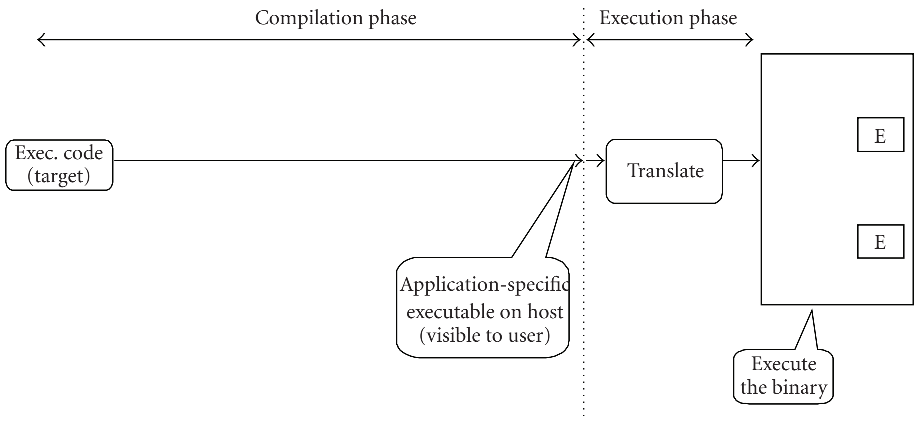

Figure 9: Dynamically compiled simulation.

the Sandblaster processor is transparent in the architectural sense. This proscribes that there are no visible implementation effects for the programmer or compiler to deal with [2]. This is in distinct contrast with VLIW designs where the implementation strongly influences the architecture. A benefit of a true architecture approach is that object code will execute unmodified (e.g., without any translation required) on any Sandblaster compliant implementation.

The Sandblaster architecture uses a SIMD datapath to implement vector operations. The compiler vectorizes C code to exploit the data level parallelism inherent in signal processing applications and then generates the appropriate vector instructions. The compiler also handles the difficult problem of outer loop vectorization

Within the architecture, there is direct support for parallel saturating arithmetic. Since saturating arithmetic is nonassociative, out-of-order execution may produce different bit results. In some wireless systems this is not permissible [20]. By architecting parallel saturating arithmetic (i.e., vector multiply and accumulate with saturation), the compiler is able to generate code with the understanding that the hardware will properly produce bit-exact results. The compiler algorithm used to accomplish this is described in [21]. Some hardware techniques to implement this are described in [22].

Additionally, our compiler can also automatically generate threads. We use the same pthreads mechanism for thread generation in the compiler as the programmer who specifies them manually. For most signal processing loops, it is not a problem to generate threads and the compiler will automatically produce code for correct synchronization.

7.6. *Tool chain generation*

Figure 10 shows the Sandblaster tool chain generation. The platform is programmed in a high-level language such as C, C++, or Java. The program is then translated using an internally developed supercomputer class vectorizing parallelizing compiler. The tools are driven by a parameterized resource model of the architecture that may be programmatically generated for a variety of implementations and organizations. The source input to the tools, called the Sandbridge

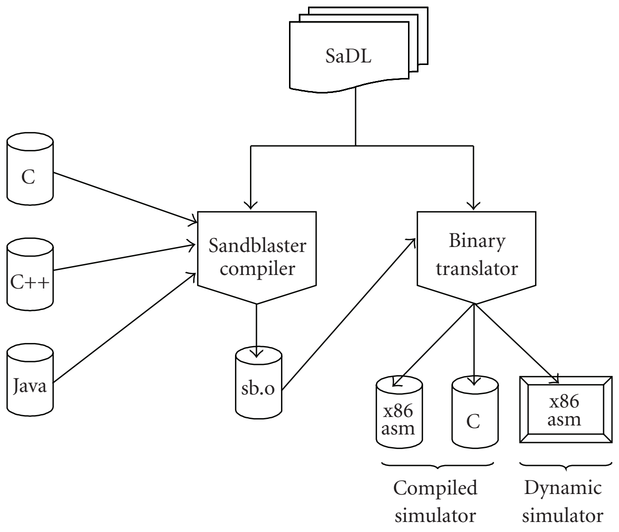

Figure 10: Tool chain generation.

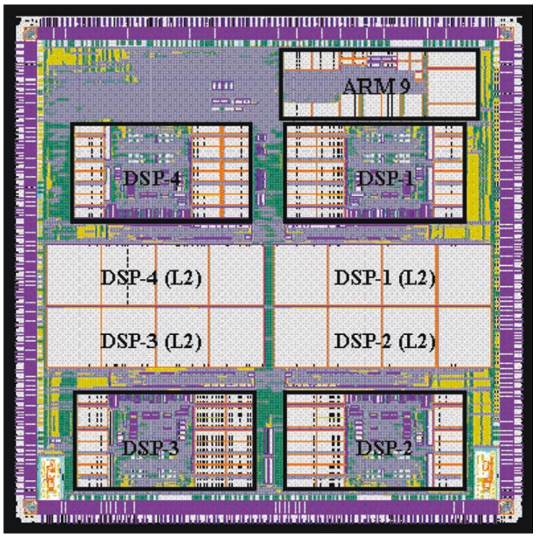

Figure 11: SB3011 device layout.

architecture description language (SaDL), is a collection of python source files that guide the generation and optimization of the input program and simulator. The compiler is retargetable in the sense that it is able to handle multiple possible implementations specified in SaDL and produce an object file for each implementation. The platform also supports many standard libraries (e.g., libc, math, etc.) that may be referenced by the C program. The compiler generates an object file optimized for the Sandblaster architecture.

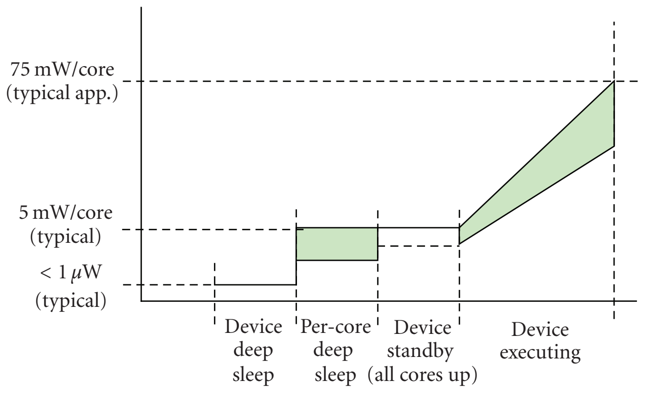

Figure 12: Processor power results for a 600 MHz 0.9 V Sandblaster device.

8. RESULTS

This section discusses the performance and power results for the processor, the simulation and compilation performance results, and finally full communications systems results.

8.1. Processor performance and power results

Figure 11 shows a picture of the SB3011 chip which was fabricated in 90 nm TSMC technology. Highlighted are the 4 Sandblaster cores, the ARM9 core, and the L2 memories. Initial samples have performed at 600 MHz at 0.9 V.

Figure 12 shows power measurements made on the initial samples for a single Sandblaster core. As described in Section 2.5, the power modes may be programmed. Figure 12 shows power at some typical configurations. When the entire device is in deep sleep it consumes less than 1 microwatt of power. As you bring each core out of deep sleep to a standby state, there is a measured range of power dissipation which on the initial samples is less than 5 milliwatts with complete state retention. The last section of Figure 12 depicts the linear nature of programs executing. Depending on the core activity, the power dissipation is linear with respect to the workload. The linear nature depicted is the result of average utilization of threads. We have measured on hardware a range of applications. WCDMA dissipates about 75 mW per core (at 600 MHz 0.9 V). Other less demanding applications such as GSM/GPRS dissipate less power.

8.2. Processor tools results

Figure 13 shows the results of various compilers on out-of-the-box ETSI C code [20]. The y-axis shows the number of MHz required to compute frames of speech in real-time. The AMR code is completely unmodified and no special include files are used. Without using any compiler techniques such as intrinsics or special typedefs, the compiler is able to achieve real-time operation on the baseband core at hand-coded assembly language performance levels. Note that the program is completely compiled from C language code. Since other solutions are not able to automatically generate DSP operations, intrinsic libraries must be used. With intrinsic libraries the results for most DSPs are near ours but they only apply to the ETSI algorithms whereas the described compiler can be applied to arbitrary C code.

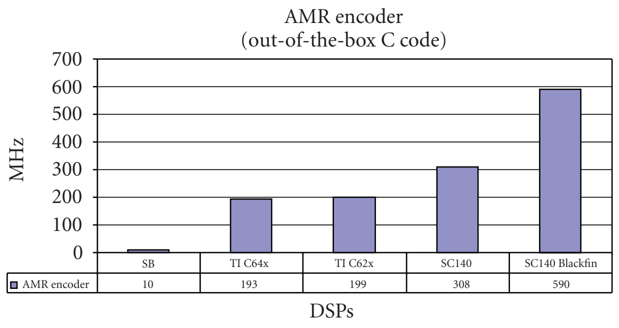

Figure 13: Out-of-the-box AMR ETSI encoder C code results. (Results based on out-of-the-box C code. C64x IDE Version 2.0.0 compiled without intrinsics using -k -q -pm -op2 -o3 -d"WMOPS = 0" -ml0 -mv6400 flags with results averaged over 425 frames of ETSI-supplied test vectors. C62x IDE Version 2.0.0 compiled without intrinsics using -k -q -pm -op2 -o3 -d"WMOPS = 0" -ml0 -mv6200 flags with results averaged over 425 frames of ETSI-supplied test vectors. Starcore SC140 IDE version Code Warrior for StarCore version 1.5, relevant optimization flags (encoder only): scc -g -ge -be -mb -sc -O3 –Og, other: no intrinsic used. Results based on execution of 5 frames. ADI Blackfin IDE Version 2.0 and Compiler version 6.1.5 compiled without intrinsics using -O1 -ipa -DWMOPS = 0 –BLACKFIN with results averaged over 5 frames of ETSI-supplied test vectors for the encoder only portion.)

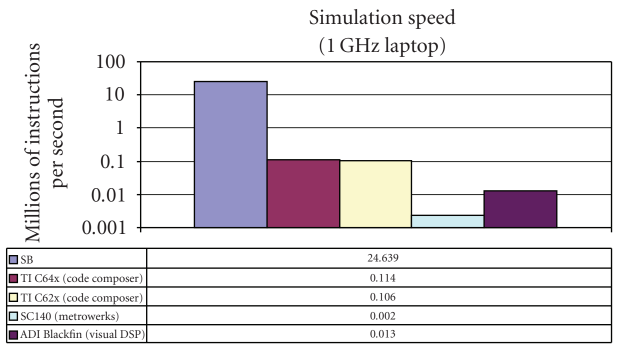

Figure 14: Simulation speed of ETSI AMR encoder.

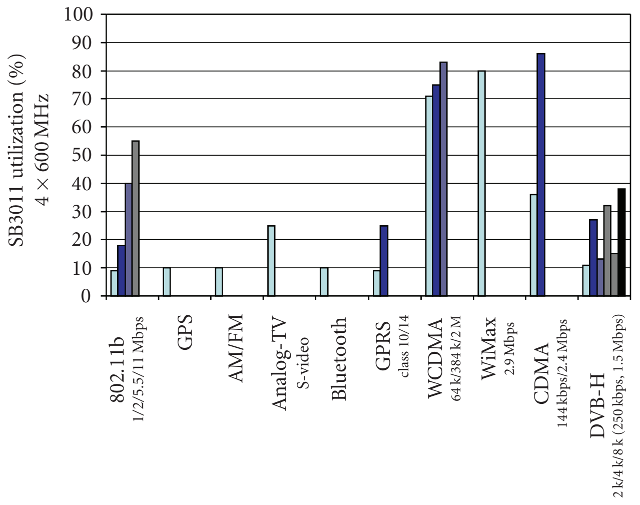

Figure 15: Communication systems results as a percentage of SB3011 utilization (4 Cores at 600 MHz).

Efficient compilation is just one aspect of software productivity. Prior to having hardware, algorithm designers should have access to fast simulation technology. Figure 14 shows the postcompilation simulation performance of the same AMR encoder as Figure 13 for a number of DSP processors. All programs were executed on the same 1 GHz laptop Pentium computer. The Sandbridge tools are capable of simulating 24.6 million instructions per second. This is more than two orders of magnitude faster than the nearest DSP and allows real-time execution of GSM speech coding on a Pentium simulation model. To further elaborate, while some DSPs cannot even execute the out-of-the-box code in real-time on their native processor, the Sandbridge simulator achieves multiple real-time channels on a simulation model of the processor. This was accomplished by using internal compilation technology to accelerate the simulation.

8.3. Applications results

Figure 15 shows the results of a number of communications systems as a percentage utilization of a 4-core 600 MHz SB3011 platform. Particularly, WiFi 802.11b, GPS, AM/FM radio, Analog NTSC Video TV, Bluetooth, GSM/GPRS, UMTS WCDMA, WiMax, CDMA, and DVB-H. A notable point is that all these communications systems are written in generic C code with no hardware acceleration required. It is also notable that performance in terms of data rates and concurrency in terms of applications can be dynamically adjusted based on the mix of tasks desired. For most of the systems, the values are measured on hardware from digitized RF signals that have been converted in real-time. This includes the design of RF cards based on industry standard components. The only exceptions are Bluetooth and DVB-H. For these systems the RF cards are still under development.

Figure 16 shows the results of various multimedia codecs. Note that the total MHz scale is 7% of the entire 4-core capacity. Results for QCIF (176 × 144) at 15 frames per second (fps) and CIF (360 × 288) at 30 fps images are shown for the H.264 decoder. For the MPEG4 decoder, out-of-the-box (OOB) and optimized (OPT) are shown for the Foreman clip at 30 frames per second. Noticeably, out-of-the-box performance is real-time and highly efficient. This is the result of our highly optimizing compiler which can both vectorize and parallelize standard C code. Also, MP3 decoding results are shown at various bit rates. A key point is that all these applications run using less than two threads and many in a percentage of a single thread. Since there are 32 threads in the SB3011 implementation, a single thread consumes 3.125% of the available processor performance.

Figure 17 shows measurements while executing either GPRS class 14 or WCDMA at a 384 kbps bit rate. Note that both of these applications dissipate less power than the stated average dissipation of 75 mW per core. The actual power

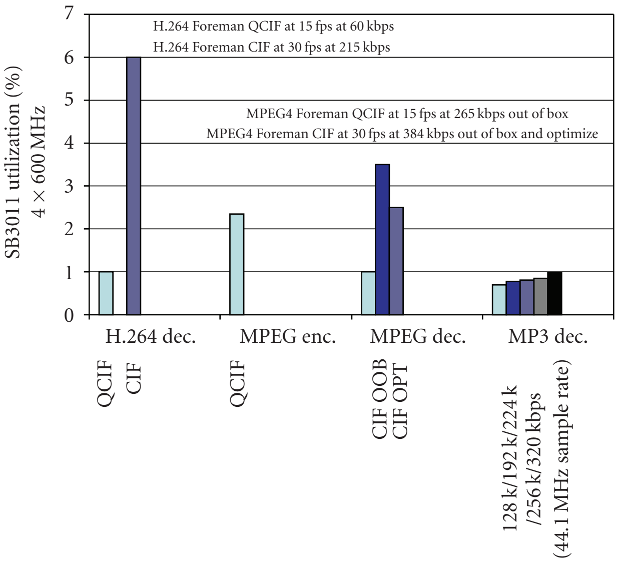

Figure 16: Multimedia results as a percentage of SB3011 utilization (4 cores at 600 MHz).

dissipation is highly dependent upon workload. As an approximation it may be possible to use the average utilization of the processor complex multiplied by the average power. However, in practice the actual results vary significantly by application. The SB3010 measurements refer to an earlier version of the chip that was predominantly synthesized. The SB3011 is a semicustom design. The software-optimized column refers to the operating system's ability to turn off cores and threads that are unused. This can result in significant power savings.

9. RELATED WORK

In this section we contrast and compare our approach for both processors and tools with other known approaches. The Sandbridge processor design provides important power and performance characteristics while the tools provide the capability of rapidly designing SDR systems.

9.1. Processors

Other SDR platforms include the Signal Processing on Demand Architecture (SODA) [23], OnDSP [4], the Embedded Vector Processor (EVP) [5], the Synchronous Transfer Architecture (STA) [6], picoArray [24], XiSystem [25], and the MS1 reconfigurable DSP (rDSP) core [26].

SODA is a programmable SDR platform that consists of four processor cores. Each core contains scratchpad memories and asymmetric pipelines that support scalar, 32-wide SIMD, and address generation operations. SODA is optimized for 16-bit arithmetic and features several specialized operations including saturating arithmetic, vector permute, vector compare and select, and predicated negation operations.

OnDSP, EVP, and STA all are VLIW architectures with support for multiple parallel scalar, vector, memory access, and control operations. For example, OnDSP provides 8-element vector operations that can operate in parallel with scalar operations. With EVP, the maximum VLIW-parallelism available is five vector operations, four scalar operations, three address updates, and loop-control. All three architectures feature dedicated instructions for wireless communications algorithms, such as FFTs and Viterbi, Reed-Solomon, and Turbo coding. STA utilizes a machine description file to facilitate the generation of different hardware and simulation models for the processor.

picoArray is a tiled architecture in which hundreds of heterogeneous processors are interconnected using a bus-based array. Within the picoArray, processors are organized in a two-dimensional grid, and communicate over a network of 32-bit unidirectional buses and programmable bus switches. Each programmable processor in the array supports 16-bit arithmetic, uses 3-way VLIW scheduling, and has its own local memory. In addition to the programmable processors, the picoArray includes specialized peripherals and connects to hardware accelerators for performing FFTs, cryptography, and Reed-Solomon and Viterbi coding.

XiSystem and the MS1 rDSP core combine programmable processors with reconfigurable logic to implement wireless communication systems. XiSystem integrates a VLIW processor, a multicontext reconfigurable gate array, and reconfigurable I/O modules in a SoC platform. The multicontext reconfigurable gate array enables dynamic instruction set extensions for bit-level operations needed in many DSP applications. The MS1 rDSP core contains a reconfigurable logic block, called the RC Array, a 32-bit RISC processor, called mRISC, a context memory, a data buffer, and an I/O controller. The mRISC processor controls the RC array, which performs general purpose operations, as well as word-level and bit-level DSP functions.

Unlike other SDR platforms, the SB3011 platform provides a fully programmable solution in which all communications systems are written in generic C code with no hardware acceleration or assembly language programming. It also is the first SDR platform to combine explicit multithreading, powerful compound instructions, vector operations, and parallel saturating arithmetic in a low-power programmable SoC multiprocessor.

Another important aspect of the SB3011 platform is the technique it uses to support explicit multithreading. Previous techniques for explicit hardware multithreading including interleaved multithreading (IMT), blocked multithreading (BMT), and simultaneous multithreading (SMT) [27]. With IMT [28], also known as fine grain multithreading or horizontal multithreading, only one thread can issue an instruction each cycle, and threads issue instructions in a predetermined order (e.g., round-robin). With BMT [29], also known as coarse-grain multithreading or vertical multithreading, instructions are executed sequentially until a long-latency event (e.g., a cache miss) occurs. The long-latency event triggers a fast context switch to another thread. With SMT [30], multiple instructions may be issued each cycle

	GSM/GPRS class 14, 1 core			WCDMA at 384 kbps, 1 of 3 cores		
	SB3010 (mW) Measured	SB3010 (mW) SW optimized	SB3011 (mW)	SB3010 (mW) Measured	SB3010 (mW) SW optimized	SB3011 (mW)
Core w/ L1 instances 32 KB I-cache 64 KB D mem.	150	85	45	171	130	65
Core w/o L1 instances	142	77	40	160	117	58

Figure 17: Application power measurements.

from multiple threads. SMT combines techniques from previous multithreaded processors and dynamically scheduled superscalar processors to exploit both instruction-level parallelism and thread-level parallelism.

As discussed in Section 4, The SB3011 features a new form of interleaved multithreading, known as token triggered threading (T^3). Unlike previous IMT implementations, the T^3 implementation on the SB3011 features compound instructions, SIMD vector operations, and greater flexibility in scheduling threads. Compared to BMT, T^3 provides greater concurrency since instructions from multiple threads are executing in parallel each cycle. Compared to SMT, T^3 has much less hardware complexity and power dissipation, since the method for selecting threads is simplified, only a single compound instruction issues each clock cycle, and dependency checking and bypass hardware are not needed. The SB3011 platform combines T^3 with chip multiprocessing to provide up to 32 simultaneously executing hardware threads.

9.2. *Tools*

In this section, we compare our solution to other high-performance tools solutions. Automatic DSP simulation generation from a C++-based class library was discussed in [31]. Automatic generation of both compiled and interpretive simulators was discussed in [32]. Compiled simulation for programmable DSP architectures to increase simulation performance was introduced in [33]. This was extended to cycle accurate models of pipelined processors in [34]. A general purpose MIPS simulator was discussed in [35]. The ability to dynamically translate snippets of target code to host code at execution time was used in Shade [36]. However, unlike Shade, our approach generates code for the entire application, is targeted towards compound instruction set architectures, and is capable of maintaining bit exact semantics of DSP algorithms. A similar approach to ours is described in [37].

10. SUMMARY

Sandbridge Technologies has introduced a completely new and scalable design methodology for implementing multiple communications systems on a single SDR chip. Using a unique multithreaded architecture specifically designed to reduce power consumption, efficient broadband communications operations are executed on a programmable platform. The instruction execution in the described architecture is completely interlocked providing software compatibility among all processors. Because of the interlocked execution, interrupt latency is very short. An interrupt may occur on any instruction boundary including loads and stores; this is critical for real-time systems.

The processor is combined with a highly optimizing vectorizing compiler with the ability to automatically analyze programs and generate DSP instructions. The compiler also automatically parallelizes and multithreads programs. This obviates the need for assembly language programming and significantly accelerates time-to-market for streaming multimode multimedia convergence systems.

REFERENCES

[1] http://www.sdrforum.org/.

[2] G. Blaauw and F. Brooks Jr., *Computer Architecture: Concepts and Evolution*, Addison-Wesley, Reading, Mass, USA, 1997.

[3] B. Case, "Philips hopes to displace DSPs with VLIW," *Microprocessor Report*, vol. 8, no. 16, pp. 12–15, 1997.

[4] J. Kneip, M. Weiss, W. Drescher, et al., "Single chip programmable baseband ASSP for 5 GHz wireless LAN applications," *IEICE Transactions on Electronics*, vol. E85-C, no. 2, pp. 359–367, 2002.

[5] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 16, pp. 2613–2625, 2005.

[6] J. P. Robelly, G. Cichon, H. Seidel, and G. Fettweis, "A HW/SW design methodology for embedded SIMD vector signal processors," *International Journal of Embedded Systems*, vol. 1, no. 11, pp. 2–10, 2005.

[7] J. Glossner, T. Raja, E. Hokenek, and M. Moudgill, "A multithreaded processor architecture for SDR," *The Proceedings of the Korean Institute of Communication Sciences*, vol. 19, no. 11, pp. 70–84, 2002.

[8] J. Glossner, M. Schulte, M. Moudgill, et al., "Sandblaster low-power multithreaded SDR baseband processor," in *Proceedings of the 3rd Workshop on Applications Specific Processors (WASP '04)*, pp. 53–58, Stockholm, Sweden, September 2004.

[9] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, 2006.

[10] J. Glossner, K. Chirca, M. Schulte, et al., "Sandblaster low power DSP," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '04)*, pp. 575–581, Orlando, Fla, USA, October 2004.

[11] B. Moyer, "Low-power design for embedded processors," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1576–1587, 2001.

[12] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[13] A. Wroblewski, O. Schumacher, C. V. Schimpfle, and J. A. Nossek, "Minimizing gate capacitances with transistor sizing," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '01)*, vol. 4, pp. 186–189, Sydney, NSW, Australia, May 2001.

[14] M. Borah, R. M. Owens, and M. J. Irwin, "Transistor sizing for minimizing power consumption of CMOS circuits under delay constraint," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 167–172, Dana Point, Calif, USA, April 1995.

[15] S. Kim, J. Kim, and S.-Y. Hwang, "New path balancing algorithm for glitch power reduction," *IEE Proceedings: Circuits, Devices and Systems*, vol. 148, no. 3, pp. 151–156, 2001.

[16] R. Goering, "Platform-based design: a choice, not a panacea," *EE Times*, 2002, http://www.eetimes.com/story/OEG2002091-1S0061.

[17] O. Silvén and K. Jyrkkä, "Observations on power-efficiency trends in mobile communication devices," in *Proceedings of the 5th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '05)*, vol. 3553 of *Lecture Notes in Computer Science*, pp. 142–151, Samos, Greece, July 2005.

[18] M. Schulte, J. Glossner, S. Mamidi, M. Moudgill, and S. Vassiliadis, "A low-power multithreaded processor for baseband communication systems," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation*, vol. 3133 of *Lecture Notes in Computer Science*, pp. 393–402, Springer, New York, NY, USA, 2004.

[19] B. Nichols, D. Buttlar, and J. Farrell, *Pthreads Programming: A POSIX Standard for Better Multiprocessing*, O'Reilly Nutshell Series, O'Reilly Media, Sebastopol, Calif, USA, 1996.

[20] K. Jarvinen, J. Vainio, P. Kapanen, et al., "GSM enhanced full rate speech codec," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '97)*, vol. 2, pp. 771–774, Munich, Germany, April 1997.

[21] V. Kotlyar and M. Moudgill, "Detecting overflow detection," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '04)*, pp. 36–41, Stockholm, Sweden, September 2004.

[22] P. I. Balzola, M. Schulte, J. Ruan, J. Glossner, and E. Hokenek, "Design alternatives for parallel saturating multioperand adders," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '01)*, pp. 172–177, Austin, Tex, USA, September 2001.

[23] Y. Lin, H. Lee, M. Woh, et al., "SODA: a low-power architecture for software radio," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA '06)*, pp. 89–100, Boston, Mass, USA, June 2006.

[24] A. Lodi, A. Cappelli, M. Bocchi, et al., "XiSystem: a XiRisc-based SoC with reconfigurable IO module," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 85–96, 2006.

[25] A. Duller, G. Panesar, and D. Towner, "Parallel processing - the picoChip way!," in *Communicating Process Architectures (CPA '03)*, pp. 125–138, Enschede, The Netherlands, September 2003.

[26] B. Mohebbi, E. C. Filho, R. Maestre, M. Davies, and F. J. Kurdahi, "A case study of mapping a software-defined radio (SDR) application on a reconfigurable DSP core," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 103–108, Newport Beach, Calif, USA, October 2003.

[27] T. Ungerer, B. Robič, and J. Šilc, "A survey of processors with explicit multithreading," *ACM Computing Surveys*, vol. 35, no. 1, pp. 29–63, 2003.

[28] B. J. Smith, "The architecture of HEP," in *Parallel MIMD Computation: HEP Supercomputer and Its Applications*, J. S. Kowalik, Ed., pp. 41–55, MIT Press, Cambridge, Mass, USA, 1985.

[29] T. E. Mankovic, V. Popescu, and H. Sullivan, "CHoPP principles of operations," in *Proceedings of the 2nd International Supercomputer Conference*, pp. 2–10, Mannheim, Germany, May 1987.

[30] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA '95)*, pp. 392–403, Santa Margherita Ligure, Italy, June 1995.

[31] D. Parson, P. Beatty, J. Glossner, and B. Schlieder, "A framework for simulating heterogeneous virtual processors," in *Proceedings of the 32nd Annual Simulation Symposium*, pp. 58–67, San Diego, Calif, USA, April 1999.

[32] R. Leupers, J. Elste, and B. Landwehr, "Generation of interpretive and compiled instruction set simulators," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '99)*, vol. 1, pp. 339–342, Wanchai, Hong Kong, January 1999.

[33] V. Zivojnovic, S. Tjiang, and H. Meyr, "Compiled simulation of programmable DSP architectures," in *Proceedings of the IEEE Workshop on VLSI Signal Processing*, pp. 187–196, Osaka, Japan, October 1995.

[34] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA—machine description language for cycle-accurate models of programmable DSP architectures," in *Proceedings of the 36th Annual Design Automation Conference (DAC '99)*, pp. 933–938, New Orleans, La, USA, June 1999.

[35] J. Zhu and D. D. Gajski, "An ultra-fast instruction set simulator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 363–373, 2002.

[36] R. Cmelik and D. Keppel, "Shade: a fast instruction-set simulator for execution profiling," Tech. Rep. UWCSE 93-06-06, University of Washington, Washington, DC, USA, 1993.

[37] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr, and A. Hoffmann, "Design innovations for embedded processors: a universal technique for fast and flexible instruction-set architecture simulation," in *Proceedings of the 39th Design Automation Conference (DAC '02)*, pp. 22–27, ACM Press, New Orleans, La, USA, June 2002.