Future Wireless Convergence Platforms

John Glossner, Ph.D., EVP & CTO
Mayan Moudgill, Ph.D., Chief Architect
Daniel Iancu, Ph.D., Chief Communications Architect
Gary Nacer, VP of Engineering
Sanjay Jintukar, Ph.D., Director of Software
Stuart Stanley, Director of RF Engineering
Michael Samori, Director of Hardware Design
Tanuj Raja, VP of Business Development
Michael Schulte, Ph.D., Professor, UW Madison
Stamatis Vassiliadis, Ph.D., Professor, TU Delft

Sandbridge Technologies, Inc.
White Plains, NY
914-287-8500
glossner@sandbridgetech.com

ABSTRACT

As wireless platforms converge to multimedia systems, architectures must converge to support voice, data, and video applications. From a processor architecture perspective, support for signal processing (both audio and video), control code, and Java execution will be required in a convergent device. Traditionally, wireless communications systems have been implemented in hardware. Convergent devices must be able to roam seamlessly across multiple communications systems. To avoid excessive hardware costs, a Software Defined Radio (SDR) approach offers a programmable and dynamically reconfigurable method of reusing hardware to implement physical layer processing. In this paper, we discuss trends in wireless platforms which are inherently convergence platforms. We also present the Sandbridge state-of-the-art example platform that supports both communications and multimedia applications processing. The architecture efficiently executes Java, Digital Signal Processing (DSP), and control code. Architectural features that reduce power dissipation and enable real-time processing are described. All of the communications and multimedia processing is executed completely in software without specialized hardware support. The processor is programmed in C with supercomputer-class compiler support for automatic vectorization, multithreading, and DSP semantic analysis.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]
C.1.4 [**Parallel Architectures**]

General Terms: Design, Performance

Keywords

Software Defined Radio, Wireless, Digital Signal Processors

1. INTRODUCTION

From the end-user point of view, a modern communications device has a color screen, a keyboard, an antenna, audio, and video. All these features require high computing capability at low power consumption. The performance requirements for mobile wireless communication devices have expanded dramatically from their inception as mobile telephones. Consumers are demanding convergence devices with full data and voice integration as well as a variety of computationally intense features and applications such as web browsing, MP3 audio, and MPEG4 video. Moreover, consumers want these wireless subscriber services to be accessible at all times anywhere in the world.

The technologies necessary to realize true broadband wireless handsets and systems present unique design challenges. Wireless handset manufacturers are challenged to deliver products that offer expanded services and operate transparently worldwide. Product designers are challenged to create extremely power efficient yet high-performance, broadband wireless devices. The design tradeoffs and implementation options inherent in meeting these demands highlight the extremely challenging requirements of next generation convergence processors.

Power dissipation constraints are requiring new techniques at every stage of design - architecture, microarchitecture, software, algorithm design, logic design, circuit design, and process design. With performance requirements exploding as bandwidth demand increases, power conscious design becomes more difficult. System-on-a-chip (SOC) integration and low voltage process technologies will contribute to lower power SOC integrated circuits (ICs), but are insufficient as the only solution for streaming broadband applications.

Convergence applications are fundamentally DSP applications. A large number of standards exist or have been proposed for the wireless and wired communication markets. Such a diversity of standards necessitates a programmable platform for their timely implementation. Traditional communications systems have typically been implemented using custom hardware solutions. Chip rate, symbol rate, and bit rate co-processors are often coordinated by programmable DSPs but the DSP processor does not typically participate in computationally intensive tasks. Even with a single communication system, the hardware development cycle is onerous, often requiring multiple chip redesigns late into the certification process. When multiple communications systems requirements are considered, both silicon area and design validation are major inhibitors to commercial success. Therefore, a software-based platform capable of dynamically reconfiguring communications systems enables elegant reuse of silicon area and dramatically reduces time to market by allowing multi-protocol support through software modifications instead of time consuming hardware redesigns. Such platforms have begun forming the basis for Software Defined Radio (SDR).

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 a reconfigurable SDR solution 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 etc., 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, remotely reconfigurable protocols provide simple and inexpensive software version control and feature upgrades. This 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. Sixth, any defects found in the field can be fixed by changing the software, possibly even transparently to the user, without requiring a hardware change or a chip respin.

In this paper we discuss trends in wireless platforms, which are inherently convergence platforms. We also present the Sandbridge state-of-the-art example platform that supports both communications and multimedia applications processing. The architecture efficiently executes Java, Digital Signal Processing (DSP), and control code. Architectural features that reduce power dissipation and enable real-time processing are described. All of the communications and multimedia processing is executed completely in software without specialized hardware support. The processor is programmed in C with supercomputer-class compiler support for automatic vectorization, multithreading, and DSP semantic analysis.

2. HISTORICAL BACKGROUND

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 non-transparent 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][4][5][6].

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 instruction dependencies between upwards of 100 in-flight instructions is a non-trivial task for a programmer. This is exactly the area where a compiler excels.

A challenge of using VLIW processors includes 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 requires 4 instructions, 128 bits, to specify 4 operations. A vector encoding may compute many more operations in as little as 21 bits (for example – multiply two 4-element vectors, saturate, accumulate, and saturate).

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

A challenge of visible pipeline machines (e.g. most DSPs and VLIW processors) is interrupt response latency. Visible memory pipeline effects in 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 a mix of computational calculations 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 as one or more multiply accumulate (MAC) units. In addition, as the saturating arithmetic is non-associative, parallel execution of 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 to the convergence processors. 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 minimizes both hardware and software design challenges inherent in real-time streaming convergence applications.

3. PROGRAMMING ENVIRONMENT

Programmer productivity is also a major concern in streaming multimedia DSP and SDR convergence applications. Because most classical DSPs are programmed in assembly language, it takes a very large software effort to program an application. As an example, with modern speech coders it may take up to nine months or more before the application performance is known if they are coded in assembly language. Then, an intensive period of design verification ensues. If efficient high-level language compilers for DSPs are available, significant increases in software productivity and programming effort can be achieved.

A DSP compiler should be designed jointly with the architecture based on the intended application domain. Trade-offs are made between the architecture and compiler subject to the application performance, power, and price constraints. However, there are a number of issues that must be addressed in designing a DSP compiler. First, there is a fundamental mismatch between DSP datatypes and C language constructs. A basic datatype in DSPs is a saturating fractional fixed-point representation. C language constructs, however, define integer modulo arithmetic. This forces the programmer to explicitly program saturation operations. A DSP compiler must deconstruct and analyze the C code for the semantics of the operations represented and generate the underlying fixed point operations.

A second problem for compilers is that previous DSP architectures were not designed with compilability as a goal. To maintain minimal code size, multiple operations were issued from the same compound instruction. To reduce instruction storage, a common encoding was 16-bits for all instructions. Often, three operations could be issued from the same 16-bit instruction. While this is good for code density, orthogonality[1] suffered. Classical DSPs imposed many restrictions on the combinations of operations and the dense encoding implied many special purpose registers. This resulted in severe restrictions for the compiler and poor code generation.

Early attempts to remove these restrictions used VLIW instruction set architectures with nearly full orthogonality. To issue four multiply accumulates requires at least four instructions (with additional load instructions to sustain throughput). This generality was required to give the compiler technology an opportunity to catch up with assembly language programmers.

Because DSP C compilers have difficulty generating efficient code, extensions have been introduced to high level languages [7]. Typical additions may include special support for 16-bit datatypes (Q15 formats), saturating types, multiple memory spaces, and SIMD parallel execution. These additions often imply a special compiler and the code generated may not be emulated easily on multiple platforms. As a result, special language constructs have not been successful.

Libraries

Due to the programming burden of traditional DSPs, large libraries are typically built up over time. Often more than 1000 functions are provided, including FIR filters, FFTs, convolutions, DCTs, and other computationally intensive kernels. The software burden to generate libraries is high but they can be reused for many applications. With this approach, control code can be programmed in C and the computationally intensive signal processing functions are called through these libraries.

Intrinsic Functions

Often, when programming in a high-level language such as C, a programmer would like to take advantage of a specific instruction available in an architecture but there is no mechanism for describing that instruction in C. For this case intrinsics were developed. In their rudimentary form, an intrinsic is an *asm* statement such as found in GCC.

An intrinsic function has the appearance of a function call in C source code, but is replaced during pre-processing by a programmer-specified sequence of lower-level instructions. The replacement specification is called the intrinsic substitution or simply the intrinsic. An intrinsic function is defined if an intrinsic

[1] Orthogonality is a property of instruction set architectures that allows any operation to be specified with any combination of other operations.

substitution specifies its replacement. The lower-level instructions resulting from the substitution are called intrinsic instructions [8].

Intrinsics are used to collapse what may be more than ten lines of C code into a single DSP instruction. A typical math operation from the ETSI GSM EFR speech coder, L_add, is given as:

```
/* GSM ETSI Saturating Add */
Word32 L_add( Word32 a, Word32 b ) {
        Word32 c;
        c = a + b;
        if ((( a^b ) & MIN_32 ) == 0) {
                if (( c^a) & MIN_32 ) {
                  c = (a < 0) ? MIN_32 : MAX_32;
                }
        }
        return( c );
}
```

Many DSPs use intrinsics to implement the L_add operation as a single instruction. Early intrinsic efforts, like inlined asm statements, inhibited DSP compilers from optimizing code sequences [8]. A DSP C compiler could not distinguish the semantics and side effects of the assembly language constructs and this resulted in compiler scheduling hazards. Other solutions, which attempted to convey side-effect free instructions, have been proposed. These solutions all introduced architectural dependent modifications to the original C source.

Intrinsics which eliminated these barriers have been explored [8]. The main technique is to represent the operation in the intermediate representation of the compiler. With the semantics of each intrinsic well known to the intermediate format, optimizations with the intrinsic functions were easily enabled yielding speedups of more than 6x.

The main detractor of intrinsics is that it moves the assembly language programming burden to the compiler writers. More importantly, each new application may still need a new intrinsic library. This further constrains limited software resources.

High-level DSP Compilation

The above discussion focused on source-level semantic mismatches between C code and DSP operations. The solutions in the industry are not ideal. However, even after providing compiler solutions for the semantic gap, there is still the difficult challenge of implementing supercomputer-class optimizations in the compiler.

In addition to classic compiler optimizations [10], there are some advanced optimizations which have proven significant for DSP applications. Software pipelining [11] in combination with aggressive inlining has proven effective in extracting the parallelism inherent in DSP applications. Interestingly, some DSP applications (speech coding for example) do not exhibit significant data dependence. A program that is data dependent will give significantly different execution times and execution paths through the program depending upon what data input the program receives. When programs are not heavily influenced by the dataset choice, profile directed optimizations may be effective at improving performance [12]. In profile driven optimization, the program is executed based on a set of data inputs. The results of the program and the execution path through the program are then fed back into the compiler. The compiler uses this information to group highly traversed paths into larger blocks of code which can then be optimized and parallelized. These techniques, when used with VLIW scheduling [13], have proven effective in DSP compilation. However, they still can be more than two times less efficient than assembly language programming.

Another challenge DSP compiler writers face is parallelism extraction. Early VLIW machines alleviated the burden from the compiler by allowing full orthogonality of instruction selection. Unfortunately this led to code-bloat. General purpose machines have recognized the importance of DSP operations and have provided specialized SIMD instruction set extensions (e.g. MMX/SSE, Altivec, VIS). Unfortunately, compiler technology has not been effective in exploiting these instruction set extensions, and library functions are often the only efficient way to invoke them.

Exploiting data parallelism is an important factor in optimizing for DSP applications. While both a VLIW and vector datapath can exploit such parallelism, extracting it from C code can be a difficult challenge. Most VLIW scheduling techniques focus on exploiting instruction level parallelism from code sequences. However, they do not exploit data parallelism. To do this, vectorizing compilers are needed. For a compiler to be able to vectorize loops coded in C, it may have to significantly reorder the loops either splitting or jamming them together, depending on the nesting depth. These techniques are typically only found in supercomputer compilers but they greatly assist in uncovering data parallelism from arbitrary C code.

4. SANDBRIDGE SDR SOLUTION

Sandbridge Technologies has designed an SDR processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [17][18]. The Sandbridge Sandblaster® design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and very low power dissipation.

The microarchitecture of the Sandblaster® processor is multithreaded and all 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 unique form of multithreading called Token Triggered Threading (T^3), which consumes much less power than other form of multithreading.

The Sandbridge tools implement all of the expected standard optimizations but also extend the optimizations into areas that were previously only explored by supercomputing designers. Since applications are growing at more than a compounded 44% per year in terms of the number of lines of C code required, it is no longer feasible to consider building new systems from libraries or assembly coding techniques. The Sandbridge programming paradigm represents the first true standard high-level language compilable platform for convergence devices [14].

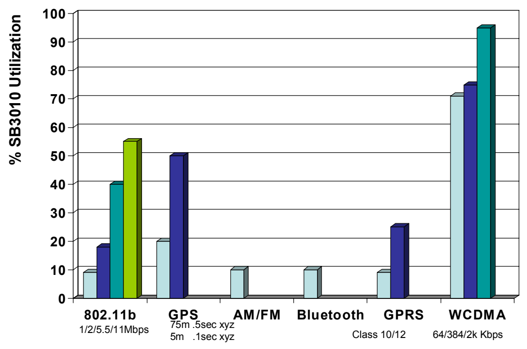

Figure 1. Communications Systems Results as a Percentage of SB3010 utilization (600MHz processor)

Figure 1 shows the results of a number of additional communications systems as a percentage of a 600MHz SB3010 platform. Particularly, 802.11b, GPS, GPRS, AM/FM radio, and Bluetooth, and WCDMA are shown. 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, accuracy, and concurrency can be dynamically adjusted based on the mix of tasks desired.

We have also run multimedia benchmarks such as H.264, MPEG4, and MP3 on the SB3010. The total utilization for CIF images is typically less than 5% of an SB3010 platform. Convergence devices should dynamically adapt to multiple coding schemes for both communications and multimedia standards. The SB3010 platform provides precisely this dynamic adaptation.

Figure 2 shows the SB3010^{TM} baseband chip. It contains four Sandblaster$^{®}$ cores and an ARM microcontroller that functions as an applications processor. It also contains a number of internal digital peripheral interfaces for moving data in and out of the chip such as AD/DA for Tx and Rx data, TDM ports, and an AMBA bus. A high-speed Universal Serial Bus (USB) provides easy connectivity to external systems. Control and test busses such as JTAG, SPI, and I^{2}C allow the chip to control RF and front end chips.

Silicon for the chip is available and fully functional. It runs at up to 800MHz per core providing more than 12 GMACs per chip. Measured results for a synthesized version of this chip have achieved 600MHz operation at 0.9V. The typical power dissipation is 150mW providing the most power efficient processor design in its class.

In addition to processor chips, Sandbridge also provides reference designs including digital and RF boards. Figure 3 shows the SB3010 digital board. The development platform comes enabled with 8MB of external SRAM, 32MB of external Flash, and 256MB of external SDRAM. Other peripherals include USB host and client interfaces with "On-the-Go" capabilities, AC-97 with Microphone and S/PDIF, IrDA, UARTs, LCD, PS2, Keypad, SD Card, MMC card, and Ethernet.

The entire SB3010 Digital Card is powered from a single 120-240V wall adaptor and attaches to any PC or laptop through a single USB connection. The software that runs on the card is unmodified from the software that executes in the simulation environment. It is identical from a programmer's perspective. The Sandbridge runtime takes care of all of the administrative chores associated with loading and executing programs. The card can also be used stand-alone. A full Linux suite is available on the onboard ARM with built-in LCD controller.

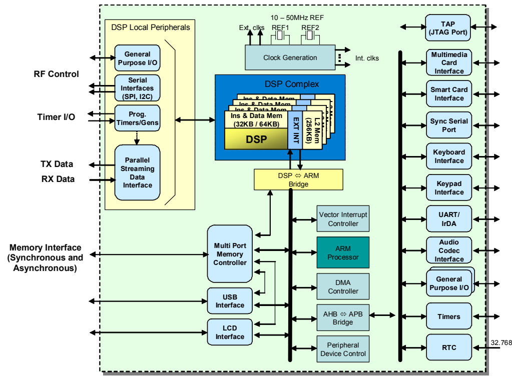

Figure 2. SDR SB3010 Baseband Processor

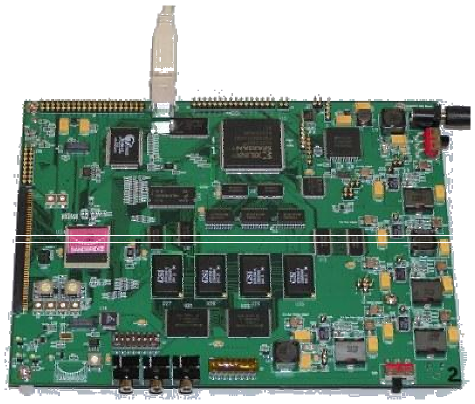

Figure 3. SB3010 Digital Card

5. Summary

As wireless platforms converge to multimedia systems, architectures must converge to support voice, data, and video applications. These convergent devices must be able to roam seamlessly across multiple communications systems. To avoid excessive hardware costs, a Software Defined Radio (SDR) approach offers a programmable and dynamically reconfigurable method of reusing hardware to implement physical layer processing. To achieve software implementations of communications systems a number of fundamental problems must be addressed. Streaming multimedia systems are inherently DSP systems. Power efficiency of DSP systems has been a concern when contrasted with hardware implementations. However, the extensive choices of communications systems have made the complexity of hardware designs intractable. This makes SDR solutions attractive but previous software programming environments have been highly labor intensive, traditionally resulting in assembly language coding which may also be intractable when multiple communications systems are considered.

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 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.

To validate our approach, we implemented a number of communication physical layers and multimedia systems including H.264, MPEG4, MP3, WCDMA, 802.11b, GSM/GPRS, and GPS.

In addition to the software design, we also built RF cards for each communications system. With a complete system, we execute RF to IF to baseband and reverse uplink processing in our lab. Silicon is available for the SB3010 platform and our measurements confirm that our communications and multimedia designs execute within field conformance requirements in real time completely in software.

6. REFERENCES

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

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

[3] B. Case, “Philips Hopes to Displace DSPs with VLIW”, *Microprocessor Report*, December, 1997, pp. 12-15.

[4] O. Wolf and J. Bier, “StarCore Launches First Architecture”, *Microprocessor Report*, Vol. 12, No. 14, October, 1998, pp. 1-4.

[5] J. Fridman and Z. Greenfield, “The TigerSHARC DSP Architecture”, *IEEE Micro*, Vol. 20, January, 2000, pp. 66-76.

[6] J. Turley and H. Hakkarainen, “TI’s New ‘C6x DSP Screams at 1,600 MIPS”, *Microprocessor Report*, Vol. 11, No. 2, February, 1997, pp. 1-4.

[7] K.W. Leary and W. Waddington, “DSP/C: A Standard High Level Language for DSP and Numeric Processing”, *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, IEEE, 1990, pp. 1065-1068.

[8] D. Batten, S. Jinturkar, J. Glossner, M. Schulte, and P. D’Arcy, “A New Approach to DSP Intrinsic Functions”, *Proceedings of the Hawaii International Conference on System Sciences*, Hawaii, January, 2000, pp. 2892-2901.

[9] D. Chen, W. Zhao, and H. Ru, “Design and Implementation Issues of Intrinsic Functions for Embedded DSP Processors”, in *Proceedings of the ACM SGIPLAN International Conference on Signal Processing Applications and Technology (ICSPAT ‘97)*, September, 1997, pp. 505-509.

[10] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley Publishing Company, CA, 1986.

[11] M. Lam, “Software Pipelining: An Effective Scheduling technique for VLIW Machines”, In *Proceedings of the SIGPLAN ‘88 Conference on Programming Language Design and Implementation*, Atlanta, GA, June, 1988.

[12] S. Jinturkar, J. Thilo, J. Glossner, P. D’Arcy, and S. Vassiliadis, “Profile Directed Compilation in DSP Applications”, *Proceedings of the International Conference on Signal Processing Applications and Technology* (ICSPAT’98), September, 1998.

[13] W. Hwu, “Super Block: An Effective Technique for VLIW and Superscalar Compilation”, *Journal of Supercomputing*, Vol. 7, pp. 229-248.

[14] J. Glossner, S. Dorward, S. Jinturkar, M. Moudgill, E. Hokenek, M. Schulte, and S. Vassiliadis, “Sandbridge Software Tools”, *Proceedings of the 3rd annual Systems, Architectures, Modeling, and Simulation (SAMOS) Conference*, July, 2003, pp. 142-148.

[15] V. Kotlyar and M. Moudgill, “Detecting Overflow Detection”, *Proceedings of the 2004 CODES+ISSS International Conference on Hardware/Software Codesign and System Synthesis*, September 8-10, 2004, Stockholm, Sweden, pp. 36-41.

[16] T. Boudreau, J. Glick, S. Greene, J. Woehr, and V. Spurlin, *NetBeans: The Definitive Guide*, O'Reilly & Associates, Sebastopol, CA, 1st edition, October, 2002.

[17] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, “A Software Defined Communications Baseband Design”, *IEEE Communications Magazine*, Vol. 41, No. 1, January, 2003, pp. 120-128.

[18] 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, November, 2002, pp. 70-84.

[19] J. Sebot and N. Drach, “SIMD Extensions: Reducing Power Consumption on a Superscalar Processor for Multimedia applications,” presented at *Cool Chips IV*, April 2001.