THE SANDBLASTER SBX 2.0 ARCHITECTURE

John Glossner[1], Mayan Moudgill[1], Daniel Iancu[1], Sanjay Jinturkar[1], Gary Nacer[1], and Michael Schulte[1,2]

[1]Sandbridge Technologies, Inc.
White Plains, NY 10601 USA
jglossner@sandbridgetech.com

[2]University of Wisconsin, Dept. of ECE
1415 Engineering Drive
Madison, WI, 53706, USA

ABSTRACT

Sandbridge Technologies has developed a new architecture that supports wireless data rates necessary for 3.5G and 4G systems. Building upon the Sandblaster 1.0 architecture, the fully object code compatible Sandblaster SBX 2.0 architecture extends support for high bit-rate processing, MIMO-OFDM acceleration, wider vector execution, and code compression. Architectural performance improvements range from 4x to more than 10x for a variety of signal processing applications while providing 100% object code compatibility with the Sandblaster 1.0 architecture. In this paper we describe the base Sandblaster 1.0 architecture and introduce the Sandblaster 2.0 enhancements.

1. INTRODUCTION

The *architecture* of a computer system is the minimal set of properties that determine what programs will run and what results they will produce [1]. 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 for 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 in the architecture 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.

In 2002, Sandbridge Technologies first described at the SDR Forum Technical Conference a multithreaded architecture for SDR applications [2]. The core architecture, called Sandblaster, supports deterministic real-time execution, vector DSP operations, RISC-style control code, and Java execution. The compound instruction set architecture was optimized for communications and multimedia applications. It includes a complete tool chain which removes the need for tedious DSP assembly language programming [3]. A well known problem for DSP compilers is saturating arithmetic [4]. So called fixed point (fractional) datatypes are non-associative and require special treatment within a compiler. The Sandblaster processor overcomes these limitations by providing both vector architectural execution and compiler algorithms that can determine the type of the variable and thus maintain serial semantics even under parallel execution [5]. The compiler is also able to automatically generate threads for the processor [6]. It ensures all synchronization and works in concert with automatic vector generation for efficient code generation. Furthermore, ultra-fast simulation, profiling, and debugging of code that is embodied in the Sandblaster development environment is a key enabler of fast application development [5].

In the original 2002 publication, we described the SB9600 baseband processor chip implementation. It contained four Sandblaster cores, an ARM processor, and an integrated set of peripherals. Handset operating requirements have significant restrictions on power dissipation. Techniques for achieving handset power levels are described in [7]. The SB9600, renamed the SB3011, was successfully fabricated in TSMC 90nm process technology and is shown in Figure 1.

The SB3011 contains four Sandblaster cores each running at 600MHz, 1.5Mbytes of onboard SRAM memory for L1 and L2 storage, multiple RF peripherals for MIMO operation, an ARM9 running at up to 300MHz, and a complete set of peripherals for smart phone integration. Measured power dissipation of the core (processor) for some important applications ranges from 45mW for GPRS to 65mW (per core) for WCDMA. Having originally published a 75mW per core target, measured results have validated the design. Table 1 summarizes the key Sandblaster parameters.

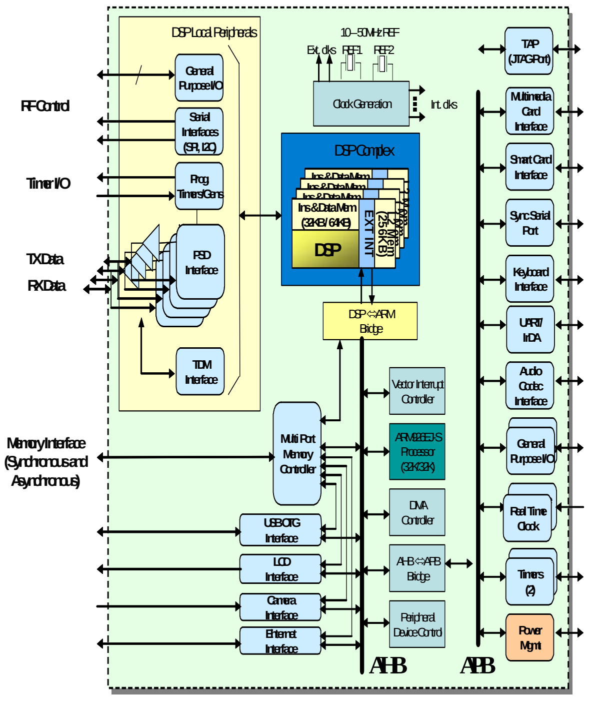

Figure 1. Sandblaster SB3011 Chip

Technology	90nm
Processor Clock	600MHz
Power Dissipation	75mW @ 1V, 25C
On-chip Memory	1.5Mbytes
Peak DSP performance	2.4 GMACs

Table 1. Key Sandblaster Parameters (per core)

With a tool chain capable of automatically generating parallel DSP code and a high-performance low-power chip fully functional, a number of communications systems have been implemented including WCDMA [8], GSM/GPRS [9], 1xEVDO [10], TD-SCDMA [11] , NTSC Video Decode [12], WiMax [13], WiFi [14], GPS [15], AM/FM radio [16], DVB [17], and SINCGARS [18].

In addition to communications systems, the processor is also capable of multimedia. A number of applications have been developed including MP3 [19], MPEG4 [20], and H.264 [21].

1.1 3.5G and 4G Systems Requirements

Table 2 shows the growing performance requirements for future cellular devices. Not only are the bit-rate requirements exploding but the video resolution is also increasing.

In communications systems, the increased performance places additional processing burden in all areas of baseband design. Particularly stressed are error correction codes such as Viterbi, turbo, and LDPC codes. However, in addition, most of the future high bit-rate systems will be MIMO-OFDM systems. Therefore fast FFTs and matrix combining are a requirement.

	3G	3.5G	4G
Cellular	384kbps	14.4Mbps	100Mbps
WLAN	11Mbps	54Mbps	108Mbps
Video	QVGA	WVGA	720p

Table 2. Cellular Terminal Technology Requirements

Due to the demanding requirements of next generation communications systems, a number of instruction set extensions for the Sandblaster processor have been considered [22].

The rest of this paper is organized as follows. Section 2 provides an architectural overview of both the Sandblaster 1.0 and 2.0 architectures. Section 3 provides architectural performance results, and Section 4 provides some concluding comments.

2. THE SANDBLASTER 2.0 SBX ARCHITECTURE

The instruction set for the original Sandblaster 1.0 architecture is very simple. In total there are about 70 instructions. In describing the architecture, we follow the format described in [1]. We start with the original Sandblaster 1.0 (SB1) architecture and describe the modifications incorporated into the Sandblaster 2.0 (SB2) architecture.

2.1 Backward Compatibility

Of primary importance is backward compatibility. All of the instructions supported in the Sandblaster 1.0 architecture are object code executable in the Sandblaster 2.0 architecture. This is a key criterion for what distinguishes a true architecture from an instruction set defined by its implementation.

2.2 Spaces

Instructions and their operands must be obtained from a storage space or an input source; the results are placed in a storage space or an output sink. The Sandblaster architecture supports memory, working store, and control store spaces.

The *memory space* is the storage space from which programs are directly executed. There is no embedding of other spaces within the memory space.

The *working store* is the set of concisely specifiable locations that temporarily contain operands or results of an operation. The working store is broken into three spaces:

general purpose registers (r), vector registers (vr), and accumulator registers (acc).

The general purpose register file contains 16 entries of 32 bits each. A 16-bit datatype occupies bits 0 to 15 and a 32-bit datatype occupies bits 0 to 31. The vector register file contains eight entries. In SB1, each register file entry is 160 bits, treated as four 16-bit or four 40-bit elements. Notably, the SB3011 implementation contains 4 cores each of which has 8 copies of the register state (e.g. 8 threads) for a total of 32 threads and 32 complete copies of the register state. In SB2, each entry is 256 bits and is treated as eight 32-bit vector elements or sixteen 16-bit vector elements. For backward compatibility, four 16-bit and four 40 bit-elements are mapped into each 256-bit entry.

A vector encoding was chosen to preserve the number of names required. It should be noted that a scalar processor may require up to 16 unique names per each SB2 vector. Thus, even with just eight 16-element locations, a scalar processor could require 128 unique names.

The accumulator register file contains four entries. Each entry is 64 bits.

The *control store* is the storage that contains the status of the Sandblaster architecture. It contains the processor status, a 32-bit Instruction Address Register (commonly referred to as the PC), and other control state.

2.3 Memory (Storage) Access

In the Sandblaster architecture, the address space is a one-dimensional vector of addresses. An address is a storage element's unique name. The name spaces of a language are the disjoint sets into which the names of the objects are grouped. In the Sandblaster architecture a set of successive integers as addresses is assigned as the name-space of specific objects. This provides an isomorphic mapping between the set of all possible n-bit names and the set of binary integers from 0 to 2^n-1. This constitutes a dense, ordered and measured set. Thus, the successor of a name can be calculated by addition. This allows the same mechanisms used for operations on data to be used for comparisons and additions desirable for names.

The address-set structure is linear with detection of addresses beyond the ends of the installed segment. This ensures that an increase in memory will not affect correct execution of programs. The minimal memory address resolution is an 8-bit byte. The byte ordering convention is big-endian. Bits, bytes, half-words, etc. are numbered from left to right. There is no requirement for data to be aligned with the datatype size (although it is advisable whenever possible).

The architecture permits generalized use of a three-address operand format. Because working store indices are costly in bits, a working store element is used as the source and destination of an operation. The *base address* specifies the location of an array in memory. The Sandblaster architecture does not provide for a separate base address. The *element address* (sometimes referred to as an offset) specifies an element within a data structure, relative to the base address. It is placed in one of the general purpose registers. The *displacement* determines the location of an item relative to the current element address. It is placed within the instruction format. The address phrase in the architecture is offset + displacement. This effectively requires a precombined base and element address. The displacement is typically limited to a sign extended 4-bit or 16-bit immediate field shifted to be aligned to the vector size. If a non-vector length displacement is required it must be computed. Index arithmetic for memory addressing takes place in the general purpose register file. All integer operations available for normal computations are available for index arithmetic.

Type	Interpretation	Vector Form
u8	1-byte unsigned integer	n/a
w8	1-byte signed integer	n/a
u16	2-byte unsigned integer	u16[4], u16[16]
w16	2-byte unsigned integer	w16[4], w16[16]
fx16	2-byte fraction	fx16[4], fx16[16]
u32	4-byte unsigned integer	u32[8], complex
w32	4-byte unsigned integer	w32[8], complex
fx40	5-byte fraction	fx40[4]

Table 3. Sandblaster 2.0 Datatypes

Addresses that refer directly to the machine-language names for data are called *direct addresses*. There are no indirect addresses (i.e. the use of a memory location that holds the direct address). The architecture provides the following direct address modes:

- Address = Register Offset + Immediate Displacement
- Address = Register Offset + Immediate Displacement;
  Offset = gpr[ra] + Signextend(imm) << 3
- Address = Register Offset + Immediate Displacement;
  Offset = gpr[ra] + Signextend(imm) << 5

2.4 Operations

An *operation code* (opcode) is the encoded specification of the operation. A secondary operation is one implied by an explicitly specified operation. An example is a sign test that explicitly sets a condition code bit after an arithmetic operation. In the Sandblaster architecture there are a few secondary operations such as jump conditional. There are also compound operations such as compare and branch that are encoded in the same operation code.

The architecture is comprised of a collection of fixed-length datatypes. Table 3 summarizes the datatypes directly supported by the SB2 architecture. The data length is specified in an instruction by the operation code and the working store that is used. Operations are provided for integer, logical, and fixed-point datatypes. It should also be noted that floating point is supported from a C-level language viewpoint but it is emulated with native data types. Each instruction specifies the type of its operands.

The term “fixed point number” has different meanings depending upon the domain of discussion. Computer architects use the term generally to include any number that has the radix point in a fixed position. This would include all integer types (unsigned and 2’s complement integers) where the position of the radix point is to the right of the digits, but excludes floating-point numbers.

From the computer architect’s viewpoint, the position of the radix point does not affect the terminology, as long as it is in a fixed position. However, a Digital Signal Processing (DSP) use of the term implies a specific choice of radix point with the position between the most significant bit and the next most significant bit. DSP fixed-point numbers are typically 2’s complement encoded with an interpreted range of [-1 to 1). This alone does not create a conflict in terminology. However, in common DSP usage, operations on this type are implied to “saturate” to the largest or smallest representable values. This complicates mathematics because the arithmetic is non-associative. In this paper we distinguish this by referring to DSP-style fixed point as a *fractional type* (i.e. fx16, fx40).

Type	Example Instructions	Notes
Data Handling	load vector & update load vector reversed	
Arithmetic	add, sub, mpy, mac	vector, mixed types, sat opt.
Logical	and, or, xor sompare, Shift	no rotate sat opt.
Synchronizatio n	load locked store conditional	
Transfer of Control	jump (conditional) call branch eq || gt || lt Loop	
Power	idle	
Special SB1	shuffle, select round, min/max count leading sign/zeros count trailing zeros thread_id	
Special SB2	FFT, Complex, FEC broadcast pack/unpack GF operations	

Table 4. Operations

Table 4 lists the classes of operations with some representative examples. Data handling operations perform no arithmetic and includes register-to-register moves, loads from memory, etc. They provide a mechanism to move data around within the machine. Moves within the vector register file may be conditionally executed. Format transformation is accomplished by sign extending a byte or half word value. A round operation is also provided for precision reducing operations.

Arithmetic is provided for all datatypes. Operations are available for all scalar and vector types. Some mixed mode support such as multiply signed-unsigned is provided. Optionally results from certain operations may be saturated for proper “DSP” style execution. Particularly, saturation of dot-product-type operations in 4-element vector form produces results guaranteed to be equivalent to serial execution of the operations with saturation after each operation. Thus, with a saturating dot product, four 16-bit elements are multiplied, saturated, added, and then saturated again as if they were scalars. Notably, the 16-element saturating vector dot-product-type operations in Sandblaster 2.0 do not follow this convention but instead maintain maximum precision at each intermediate stage in the computation. Note that the 4-element saturating form is still available in the SB2 architecture.

Logical operations are provided for And, Or, and Exclusive-or functions. Vector versions of these are also available including a vector nand. Additionally, vector compare operations set a mask register which can be used to select between elements of a vector. Shift operations are provided both in logical and saturating form.

Synchronization is performed with load locked and store conditional operations. From these basic primitives many conventional software synchronizations may be constructed including semaphores.

Transfer of control is typically accomplished by a jump operation which may be dependent upon a condition or a compare and branch compound operation which first performs the comparison and then determines if a branch is to be taken. A call instruction is provided with automatic saving of the instruction address register. For many DSP (or streaming) applications it is desirable to loop a number of times on the same set of operations. If scalar architectures are used the number of scalar names in conjunction with typically visible pipelines precludes the usefulness of looping type operations. A vector architecture with transparent pipelines allows complete reuse of names.

Figure 2 shows an example of a name saving instruction sequence. Here a single compound instruction with three compound operations for SB1 is shown. 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 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.

```
L0: lvu %vr0, %r3, 8
||  vmulreds %ac0,%vr0,%vr0,%ac0
||  loop %lc0,L0
```

Figure 2. SB1 Sum of Squares Inner Loop

The equivalent code in SB2 that performs 16 multiplies per compound instruction is shown in Figure 3. The first compound operation, lru, loads the vector register vr0 with sixteen 16-bit elements and updates the address pointer r3 to the next element. The rmulreds operation reads four fixed point (fractional) 16-bit elements from vr0, multiplies each element by itself, adds all sixteen products plus an accumulator register, ac0, saturates the result and stores it back into ac0.

```
L0: lru %vr0, %r3, 32
||  rmulreds %ac0,%vr0,%vr0,%ac0
||  loop %lc0,L0
```

Figure 3. SB2 Sum of Squares Inner Loop

In addition to traditional operations, a number of specialized operations are provided. Some are application specific while others are the result of microarchitectural features. Support for multithreading is provided by allowing an instruction stream to determine what thread it is executing in by accessing a thread id register. This allows fast interruption between independent instruction streams. Additionally, an operating system may disable execution of specific hardware threads by executing an idle instruction. This turns off (if an implementation supports it) all execution within a hardware thread. Significant power savings can be achieved using this mechanism.

As mentioned in the Section 1, certain operations are difficult to execute in software when the data rates are very high. The SB2 architecture provides application specific instructions for FFTs, Galois Field arithmetic, and error correction such as Viterbi, turbo, and LDPC codes. Also, since the vector path is 16-elements wide there is now direct support for broadcasting of a scalar to all vector computations and support for a much richer class of permutations and shuffles. Within the 16-element vector unit, a native 32-bit vector element and complex arithmetic are also supported.

2.5 Instruction Execution

The SB1 architecture increments the instruction address by eight bytes each cycle. Three operation codes of 21-bits each are grouped together in the instruction format and issue as a single 64-bit compound instruction. The address where the next instruction resides (called its location) is always an 8-byte linear sequence arranged in a vector.
In the SB2 architecture the leftmost bit of the instruction format now specifies whether the three operation codes are issued in serial fashion or parallel fashion (as in SB1). However, all branch targets must still be 8-byte aligned.

3. RESULTS

The architectural changes have added about 140 additional instructions to the base Sandblaster 1.0 architecture. The impact is significantly reduced instruction counts on communications and multimedia codes. Basic vector performance is enhanced by a factor of four through parallel operations on sets of 16 elements. An improved shuffle network may improve performance further. The execution time of error correction and other application specific codes may be reduced by much more than an order of magnitude.

While we have not discussed microarchitectural performance, it is anticipated that chip implementations built using the Sandblaster 2.0 architecture will execute at about the same clock frequency as the SB3011.

4. CONCLUSION

We have introduced the Sandblaster 2.0 architecture which builds upon the Sandblaster 1.0 architecture. The new architecture is object code compatible with the original architecture and provides new operations to enable software execution of future high speed wireless communications systems such as 802.16e, HSPA, and LTE.

REFERENCES

[1] G. Blaauw and F. Brooks, Computer Architecture: Concepts and Evolution, Addison-Wesley publishers, Reading, Massachusetts, 1997.

[2] J. Glossner, E. Hokenek, and M. Moudgill, “Multithreaded Processor for Software Defined Radio”, *Proceedings of the 2002 Software Defined Radio Technical Conference*, Volume I, pp. 195-199, November 11-12, 2002, San Diego, California.

[3] S. Jinturkar, J. Glossner, E. Hokenek, and M. Moudgill, “Programming the Sandbridge Multithreaded Processor”, *Proceedings of the 2003 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC)*, March 31-April 3, 2003, Dallas, Texas.

[4] P. Balzola, M. Schulte, J. Ruan, J. Glossner and E. Hokenek, “Design Alternatives for Parallel Saturating Multioperand Adders,” in *Proceedings of the International Conference on Computer Design (ICCD 2001)*, Austin , TX, IEEE Computer Society Press, pp. 172-177, September, 2001.

[5] 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*, pp. 142-148, Samos, Greece, July 21-24, 2003.

[6] S. Jinturkar, J. Glossner, V. Kotlyar, and M. Moudgill, "The Sandblaster Automatic Multithreaded Vectorizing Compiler", *Proceedings of the 2004 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC)*, Santa Clara, California, September 27-30, 2004.

[7] J. Glossner, K. Chirca, M. Schulte, H. Wang, N. Nasimzada, D. Har, S. Wang, A. Hoane, G. Nacer, M. Moudgill1, and S. Vassiliadis, "Sandblaster Low Power DSP", in *Proceedings of the Custom Integrated Circuits Conference (CICC)*, Orlando, Florida, October 2004.

[8] J. Glossner, D. Iancu, E. Hokenek, and M. Moudgill, "A Reconfigurable Baseband for 2.5/3G and Beyond", *Proceedings of the 2003 World Wireless Congress*, pp. MC.11-1-6, May 27-30, 2003, San Francisco, California.

[9] R. Kalavai, M. Senthilvelan, S. Agrawal, S. Jinturkar, and J. Glossner, " Implementation of GSM/GPRS Physical Layer on Sandblaster DSP", *Proceedings of Software Defined Radio Technical Forum (SDR Forum '06)*, Orlando, Florida, November, 2006.

[10] S. Watanabe, Y. Kunisawa, D. Kamisaka, A Software Radio Implementation of CDMA2000 1xEV DO on a Single DSP Chip Designed for Mobile Hand Terminal, Proceedings of the IEEE Vehicular Technology Conference, 25 – 28 September 2006, Montréal, Canada.

[11] S. Shamsunder and J. Glossner, "Reduced Complexity Software Receivers for TD-SCDMA Downlink", CD proceedings at the 2004 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), Santa Clara, California, September 27-30, 2004.

[12] V. Kotlyar, D. Iancu, J. Glossner, Y. He, and A. Iancu, "Real-time Software Implementation of NTSC Analog TV on Sandblaster SDR Platform," *Proceedings of the 4th Karlsruhe Workshop on Software Radios*, Karlsruhe, Germany, March 22-23, 2006, pp. 171-176.

[13] D. Iancu, H. Ye, E. Surducan, M. Senthilvelan, J. Glossner, V. Surducan, V. Kotlyar, A. Iancu, G. Nacer, and J. Takala, "Software Implementation of WiMAX on the Sandbridge SandBlaster Platform," *Proceedings of the 6th Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS'06)*, Samos, Greece, July, 2006.

[14] V. Ramadurai, S. Jinturkar, S. Agarwal, M. Moudgill, and J. Glossner, "Software Implementation of 802.11a blocks on Sandblaster DSP", *Proceedings of Software Defined Radio Technical Forum (SDR Forum '06)*, Orlando, Florida, November, 2006.

[15] D. Iancu, J. Glossner, H. Ye, M. Moudgill, and V. Kotlyar, "Rake Receiver Enhanced GPS System", *Proceedings of Software Defined Radio Technical Forum*, Volume A, pp. 97-105, 16-18 November, 2004, Scottsdale, Arizona.

[16] D. Iancu, J. Glossner, H. Ye, Y. Abdelilah, and S. Stanley, "Reduced Complexity Software AM Radio", *Proceedings of the Symposium Trends in Communications (SympoTIC '03)*, pp. 122-125, Bratislava, SLOVAKIA, 26 – 28 October 2003.

[17] D. Iancu, H. Ye, Y. Abdelilah, E. Surducan, and John Glossner, "On the Performance of Multiple OFDM Receivers for DVB" *Proceedings of the Joint IST Workshop on Mobile Future & Symposium on Trends in Communications (SympoTIC'04)*, Bratislava, Slovakia, pp. 1-4, October 24-26, 2004.

[18] B. Beheshti, J. Glossner, D. Routenberg, L. Zannella, and P. Steensma, "Evaluation of Military Waveform Processing on a COTS Reconfigurable SDR Processing Platform", *Proceedings of Software Defined Radio Technical Forum*, Volume A, pp. 147-151, 16-18 November, 2004, Scottsdale, Arizona.

[19] S. Jinturkar, V. Ramadurai, V. Kalashnikov, G. Nacer, and J. Glossner, "Implementing MP3 Decoder on Sandblaster DSP", *Proceedings of the 2006 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC)*, Santa Clara, California, November, 2006.

[20] S. Agrawal, S. Jinturkar, V. Ramadurai, M. Moudgill, and J. Glossner, "Multithreading MPEG4 Encoder on Sandblaster DSP", CD *proceedings at the 2005 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC)*, Santa Clara, California, October 24-27, 2005.

[21] V. Ramadurai, S. Jinturkar, M. Moudgill, and J. Glossner, "Multithreading H.264 Decoder on Sandblaster DSP", CD *proceedings at the 2005 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC)*, Santa Clara, California, October 24-27, 2005.

[22] S. Mamidi, E. R. Blem, M. J. Schulte, J. Glossner, D. Iancu, A. Iancu, M. Moudgill, and S. Jinturkar, "Instruction Set Extensions for Software Defined Radio," *accepted for publication in the Journal on Embedded Systems*, 2007.