THE SANDBLASTER 2.0 ARCHITECTURE AND SB3500 IMPLEMENTATION

Mayan Moudgill, John Glossner, Sitij Agrawal, and Gary Nacer

Sandbridge Technologies, Inc.
Tarrytown, NY 10601 USA
glossner@sandbridgetech.com

ABSTRACT

The Sandblaster architecture is a high-performance vector architecture targeted at digital signal processing applications. The Sandblaster 1.0 architecture was targeted at implementing the physical layer of 3G wireless standards, with peak data rates of up to 15 Mbps. In this paper, we describe an object code compatible version 2.0 of the Sandblaster architecture, which is targeted at the 4G standards, which have support higher data-rates and more complex algorithms.

To achieve the necessary performance to implement 4G standards, the 2.0 version of the architecture extends the 4-MAC Sandblaster 1.0 architecture to a 16-MAC architecture, with accompanying changes in the width of the vector register file. It also introduces new vector operations that are specialized to key algorithms specified in the 4G standards.

The architectural enhancements have been implemented in the SB3500 chip fabricated in low power 65nm process technology. The chip is fully functional, provides nearly 30 GMACs of DSP performance at 600MHz and validates the design objectives of the 4G standards.

1. INTRODUCTION

Experience with programming various 3G standards on the SB3010 and SB3011 chip implementations [1] of the original Sandblaster 1.0 architecture [2] identified certain areas for architectural improvement to support higher datarates [3]. Also, the next set of wireless standards (so-called 4G), such as WiMax and LTE, would require more processing power than could be provided by a low-power implementation of the 1.0 architecture. The Sandblaster 2.0 architecture was developed so as to allow the software implementation of the physical layer of the various 3.5G and 4G standards.

The most significant change to the architecture is the introduction of 16-wide vector operations, in contrast to the 4-wide vector operations of the original 1.0 architecture. Also, the 2.0 architecture contains instructions that are specialized for the efficient execution of key 3.5G and 4G kernels.

Section 2 of this paper gives an overview of the architecture. Section 3 focuses on the 16-wide vector operations. Section 4 discusses additional enhancements. The SB3500 chip is presented in Section 5. The performance achievable from these changes is discussed in Section 6. We present related work in Section 7 and provide concluding remarks in Section 8.

2. ARCHITECTURE OVERVIEW

The Sandblaster architecture uses a 64-bit instruction word, which consist of a serial S-bit and up to 3 21-bit compound operations. If the S-bit is not set (e.g. 0), then all 3 operations are executed in parallel, otherwise they are executed serially. If an operation in a serial instruction is a taken branch, then operations subsequent to the branch will not be executed. All branches are to 8 byte boundaries. In particular, it is not possible to branch into the middle of an instruction word, even if that instruction is a serial instruction.

There are 4 categories of operations: branch, integer, memory and vector. A parallel instruction can contain at most one operation from each category; a serial instruction has no such restriction.

The Sandblaster architecture specifies several heterogeneous register files. In general, a register file is accessed by only one category of instructions. A list of the more commonly used registers and their properties can be found in Table 1. Only the gener

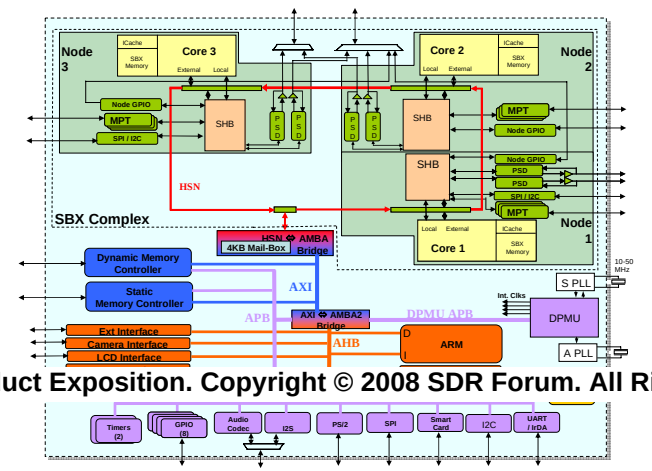

Figure 1 SB3500 Chip

al purpose and vector registers can be loaded from or stored to memory. The other registers must be moved to/from a general purpose register using the copy from/to special purpose register (cfsr/ctsr) operations. The cfsr/ctsr instructions are also used for accessing system specific registers.

The Sandblaster architecture is a load/store architecture; i.e. only load or store operations access memory. These operations are register+immediate addressing, where the base is provided by a General Purpose Register (GPR).

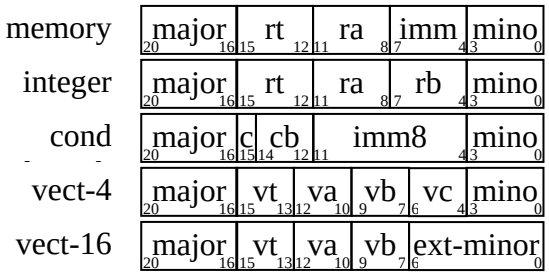

Figure 2 Operation formats

Only the GPR and vector registers can be load/store targets/sources. Note that this implies that the memory data-path is 256 bits wide. Also, the architecture provides a 32-bit address space.

Loads to the vector register file can either load in forward order, so that when loading an array of shorts, the value at index 0 is loaded into bits 0..15 of the register, or in reverse order, so that value at index 0 is loaded into bits 240..255 of the register. The increment may be computed automatically based on a 4-wide or 16-wide vector operation.

Figure 2 shows some examples of operation encodings for the various operations. The 5-bit major opcode identifies the category the operation belongs to. The minor and extended minor fields further identify the function. Generally, most registers accessed by an operation are explicitly encoded, and operations are non-destructive (i.e. a register field does not encode both a source and a target register). However, this is not universally true. For example:

- add-with-carry implicitly uses cb0 for the carry bit, both reading and writing it.
- load-and-update reads ra for the address base computes base+imm and writes that value to ra, as well as using the value as the address to load into rt.

An operation with an immediate value can be encoded in an instruction word with an immediate-extender operation next to it. In that case, the immediate value is extended by 12 bits from the second operation.

3. VECTOR UNIT

The biggest change between the Sandblaster 1.0 and 2.0 architectures is the vector operations and registers.

Name	Number	Bits	Category	Notes
General Purpose	16	32	Integer, Memory	can be loaded/stored
Condition	8	1	Branch	cb0 also set/used by integer
Jump Target	2	32	Branch	
Loop Count	2	32	Branch	
Vector	8	256	Vector	can be loaded/stored
Mask	4	32	Vector	
Accumulator	4	64	Vector	
Search	2	32	Vector	

Table 1: Register files

The vector registers in the 1.0 architecture were 160 bits wide, and were connected to memory by a 64-bit data-path. In the 2.0 architecture, the vector registers are 256-bit wide and connected to memory using a 256 bit data path. Further, the mask and accumulator registers have been expanded from 4 & 40 bits to 32 and 64 bits respectively.

In most cases, a SIMD operation in the 1.0 architecture operated on 4 values in parallel. By contrast, the 2.0 architecture operates on 16 short (16-bit) values or 8 (32-bit) integer values in parallel. Also, the 1.0 operations were fairly general-purpose. In the 2.0 architecture, the general purpose operations are augmented by operations specialized for key kernels of 4G wireless communications systems.

3.1. Element-wise operations

The element-wise operations include common operations such as logical, shift and arithmetic operations that read 2 registers, perform 16 short or 8 integer operations in parallel, and write the results back to a third register. An example would be the element-wise add integer operation, *radd32:*

```
for(i=0; i<8; i++)
  vt[i] = va[i] + vb[i];
```

In this operation, the registers va, vb, and vt are interpreted as storing 8 32-bit values.

Element-wise multiplies are only done on short values; a single operation can specify 16 short multiplies to be done in parallel. In one set of variants, either the upper or lower 16 bits of the 32 bit product are written as the result, as in the *rmul* operation:

```
for( i=0; i <16; i++ )
   vt[i] = (va[i]*vb[i])&0xffff;
```

Alternatively, the full 32 bit product may be written to 2 registers. There are also multiply-and-add or subtract variants.

Complex multiplies treat the register file contents as though they were alternating short real and imaginary values, so that a register contains 8 short complex numbers. A complex multiply uses 4 short multipliers, so implementing 8 complex multiplies in parallel would have required 32 multipliers. Instead, the Sandblaster 2.0 complex multiply operations multiply either the upper or lower halves of registers together, writing the complex product to the upper or lower half of the result register.

There are both element-wise 8-wide integer and 16-wide short compares. The result of the compare is written to the lower 8/16 bits of a mask register. The contents of the mask register can then be used to select between the elements of two registers. Element-wise max and min operations combine the effects of a compare greater/less than and select operations together, as is show in the pseudo code for *rmax:*

```
for( i=0; i<16; i++ )
  mr[i] = va[i]>vb[i];
  vt[i] = mr[i]?va[i]:vb[i];
```

Element-wise masked sum operations sum together different arrangements of pairs of elements from the input registers. These operations use the bits in the mask register to determine whether the corresponding element should be negated or not prior to adding. For example, *rmsum:*

```
for( i=0; i<16; i++ )
  a = mr[i]?-va[i]:va[i];
  b = mr[16+i]?-vb[i]:vb[i];
  vt[i] = a + b;
```

3.2. Reduction operations

Reduction operations take the multiple elements of a vector and combine the values into a scalar result, which is then written to an accumulator register.

The multiply reduction operations do element-wise multiplies, and then sum the products together. For example *rmulred:*

```
for(i=0; i<16; i++)
  act += va[i] * vb[i];
```

where act is the lower 32 bits of the accumulator target register.

Complex multiplies can also be reduced; in this case, the 32 bits of the real part of the sum are stored in the lower 32 bits of the accumulator and the imaginary part of the sum is stored in the upper 32 bits of the accumulator.

Masked sum operations can also be reduced. These come in two variants; one adds up all the sums. The other variant adds up half the sums into the lower 32 bits of the accumulator and the adds up the other half into the upper 32 bits of the accumulator.

Search operations are also a type of reduction operation. The elements of a register are compared against each other and an accumulator to find the maximum (or minimum). These instructions also modify a pair of registers - the position and count registers - so that the position of the maximum/minimum can be determined. This is illustrated by the *rsearchmin* operation:

```
for( i=0; i<16; i++ )
  if( va[i] < act )
    act = va[i];
    pos = count;
  count++
```

To search an array for the minimum value, first the count and pos values are cleared using the ctsr register, and the act register is set to 0x7fff_ffff. After a sequence of rsearchmin operations, act will contain the smallest value seen so far, count will contain the number of array elements examined, and pos will contain the position where the smallest value was encountered.

3.3. Specialized operations

Significantly, the Sandblaster 2.0 architecture introduced groups of vector operations designed to improve the performance of specific algorithms required by the 4G standards.

One group of operations is used to implement fast-fourier transforms (FFTs). These operations do 4 complex multiplies per cycle, producing 8 complex elements of the result. Depending on the operation, the values are either written to one register or to the upper/lower halves of two registers.

Galois field arithmetic support is provided by operations that do polynomial multiply, multiply-reduce and multiply-and-add and compute the polynomial-modulus.

Viterbi decoding adds operations that perform 16 viterbi butterflies in parallel, reading 3 registers (2 for the state and 1 for weight) and writing the resulting state into 2 registers. The trace-back bits are written to the accumulator registers. There are also flavors of the masked sum operations that are used to compute the branch metrics from the input samples.

Turbo decoding is supported by operations that compute the forward, backward and likelihood values. The turbo-decode operations assume that the constraint length of the convolutional coders is 3. They execute two steps of the forward (or backward) pass per operation. In addition, they may combine the forward (backward) steps with the result of prior backward (forward) steps to compute the likelihood. This likelihood is stored in an accumulator register.

Dibit turbo decode is supported by similar operations. However, dibit turbo-decoding only does one step of the forward/backward pass per operation.

3.4. Other operations

The other vector operations rearrange data in the registers. These include:

- packing/unpacking 8 bit data to 16 bit data
- packing/unpacking 16 bit data to 32 bit data,
- shuffling the elements of a pair of register
- copying the contents of an accumulator to all elements of a register
- rotating register pairs
- shifting accumulator data into a register

3.5. Fixed point operations.

Digital signal processing typically uses fixed-point arithmetic. Consequently, all the vector operations that do addition, subtraction, multiplies, and left-shift have a fixed-point version. Further, operations such as complex-multiplies and the specialized operations come only in the fixed point versions.

Fixed point arithmetic differs from the standard 2s complement arithmetic in several ways:

- a fixed point multiply is further multiplied by 2
- if the result of an arithmetic operation overflows the number of bits available, it is saturated to the maximum/minimum representable value
- when converting from a type with more bits to fewer bits, the upper bits are used.

The following pseudo-code for a fixed-point 16 bit multiply producing a 16 bit result illustrates all these features:

```
long long p; // extra bits to keep track
             // of overflow
p = x*y;
// fixed point multiply
p = p*2;
// saturation
if(p > 0x7fff_ffff )
  p = 0x7fff_ffff;
else if( p < -0x8000_0000 )
  p = -0x8000_0000;
// convert to 16-bit
z = (p>>16)&0xffff
```

When an operation does a sequence of fixed-point arithmetic computations, one can saturate after each intermediate computation. Alternatively, one can keep all intermediate results at full precision (i.e. use enough bits so that there is no possibility of overflow) and then saturate the combined result. The vector operations from the Sandblaster 1.0 architecture saturated after each intermediate operation [4]; the operations introduced in the 2.0 architecture saturate only the final result.

3.6. Rotation/Shifting

Many wireless kernels involve data-streaming, which involve operating on subsequences of data offset from each other by one or two positions. For example, consider a 16-tap FIR filter:

```
for( j=0; j<M; j++ )
  sum = 0;
  for( i=0; i<16; j++)
    sum += x[i+j]*c[i];
  z[i] = sum
```

In this example, the dot-product of c[0..15] with x[0..15] is computed, then the dot-product with x[1..16], x[2..17] and so on. The 2.0 architecture supports this idiom through register pair rotation.

In register pair register rotation, each pair of even/odd registers is treated as a circular shift register. The values in them can be shifted by 1, 2 or 4 shorts. The pseudo-code below illustrates the 1-short (i.e. 16 bit) rotate:

```
val_e0 = ve[0];
val_o15 = vo[15];
for( i=0; i<15; i++ )
  ve[i] = ve[i+1];
  vo[i+1] = vo[i-1];
ve[15] = val_o15;
vo[0] = val_e0;
```

Note that elements in the even and odd registers are shifted in opposite directions by a rotate.

Array x from in the example can be streamed using the register rotation operation, *rrot*, as follows:

- load x[0..15] into an even register
- load x[16..31] into an odd register in reverse order
- after every complete execution of the inner loop, rotate the even/odd pair by 1.
- After 16 iterations of the outer loop, load the next 16 values of the x array into the odd register, in reverse order

Shifting can also be used to save a series of accumulated values. Reduction operations as well as certain specialized operations store their result into accumulator registers. The *rshift* operation can then shift the value from an accumulator into a vector file, as shown in the following pseudo-code:

```
for( i=0; i<15 i++ )
  ve[i] = ve[i+1];
ve[15] = aca;
```

In this example, 16 bits from the accumulator are shifted into an even register. If the target had been an odd vector register, it would have been in the opposite direction, as in the rotate instructions. Different variants of the instruction can shift different 16, 32, and 64 bits from an accumulator into the vector register. The *rshift0* operation

additionally clears (i.e. sets to 0) the shifted accumulator register.

A 16 tap FIR filter would use a *rmulreds*, *rshift0* and *rrot* operation. Of these, 1 operation is to do the actual computation and the other 2 are overhead to rearrange the data. This is a fairly common occurrence. Consequently, the architecture has some operations that combine an operation with accumulator shifting and clear and register-pair rotation. These include the *rmulreds1r* operation.

4. OTHER CHANGES

There have been other extensions made to the Sandblaster 2.0 architecture to allow it to better handle DSP algorithms.

4.1. I-cache

The Sandblaster 2.0 architecture adds operations to control the instruction cache. These are part of the branch category, and use the jump-target registers to specify a instruction address. This address is used by operations to flush a set and to prefetch an instruction into the cache.

The prefetch instructions allow the cache to be warned up, so that the initial cold-miss penalty can be avoided. This improves the worst-case run-time of an algorithm, thereby improving real-time performance.

4.2. Integer unit

The integer unit has added several operations. These are:

- Parity: compute the even parity of a word
- Galois field: compute the polynomial product and modulo. These operations are similar to the operations in the vector unit.
- Reversal: swap the bits or bytes of a word.
- Traceback: help traverse the trace-back array generated by the viterbi vector operations. One operation generates the address of the next word, and the other extracts the appropriate bit.

4.3. DMA

In most systems, a direct memory access (DMA) engine is provided as a memory-mapped peripheral. The Sandblaster 2.0 architecture, the DMA has been made part of the architecture. The DMA control registers are part of the architected state, and accessed via cfsr/ctsr instructions. A process can be swapped even if it has a DMA operation in flight; the DMA is architected so that a DMA operation can be halted in-flight and the control registers copied out. After the process has been resumed, the control registers can be restored, and the DMA restarted from where it was halted.

The DMA also implements scatter and gather functions. Thus, apart from block copies, the DMA can be programmed to implement gathers:

```
for(i=0; i<N; i++)
  dest[i] = src[off[i]];
```

and scatters

```
for(i=0; i<N; i++ )
  dest[off[i]] = src[i];
```

The scatter and gather can occur at a granularity of 1, 2, 4 or 8 bytes.

Various wireless algorithms require permutation of large amounts of data. For example, turbo-decoding requires input data and likelihood to be repeatedly interleaved. The scatter DMA allows for efficient implementation of the algorithm.

5. SB3500 CHIP IMPLEMENTATION

The architecture enhancements have been incorporated into the SB3500 chip implementation. As shown in Figure 1, the chip contains 3 Sandblaster 2.0 cores. Each of the cores typically runs at 600MHz while providing twice the power efficiency of the SB3011 chip design. The peak performance of the chip is nearly 30 GMACs at handset power dissipation levels. The same as with the SB3011 chip implementation, the chip contains a full ARM subsystem with all the peripherals required to operate a smart phone device including USB 2.0, camera, video, smart card, SIM, keyboard, and LCD ports. The chip is enhanced with a split transaction AXI bus to allow HD video processing while performing 4G baseband communications. The chip is fabricated in 65nm technology and is fully functional.

6. RESULTS

On the Sandblaster 1.0 architecture we have implemented multiple real-time communications systems including WCDMA [12], GSM/GPRS [13], 1xEVDO [14], TD-SCDMA [15] , NTSC Video Decode [16], WiMax [17], WiFi [18], GPS [19], AM/FM radio [20], DVB [21], and SINCGARS [22]. Based on the analysis of these systems combined with 4G WiMax and LTE analysis, we have implemented kernels for various wireless standards, and measured the number of instructions used. Some of the results are summarized in Table 2. As can be seen, the combination of specialized operation support and combined compute/rotate/shift operations allow us to achieve close to optimal performance.

Algorithm	Type/Phase	Instructions
FIR	16-tap real	1 /output
	16-tap complex	2 /complex output
FFT	Core	N/6*(logN-1)
	bit-reversal	N/3 for N point FFT
64 state Viterbi	forward pass	2 per bit
	Traceback	2 per bit
Turbo decode	Forward	21/32 per bit
	Backward+ likelihood	22/32 per bit
Dibit turbo decode	Forward	12/8 per dibit
	Backward+ likelihood	13/8 per dibit

Table 2: Performance

While not completely described at the system level, these performance results enable the real-time software execution of high data rate 4G systems.

7. RELATED WORK

In this section we contrast and compare our approach to other processor designs. Other SDR platforms include the Signal Processing on Demand Architecture (SODA) [5], OnDSP [6], the Embedded Vector Processor (EVP) [7], the Synchronous Transfer Architecture (STA) [8], picoArray [10], XiSystem [9], and the MS1 reconfigurable DSP (rDSP) Core [11].

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 processor 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 multi-context reconfigurable gate array, and reconfigurable I/O modules in a SoC platform. The multi-context 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.

8. CONCLUSIONS

We have presented an architectural description of the next generation Sandblaster 2.0 architecture. We have described the major enhancement to the base 1.0 architecture that includes wider vectors and application specific instruction support. We have briefly described the SB3500 chip implementation that incorporates the architectural extensions. The chip validates the performance and power design objectives of the architecture. Based on previously implemented systems along with the kernel analysis we can implement future 4G standards completely in software on implementations of this processor.

9. REFERENCES

[1] J. Glossner and D. Iancu, “The Sandbridge SB3011 SDR Platform” accepted for publication at Symposium on Trends in Communications (SympoTIC’06), Invited keynote, Bratislava, Slovakia, June 24-26, 2006.

[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. 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 on a Multithreaded Processor," Proceedings of the ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems, San Jose, CA, pp. 266-273, September 2005.

[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] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture For Software Radio," Proceedings of the 33rd Intl. Symposium on Computer Architecture, pp. 89-100, June 2006.

[6] J. Kneip, M.Weiss,W. Drescher, V. Aue, J. Strobel, T. Oberthür, M. Bolle, and G.Fettweis, "Single Chip Programmable Baseband ASSP for 5 GHz Wireless LAN Applications," IEICE Transactions on Electronics, pp. 359-367, February 2002.

[7] C. van Berkel, F. Heinle, P.P.E. Meuwissen, K. Moerman, and Matthias Weiss, "Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices," EURASIP Journal on Applied Signal Processing, Vol. 16, pp. 2613–2625, 2005.

[8] 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, January 2005.

[9] A. Duller, G. Panesar, and D. Towner, "Parallel Processing — the picoChip Way!," Communicating Processing Architectures 2003, pages 125–138, 2003.

[10] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis; L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M, Toma, R. Guerrieri, "XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module," IEEE Journal of Solid-State Circuits, Vol. 41, No. 1, pp. 85-96, January 2006.

[11] B. Mohebbi, E. C. Filho, R. Maestre, M. Davies, F. J. Kurdahi, "A Case Study of Mapping a Software-Defined Radio (SDR) Application on a Reconfigurable DSP Core, Proceedings of the International Conference on Codesign and System Synthesis, pp. 103-103, 2003.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.