The Sandbridge SDR Communications Platform

John Glossner[†,*], Mayan Moudgill[*] ,and Daniel Iancu[*]

[*]Sandbridge Technologies, Inc.
1 North Lexington Ave.
White Plains, NY 10601 USA
{jglossner , mayan, diancu}@SandbridgeTech.com

[†]Delft University of Technology
EEMCS
Delft, The Netherlands

Abstract

Multimedia convergence devices require dramatic increases in computing performance in mobile wireless terminals. Integration of computationally intensive hardware blocks has factorially raised design complexity. To enable elegant reuse of hardware, software defined radio solutions have been proposed. Deploying SDR solutions into mobile terminals presents a number of challenges including power dissipation constraints, software productivity, and real-time system response. In this paper we describe the Sandbridge SDR communications platform and our solutions to providing mobile multimedia convergence devices.

1. Introduction

Performance requirements for mobile wireless communication devices have expanded dramatically since 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. Such complex functionality and features require high computing capability at low power consumption; adding new features requires adding computing capability.

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

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 multimedia.

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. In second generation mobile communications, GSM and IS-95 data rates were limited to less than 15 kbits/s. Future third generation (3G) systems and beyond (B3G) may provide data rates more than 1000 times the previous rates. Escalating communication rates are accelerating DSP processing requirements.

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 difficult 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. A software-based platform capable of dynamically reconfiguring communications systems enables elegant reuse of silicon area and dramatically reduces time to market through software modifications instead of time consuming hardware redesigns.

Digital Signal Processors (DSPs) are now capable of executing billions of operations per second at power efficiency levels appropriate for handset deployment. This has brought Software Defined Radio (SDR) to prominence as technology begins to allow the designer to address a new and hitherto difficult portion of the software radio implementation space.

The SDR Forum [1] defines five tiers of software radio:

- *Tier-0:* 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 analogue 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.

In the subsequent sections we describe the Sandbridge approach to delivering these benefits. The derivation of this solution outlines the major challenges of the technology and of the market, particularly the need to future proof designs against continually evolving standards and air interfaces.

2. Processor Architecture

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 defines the architecture.

The requirement for 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 (e.g. 0-overhead loop buffers, visible memory, and exposed pipelines). 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 (i.e. not transparent) 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 more 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 trend 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 [3][4][5][6][7]. 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 DSP processors includes large program executables (code bloat) that results 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 a 4 vector, saturate, accumulate, saturate).

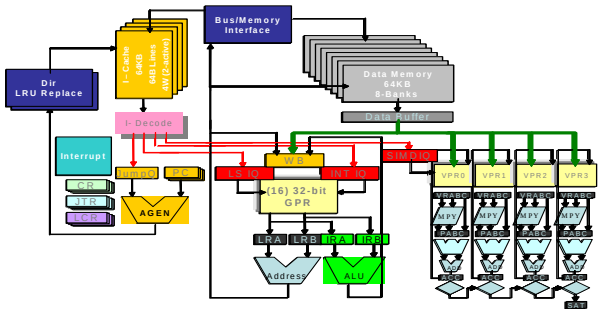

Figure 1. *Sandblaster Microarchitecture*

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 targets of each instruction. This can result in high power dissipation, 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 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.

DSP processing requires support for filtering and highly compute intensive functions to enable software execution of baseband physical layers. DSPs have traditionally implemented one or more multiply accumulate (MAC) units to perform these functions. DSP operations typically operate on fixed point (fractional) saturating datatypes. Because 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.

Embedded control processing can be implemented using a standard Reduced Instruction Set Computer (RISC); this is also amenable to high level language compilation.

In addition to Embedded and DSP processing, Java processing may be required. Future wireless systems will make significant use of Java; indeed, a number of carriers are already providing Java-based services and all 3G terminals could potentially be required to support Java [8].

Figure 1 shows the Sandbridge multi-threaded processor. It is capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [9][10]. This 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 design includes a unique combination of modern techniques such as a SIMD Vector/DSP unit, a parallel reduction unit, a RISC-based integer unit, and instruction set support for Java execution. Instruction space is conserved through the use of compounded operations that are combined in a single instruction for execution. The resulting combination provides for efficient control, DSP, and Java processing execution.

3. Processor Software Tools

Programmer productivity is one of the major concerns in complex DSP applications. Because most classical DSPs are programmed in assembly language, it takes a very large software effort to program an application. For modern speech coders, [21] for example, it may take up to nine months or more before the application performance is known. Then, an intensive period of design verification ensues. If efficient compilers and simulation tools for DSPs were available, significant gains in software productivity could be achieved [39]. In reality, the availability of such tools will be an essential determinant of the rate of adoption new generation DSP architectures.

The ideal DSP design process is achieved by co-designing the DSP compiler alongside the architecture, based on the intended application domain, trade-offs may be made between the architecture and the compiler subject to the application performance, power and price constraints[20]. Even so, there are a number of basic 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 data-type 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. The compiler must then deconstruct these idioms to recognize 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.

Unfortunately, to reduce instruction storage, it was common to use 16-bit encoding for all instructions. Often, three operations could be issued from the same 16-bit instruction. While this was good for code density, orthogonality suffered. Many special purpose registers were required and severe restrictions on operation combinations were imposed.

Early attempts to remove these restrictions used VLIW instruction set architectures with nearly full orthogonality. To issue four multiply accumulates (MACs) minimally requires 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, language extensions have been introduced to high level languages [22][23]. Typical additions may include special type support for 16-bit data types (Q15 formats), saturation types, multiple memory spaces, and SIMD parallel execution support. These additions often imply a special compiler, and the code may not be emulated easily on multiple platforms. As a result, special language constructs have not been successful.

To reduce 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.

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 [24].

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 substitution specifies its replacement. The lower-level instructions resulting from the substitution are called intrinsic instructions [25]. Intrinsics are used to collapse what may be more than ten lines of C code into a single DSP instruction.

Early intrinsic efforts, like inlined asm statements, inhibited DSP compilers from optimizing code sequences [26]. A DSP C compiler could not distinguish the semantics and side effects of the assembly language constructs. 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 were explored in [27]. This technique represented the operation in the intermediate representation of the compiler. With the semantics of each intrinsic well know to the intermediate format, optimizations with the intrinsic functions were easily enabled yielding speedups of more than 6 times.

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

In addition to classic compiler optimizations [28], there are some advanced optimizations which have proven significant for DSP applications. Software pipelining [29] in combination with aggressive inlining has proven effective in extracting the parallelism inherent in DSP applications. Interestingly, some DSP applications are not data dependent. In these cases, profile directed optimizations are very effective at improving performance [30]. These techniques, when used with VLIW scheduling [31], have proven effective in DSP compilation. However, they still can be more than two times less efficient than assembly language programmers.

Provision of effective design tools is essential for any new DSP architectural solution – thus Sandbridge has developed a supercomputer-class vectorizing compiler to accompany its SDR processor. A unique aspect of this compiler is that DSP operations are automatically generated using a technique called semantic analysis.

In semantic analysis, a sophisticated compiler searches for the meaning of a sequence of C language constructs. A programmer writes C code in an architecture independent manner - such as for a microcontroller - focusing primarily on the function to be implemented; if DSP operations are required, the programmer implements them using standard modulo C arithmetic. The compiler analyzes the C code, automatically extracts the DSP operations and generates optimized DSP code without the excess operations required to specify DSP arithmetic in C code. This technique has a significant software productivity gain over intrinsic functions and does not force the compiler writers to become DSP assembly language programmers.

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

A final difficult consideration is vectorizing saturating arithmetic. Because saturating arithmetic is non-associative, the order in which the computations are computed is significant. Because the Sandbridge compiler

was designed in conjunction with the processor, special hardware support allows the compiler to safely vectorize non-associative loops.

Figure 2 shows the results of various compilers on out-of-the-box ETSI C code for the AMR encoder. The y-axis shows the number of MHz required to compute frames of speech in real-time. The AMR C code is completely unmodified and no special include files are used. Without using any techniques such as intrinsics or special typedefs, the Sandbridge compiler is able to achieve real-time operation on the Sandblaster core at hand-coded assembly language performance levels. Note that it is completely compiled from high-level language. Since other solutions are not able to automatically generate DSP operations, intrinsic libraries must be used. With intrinsic libraries the results for most DSPs are near the Sandbridge results; however, they only apply to the ETSI algorithms, whereas the Sandbridge compiler can be applied to arbitrary C code.

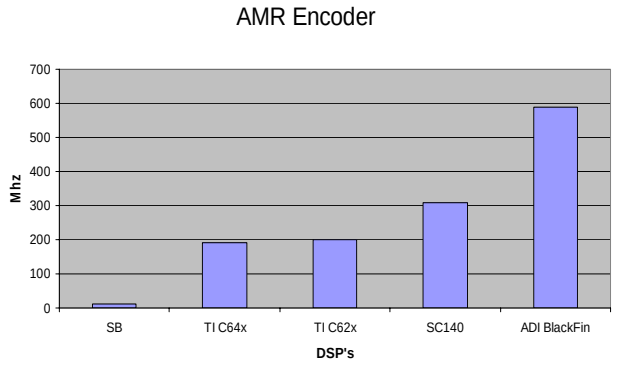

Figure 2. *Out-of-the-box AMR ETSI Encoder C code Results*[i]

4. A 3G System SDR Implementation

Implementing a third generation wireless technology (3G) design is up to ten times more complex than previous 2/2.5G designs [35][36]. Furthermore, modern communications devices will increasingly be required to connect to networks with multiple protocols. A factorial combination of choices presents intractable chip implementations. To facilitate successful system-on-a-chip (SoC) designs requires new approaches to communications systems implementation.

Available communications systems have been developed in hardware due to high computational processing requirements. DSPs in these systems have been limited to speech coding and orchestrating the custom hardware blocks. In high-performance 3G systems there may be over 2 million logic gates required to implement physical layer processing. The design typically contains an ARM processor, a DSP, and many hardware blocks for specific communications processing (e.g. WCDMA, cdma2000, IS-95, GSM, Bluetooth, etc.) – and this is just for the modem. Multimedia terminals typically include an additional ARM, an additional DSP, and additional hardware accelerators for multimedia functions. Choosing the proper combination of accelerators poses both implementation challenges and time-to-market constraints.

A complex 3G system may also take many months to implement. After logic design is complete, any errors in the design may cause up to a 9 month delay in correcting and refabricating the device. This labor intensive process is counter productive to fast handset development cycles. An SDR design can take a completely new approach to communications system design.

The SDR Convergence Processor Approach

Rather than designing custom blocks for every function in the transmission system, a small and power efficient core can be highly optimized and replicated to provide a platform for broadband communications - an SDR processor capable of executing operations appropriate to broadband communications. This approach scales well with semiconductor generations and allows flexibility in configuring the system for future specifications and any field modifications that may be necessary.

The Sandbridge communications design approach begins with Matlab models of both the basestation and the terminal. Within Matlab we implement all conformance testing necessary to ensure that our algorithms perform properly. We then code in high-level ANSI C and use our sophisticated tool chain to immediately produce production executable code.

The UMTS WCDMA FDD Requirement

The major blocks for the transmitter and receiver for the UMTS WCDMA FDD-mode communication system [37] [38] are shown in Figure 3. This target system was chosen because it is computationally intensive with tight constraints on latency.

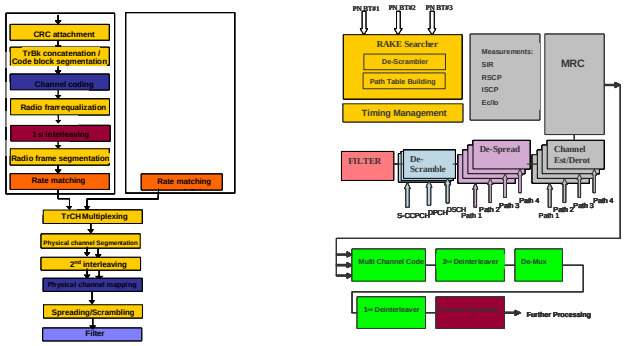

Figure 3. *WCDMA Transmission System*

For the receiver, the incoming I and Q signals are filtered using a Finite Input Response (FIR) representation of a Root Raised Cosine filter. This filter is a matched filter in that both the transmitter and receiver use the same filter. The filter is ideally implemented on a DSP. As bit-widths continue to widen, often consuming 10 to 14 bits in GSM and advanced communications systems, DSPs with appropriate datatypes may offer more efficient processing than custom silicon. After synchronization and multi path search, the strongest paths are descrambled, de-spread, equalized, and finally combined in the Maximal Ratio Combining (MRC) block. The output of the MRC block is a soft representation of the transmitted symbols. The soft bits are then de-multiplexed, de-interleaved, and channel decoded. On the receiver side there is also the measurement block responsible for measuring and reporting to the base station the communication channel characteristics as well as the received power at the terminal antenna. The power and communication channel characteristic measurements are necessary to keep the cell continuously functioning at maximum capacity.

Also shown in Figure 3 is the transmitter. In terms of computational requirements, it is significantly less complicated than the receive chain processing. Additionally, each step of the processing chain is described by the WCDMA standard. After the Cyclic Redundant Check (CRC) and transport block segmentation, the data is turbo or convolutional encoded, interleaved, assembled into radio frames, and then rate matched. The transport channels are parsed into physical channels, interleaved again, and mapped into transmit channels, spread, scrambled, and shaped before being sent to the DAC.

An important part of the WCDMA radio is generation of the RF front-end controls. This includes Automatic Frequency Control (AFC), Automatic Gain Control (AGC), and controls for the frequency synthesizers. These controls have tight timing requirements. Software implementations must have multiple concurrent accesses to frame data structures to reduce timing latencies. A multithreaded processor is an important component in parallelizing tasks and therefore reducing latency.

In WCDMA, turbo decoding is required to reduce the error rate. Because of the heavy computational requirements, this function is usually implemented in hardware. A high throughput WCDMA turbo decoder may require more than 5 billion operations per second. Implementing this function without special purpose accelerators requires high parallelism and innovative algorithms. We developed a new algorithm that reduces the latency and allows full 2 Mbits/s turbo decoding to be completely performed in software with no special purpose accelerators. The algorithm is described in [10].

3G SDR Implementation and Performance

Sandbridge Technologies has functional silicon for our multithreaded processor. The chip supports 8 hardware thread units and executes many baseband processing algorithms in real-time. In addition, a complete SDR product, which includes the SB3000 baseband processor as well as C code for the UMTS WCDMA FDD mode physical layer standard is being developed. Using its internally developed compiler, real-time performance on a 768 kbits/s transmit chain and a 2 Mbits/s receive chain has been achieved, which includes all the blocks shown in Figure 3. As shown in Figure 4, the SB3000 contains four Sandblaster cores and provides processing capacity for full 2 Mbits/s WCDMA FDD-mode including chip, bit, and symbol rate processing

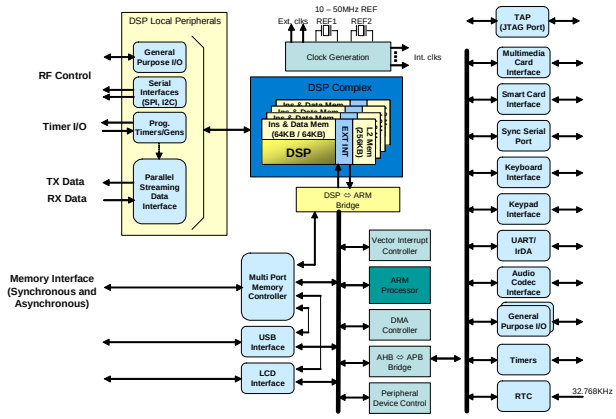

Figure 4. *SDR SB3000 Baseband Processor*

The measured performance requirements for IEEE802.11b, GPRS, WCDMA, and other communications systems as a function of SB3000 utilization for a number of different transmission rates are shown in Figure 5.

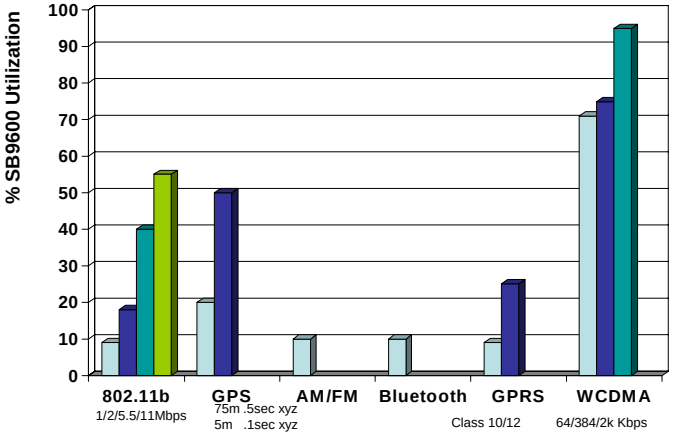

Figure 5. *SB3000 Baseband Performance*

5. Conclusions

Sandbridge Technologies has introduced a completely new and scalable design methodology for implementing multiple transmission 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 uses completely interlocked instruction execution providing software compatibility among all processors. Because of the interlocked execution, interrupt latency is very short. An interrupt may occur on any instruction boundary including loads and stores; this is critical for real-time systems.

The processor is combined with a highly optimizing compiler with the ability to analyze programs and generate DSP instructions. This obviates the need for assembly language programming and significantly accelerates time-to-market for new transmission systems.

To validate our approach, we designed our own 2 Mbits/s WCDMA, IEEE802.11b, GSM/GPRS, and GPS physical layers. First, we designed a MATLAB implementation to ensure conformance to the 3GPP specifications. We then implemented the algorithms in fixed point C code and compiled them to our platform using our internally developed tools. The executables were then simulated on our cycle accurate simulator thereby ensuring complete logical operation.

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. Our measurements confirm that our communications designs, including 2 Mbits/s WCDMA, will execute within field conformance requirements in real time completely in software on the SB3000 platform.

6. References

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

[2] Blaauw, Gerrit A. and Brooks Jr., Frederick P., "Computer Architecture: Concepts and Evolution", Addison-Wesley, Reading, MA, 1997.

[3] G.G. Pechanek, C.J. Glossner, W.F. Lawless, D.H. McCabe, C.H.L. Moller, and S.J.Walsh, "A Machine Organization and Architecture for Highly Parallel, Scalable, Single Chip DSPs", In Proceedings of the 1995 DSPx Technical Program Conference & Exhibition, pp. 42-50, 5/15-18/95, San Jose, California.

[4] B. Case, "Philips hopes to displace DSPs with VLIW", Microprocessor Report, December, 1997, pp. 12-15.

[5] O. Wolf and J. Bier, "StarCore Launches First Architecture", Microprocessor Report, Volume 12, Number 14, October, 1998, pp 1-4.

[6] J. Fridman and Z. Greenfield, "The TigerSHARC DSP Architecture", IEEE Micro, Vol. 20, No. 1, January, 2000, pp 66-76.

[7] J. Turley and H. Hakkarainen, "TI's New 'C6x DSP Screams at 1,600 MIPS", Microprocessor Report, Volume 11, Number 2, February, 1997, pp 1-4.

[8] J. Yoshida, "Java Chip Vendors set for Cellular Skirmish", EE Times, January 30th, 2001.

[9] 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.

[10] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A Software Defined Communications Baseband Design", IEEE Communications Magazine, Vol. 41, No. 1, pages 120-128, January, 2003.

[11] See http://www-ee.eng.hawaii.edu/~nava/HEP/introduction.html

[12] J. Gosling, "Java Intermediate Bytecodes", ACM SIGNPLAN Workshop on Intermediate Representation (IR95), pp. 111-118, January, 1995.

[13] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper", Sun Microsystems Press, October, 1995.

[14] T. Lindholm and F. Yellin, "Inside the Java Virtual Machine", Unix Review, Vol. 15, No. 1, pp. 31-39, January, 1997.

[15] J. Glossner, M. Schulte, and S. Vassiliadis, "A Java-Enabled DSP", in Embedded Processor Design Challenges, Systems, Architectures, Modeling, and Simulation (SAMOS), Lecture Notes in Computer Science 2268, E. Deprettere, J. Teich, and S. Vassiliadis editors, pp 307-325, Springer-Verlag, Berlin, 2002.

[16] D. M. Tullsen, S. J. Eggers, H. M. Levy, "Simultaneous Multithreading: Maximising on-chip Parallelism," in 22nd Annual International Symposium on Computer Architecture, pg 392—403, June, 1995.

[17] J. Glossner and S. Vassiliadis, "The Delft-Java Engine: An Introduction", Lecture Notes in Computer Science. Third International Euro-Par Conference (Euro-Par '97), pp 776-770, Passau, Germany, August, 1997.

[18] J. Glossner and S. Vassiliadis, "Delft-Java Dynamic Translation", Proceedings of the 25th EUROMICRO conference (EUROMICRO '99), Milan, Italy, September, 1999.

[19] K. Ebcioglu, E. Altman, and E. Hokenek, "A Java ILP Machine Based on Fast Dynamic Compilation", IEEE MASCOTS International Workshop on Security and Efficiency Aspects of Java, Eilat, Israel, January, 1997.

[20] M. Saghir, P. Chow, and C. G. Lee, "Towards Better DSP Architecture and Compilers", Proceedings of the International Conference on Signal Processing Applications and Technology, pages 658-664, October, 1994.

[21] European Telecommunications Standards Institute, Digital Cellular Telecommunications System, ANSI-C code for the GSM Enhanced Full Rate (EFR) speech codec (GSM 96.53), March, 1997, ETS 300 724.

[22] 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.

[23] B. Krepp, "DSP-Oriented Extensions to ANSI C", Proceedings of the International Conference on Signal

Processing Applications and Technology (ICSPAT '97), DSP Associates, 1997, pp. 658-664.

[24] R. Stallman, "Using and Porting GNU CC", Free Software Foundation, June 1996, version 2.7.2.1.

[25] 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.

[26] 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.

[27] D. Batten, S. Jinturkar, J. Glossner, M. Schulte, R. Peri, and P. D'Arcy, "Interaction Between Optimizations and a New Type of DSP Intrinsic Function", Proceeding of the International Conference on Signal Processing Applications and Technology (ICSPAT '99), Orlando, Florida, November, 1999.

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

[29] 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.

[30] 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.

[31] W. Hwu, "Super Block: An Effective Ttechnique for VLIW and Superscalar Compilation", Journal of Supercomputing, Volume 7, pp. 229-248.

[32] www.netbeans.org, "Netbeans IDE".

[33] B. Nichols, D. Buttlar, and J. Proulx-Farrell, "Pthreads Programming: A POSIX Standard for Better Multiprocessing", O'Reilly & Associates, Sebastopol, CA, 1st edition (September 1996).

[34] T. Boudreau, J. Glick, S. Greene, J. Woehr, and V. Spurlin, "NetBeans: The Definitive Guide", O'Reilly & Associates, Sebastopol, CA, 1st edition (October 15, 2002).

[35] J. Glossner, E. Hokenek, and M. Moudgill, "Wireless SDR Solutions: The Challenge and Promise of Next Generation Handsets", November 2002 Communications Design Conference, San Jose, California. Available at http://www.commdesignconference.com/proceedings.htm.

[36] J. Glossner, M. Schulte, and S. Vassiliadis, "Towards a Java-enabled 2Mbps Wireless Handheld Device", in Proceedings of the Systems, Architectures, Modeling, and Simulation (SAMOS) Conference, Samos, Greece, July 14-16, 2001.

[37] 3GPP, 3rd Generation Partnership Project, Technical Specifications V3.8.0 (2001-09)

[38] H. Holma and A. Toskala, "WCDMA For UMTS Radio Access For Third Generation Mobile Communications", John Wiley & Sons, 2001

[39] S. Jinturkar, J. Glossner, E. Hokenek, and M. Moudgill, "Programming the Sandbridge Multithreaded Processor", Accepted for publication at the 2003 Global Signal Processing Expo (GSPx) and International Signal Processing Conference (ISPC), March 31-April 3, 2003, Dallas, Texas.

[i] Results based on out-of-the-box C code. C64x IDE Version 2.0.0 compiled without intrinsics using -k -q -pm -op2 -o3 -d"WMOPS=0" -ml0 -mv6400 flags with results averaged over 425 frames of ETSI supplied test vectors. C62x IDE Version 2.0.0 compiled without intrinsics using -k -q -pm -op2 -o3 -d"WMOPS=0" -ml0 -mv6200 flags with results averaged over 425 frames of ETSI supplied test vectors. Starcore SC140 IDE version Code Warrior for StarCore version 1.5, Relevant Optimization Flags (Encoder Only): scc -g -ge -be -mb -sc -O3 -Og Other: No Intrinsic Used. Results based on execution of 5 frames. ADI Blackfin IDE Version 2.0 and Compiler version 6.1.5 compiled without intrinsics using -O1 -ipa -DWMOPS=0 –BLACKFIN with results averaged over 5 frames of ETSI supplied test vectors for the encoder only portion.