Sandblaster Low-Power Multithreaded SDR Baseband Processor

John Glossner[1,3], Michael Schulte[2], Mayan Moudgill[1], Daniel Iancu[1], Sanjay Jinturkar[1], Tanuj Raja[1], Gary Nacer[1], and Stamatis Vassiliadis[3]

[1] Sandbridge Technologies
1 North Lexington Ave.
White Plains, NY, 10512, USA
{jglossner,mayan,diancu,sjinturkar, traja,gnacer}@sandbridgetech.com
http://www.sandbridgetech.com

[2] University of Wisconsin
Dept. of ECE
1415 Engineering Drive
Madison, WI, 53706, USA
schulte@engr.wisc.edu
http://mesa.ece.wisc.edu

[3]Delft University of Technology
Electrical Engineering, Mathematics and Computer Science Department
Delft, The Netherlands
s.vassiliadis@its.tudelft.nl
http://ce.et.tudelft.nl

ABSTRACT

General-purpose processors have utilized complex and energy inefficient techniques to accelerate performance. In embedded DSP designs, power constraints have precluded general-purpose microarchitectural techniques. Rather than minimize average execution time, embedded DSP processors require the worst-case execution time to be minimized. Subsequently, Very Long Instruction Word (VLIW) processors have been employed, but architecturally visible side effects have imposed restrictions on parallelism due to interrupt and latency considerations – particularly if all loads must complete prior to servicing interrupts. In this paper, we present a low-power multithreaded interlocked (transparent) processor capable of parallelizing non-associative DSP arithmetic. We describe specific memory and logic techniques for reducing power dissipation and discuss how multithreading enables low-power optimization. We further describe the programming environment for our SDR DSP processor. Finally, we present automated results for complete physical layer processing of a number of communications protocols including WCDMA, GSM/GPRS, WLAN, and GPS.

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 realised. The design tradeoffs and implementation options inherent in meeting such demands highlight the extremely onerous requirements for next generation baseband processors. Tremendous hardware and software challenges exist to realize convergence devices.

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

1.1 SDR Based Approach

The SDR Forum [1] defines five tiers of solutions. Tier-0 is a traditional radio implementation in hardware. Tier-1, Software Controlled Radio (SCR), implements the control features for multiple hardware elements in software. Tier-2, Software Defined Radio (SDR), implements modulation and baseband processing in software but allows for multiple frequency fixed function RF hardware. Tier-3, Ideal Software Radio (ISR), extends programmability through the RF with analog conversion at the antenna. Tier-4, Ultimate Software Radio (USR), provides for fast (millisecond) transitions between communications protocols in addition to digital processing capability.

The advantages of reconfigurable SDR solutions versus hardware solutions are significant. First, reconfigurable solutions are more flexible allowing multiple communication protocols to dynamically execute on the same transistors thereby reducing hardware costs. Specific functions such as filters, modulation schemes, encoders/decoders 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.

SDR enabling technologies also have significant advantages from the consumer perspective. First, mobile terminal independence with the ability to "choose" desired feature sets is provided. Second, global connectivity with ability to roam across operators using different communications protocols is enabled. Third, future scalability and upgradeability provide for longer handset lifetimes.

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. SANDBLASTER PROCESSOR

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 of the same architecture should run unchanged on different implementations. An architectural function is *transparent* if its implementation does not produce any architecturally visible side effects. An example of a non-transparent function is the load delay slot made visible due to pipeline effects. Generally, it is desirable to have transparent implementations. Most DSP and VLIW implementations are not transparent and therefore the implementation affects the architecture [3][4][5][6].

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 two 4-element vectors, saturate, accumulate, saturate).

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

A challenge of visible pipeline machines (e.g. most DSPs and VLIW processors) is interrupt response latency. Visible memory pipeline effects in highly parallel inner loops (e.g. a load instruction followed by another load instruction) are not typically interruptible because the processor state cannot be restored. This requires programmers to break apart loops so that worst case timings and maximum system latencies may be acceptable.

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.

2.1 Low Power Architecture

Sandbridge Technologies has designed a multithreaded processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [7][8]. The Sandbridge Sandblaster design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and very low power dissipation.

The Sandblaster architecture is a compound instruction set architecture. Historically, DSPs have used compound instruction set architectures to conserve instruction space encoding bits. In contrast, VLIW architectures contain full orthogonality, but only encode a single operation per instruction field, such that a single VLIW is composed of multiple instruction fields. This has the disadvantage of requiring many instruction bits to be fetched per cycle, as well as significant write ports for register files. Both these effects contribute heavily to power dissipation.

In the Sandblaster architecture, specific fields within the instruction format may issue multiple sub-operations including data parallel vector operations. Restrictions may apply if a particular operation is chosen. In contrast, a VLIW ISA may allow complete orthogonality of specification and then either in hardware or through no operation (NOP) instructions fills in any unused issue slots.

In addition to compound instructions, the Sandblaster architecture also contains vector operations that perform multiple data parallel operations concurrently. As an example, Figure 1 shows a single compound instruction with three compound operations. The first compound operation, lvu, loads the vector register, vr0, with four 16-bit elements and updates the address pointer, r3, to point to the next four elements. The vmulreds operation reads four fixed point (fractional) 16-bit elements from vr0, multiplies each element by itself, saturates each product, adds all four saturated products plus an accumulator register, ac0, with saturation after each addition, and stores the result back in ac0. The vector architecture guarantees Global System for Mobile communication (GSM) semantics (e.g. bit-exact results) even though the arithmetic performed is non-associative [9].

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

Figure 1. Compound instruction for sum of squares inner loop

All the code shown in Figure 1 is encoded in a single 64-bit compound instruction. Each compound operation, including each vector operation, is specified with 21 bits. Like most DSP architectures, arbitrary operations are not specifiable within the same instruction. The instruction shown in Figure 1 may require more than 256 bits to encode on a VLIW machine. Furthermore, since the pipeline in a VLIW machine typically produces architecturally visible side effects (i.e. it is not transparent), it may take a deeply software pipelined loop to obtain single-cycle throughput, thereby exploding the instruction storage requirements. To further distinguish our approach from VLIW and exposed pipeline architectures, each instruction is completely interlocked and architecturally defined to complete with no visible pipeline effects. This is critical for fast interrupt processing.

Simple and orthogonal instruction formats are used for all instructions. The type of operation is encoded to allow simple decoding and execution unit control. Multiple operation fields are grouped within the same bit locations. All operand fields within an operation are uniformly placed in the same bit locations whether they are register-based or immediate values. As in VLIW processors, this significantly simplifies the decoding logic. Unlike a VLIW processor, our architecture is fully interlocked and transparent. In addition to the benefit of code compatibility, this ensures that many admissible and application-dependent implementations may be derived from the same basic architecture.

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

2.2 Low Power Microarchitecture

Figure 2 shows the microarchitecture of the Sandblaster multithreaded processor. In a multithreaded processor, all threads of execution operate simultaneously. An important point is that multiple copies (e.g. banks and/or modules) of memory are available for each thread to access. The Sandblaster architecture supports multiple concurrent program execution by the use of hardware thread units (called contexts). The architecture supports up to eight concurrent hardware contexts. The architecture also supports multiple operations being issued from each context.

As technology improves, processors are capable of executing at very fast cycle times. Current state-of-the-art performance for 130nm technologies can produce processors faster than 3GHz. Unfortunately, current high-performance processors consume significant power. If power-performance curves are considered for both memory and logic within a technology, there is a region in which you get approximately linear increase in power for linear increase in performance. Above a specific threshold, there is an exponential increase in power for a linear increase in performance. Even more significant, memory and logic do not have the same threshold. The Sandblaster implementation of multithreading allows the processor cycle time to be decoupled from the memory access time. This allows both logic and memory to operate in the linear region, thereby significantly reducing power dissipation. The decoupled execution does not induce pipeline stalls due to the unique pipeline design.

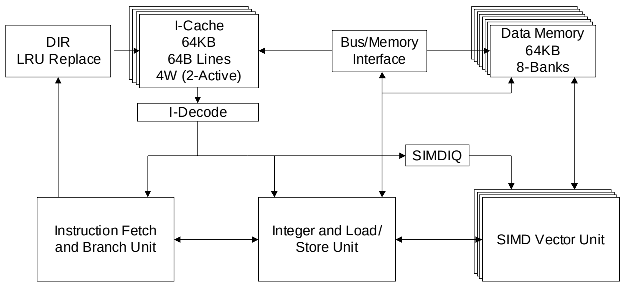


Figure 2. Sandblaster microarchitecture

An Instruction Cache Unit (ICU) stores instructions to be fetched for each thread unit. We use associative caches to reduce the likelihood of one thread evicting another thread's active program. In our implementation, a Thread Identifier register (not shown) is used to select whether the line from the left or right bank is evicted. This effectively reduces the complexity of the line selection. In a 4-way set associative cache, only one additional LRU bit is needed to select which of the two lines from the left or right bank should be evicted. This approach gives a complexity of *n(n-2)/8* LRU bits for a general n-way set-associative cache. This method of using thread information and banked memory access significantly reduces the complexity of the cache logic.

The pipeline for one particular implementation of the Sandblaster DSP is shown in Figure 3. The execution pipelines are different for various functions. The Load/Store (Ld/St) pipeline is shown to have 9 stages. It is assumed that the instruction is already in the cache. The first stage decodes the instruction. This is followed by a read from the General Purpose Register file. The next stage generates the address to perform the Load or Store. Five cycles are used to access data memory. Finally, the result for a Load instruction is written back (WB) to the referenced register file location. Once an instruction from a particular context enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction issuing from the same thread tries to use the result. Similarly, there are multiple (variable) stages for other execution pipelines. The integer unit has three execute stages for multiplication (I_MUL) and two execute stages for addition (ALU). The Vector unit has four execute stages - two for multiplication and two for addition.

Most interlocked architectures require significant interlock checking hardware and bypass logic for both correctness and performance reasons. Multithreading mitigates this effect. With the carefully designed pipeline shown in Figure 3, there is only one interlock that must actually be checked for in hardware – a long memory load or store. All other operations are guaranteed to complete prior to the same thread issuing a new instruction. This completely removes the power consuming interlock checks associated with most interlocked architectures.

Ld/St	Inst Dec	RF Read	Agen	XFer	Int Ext	Mem 0	Mem 1	Mem 2	WB
ALU	Inst Dec	RF Read	Exec 1	Exec 2	XFer	WB			
I_Mul	Inst Dec	RF Read	Exec 1	Exec 2	Exec 3	XFer	WB		
V_Mul	Inst Dec	VRF Read	Mpy1	Mpy2	Add1	Add2	XFer	WB	

Figure 3. Processor pipeline

3. PROGRAMMING THE SANDBLASTER SDR PROCESSOR

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. 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. 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.

3.1 Integrated Development Environment

The Sandbridge Technologies Integrated Development Environment (IDE) provides an easy to use graphical user interface to all the software tools. The IDE is based on the Open source Netbeans integrated development environment [13]. The IDE is the graphical front end to the C compiler, assembler, simulator and the debugger. The IDE provides the ability to create, edit, build, execute and debug an application. In addition, it provides the ability to mount a file system, access CVS, access the web and communicate with the Sandblaster hardware board.

3.2 Optimizing ANSI C Compiler

There are a number of issues that must be addressed in designing a DSP compiler. First, there is a fundamental mismatch between DSP data types 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.

As DSP C compilers have difficulty generating efficient code, language extensions have been introduced to high-level languages [11]. Typical extensions may include special 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. Sandbridge Technologies has built a new best-in-class optimizing ANSI C compiler for the Sandblaster DSP which does not rely on any extensions. This compiler applies a number of high performance compiler optimizations, which enable the generation of very efficient assembly code and obviates the need to write assembly code on this processor. In addition to applying a number of well know scalar and loop optimizations, the compiler applies DSP optimizations, supercomputer-class vector optimizations, and automatic parallel multithreaded optimizations.

ANSI C does not provide language features to program saturated DSP computations. Therefore, a programmer has to write emulation C code to perform the same operation. The assembly code generated for this emulation C code is very inefficient. Therefore, DSP compilers typically use mechanisms called intrinsics to substitute the emulation C code with equivalent assembly code[12]. However, this requires the user/compiler vendor to specify a predefined mapping between the snippets of assembly code and the emulation C code. Unfortunately, this forces the user to understand the details of the underlying processor's assembly language and the details of the compiler's operation. It also makes the code non-portable and difficult to maintain. This approach is used by compilers on a number of well-known DSPs [4][6].

However, the Sandbridge compiler does not use this approach. Sandbridge has developed proprietary semantic analysis techniques, which eliminate the need for intrinsics. A programmer writes C code in a processor independent manner - such as for a micro controller - focusing primarily on the function to be implemented. If saturated DSP operations are required, then the programmer writes the saturation emulation code in standard modulo C arithmetic. The compiler converts the C code into a dependence flow graph, analyzes the range of the arithmetic operations in the emulation code, propagates it across code segments, determines if it is a saturating or non-saturating operation and emits the correct assembly code. The semantic analysis does not rely on a coding style or patterns in the C source code. This makes the approach very general and applicable to any piece of C code. This technique has significant software productivity gains over intrinsic functions and does not force application developers to become DSP assembly language programmers.

Another important technique used by the compiler is the exploitation of SIMD instructions. The Sandbridge architecture uses SIMD instructions to implement vector operations. The compiler performs high performance inner and outer loop vector optimizations that use SIMD instructions to exploit the data level parallelism inherent in signal processing applications. These optimizations include vector load, store and multiply-add-reduce-saturate. In conjunction with loop optimizations, these provide very efficient and tight loops that can provide as many as 16 RISC operations in a single cycle. It is important to note that though saturation operations are non-associative (i.e. the order of computation is important), they do take advantage of the SIMD instructions. This is because the compiler was designed in conjunction with the processor and special hardware support allows the compiler to safely vectorize such non-associative operations [10].

Figure 4 shows the results of compilers for state-of-the-art DSPs on out-of-the-box AMR ETSI C code. The x-axis shows the DSP vendor and the y-axis shows the number of MHz required to compute frames of speech in real-time. In all cases, the highest optimization level that produced the logically correct code was used. The AMR code is completely unmodified and no special include files are used. Without using any compiler techniques such as intrinsics or special typedefs, the Sandbridge compiler is able to achieve real-time operation on the Sandblaster^{TM} core at hand-coded assembly language performance levels. Note that it is completely compiled from a high-level language.

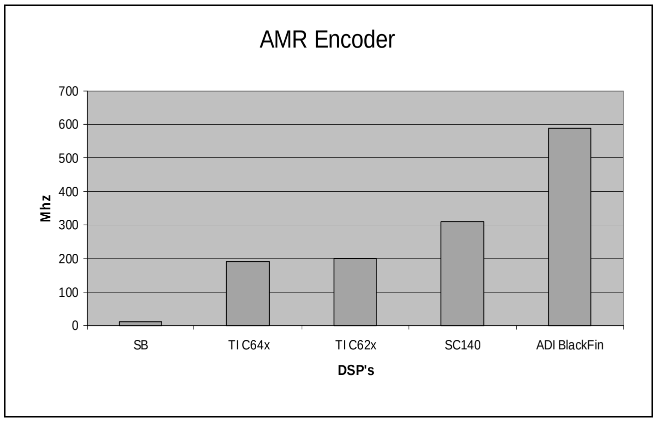


Figure 4. Out-of-the-box AMR ETSI encoder C code results

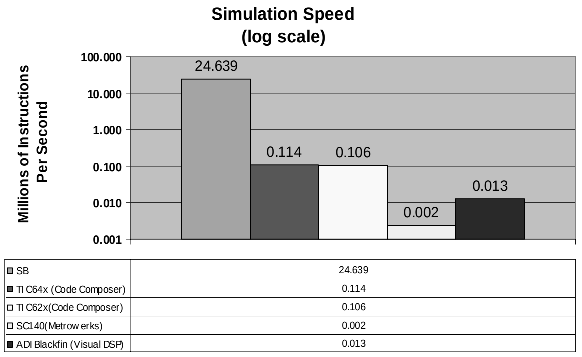
Since other solutions are not able to automatically generate DSP operations, they must use proprietary intrinsics to improve their performance. With intrinsic libraries the results for most DSPs are near the Sandbridge results. However, as mentioned earlier, these intrinsics make the code non portable, dependent on the names of the emulation C routines, and harder to maintain. The Sandbridge solution does not suffer from these disadvantages.

3.3 Simulation Environment

Efficient compilation is just one aspect of software productivity. Prior to having hardware, algorithm designers should have access to fast simulation technology. Sandbridge recognizes this fact and has provided an ultra fast cycle counting accurate simulator, which improves the programmer productivity. The simulator uses an architecture description of the underlying DSP and provides close to accurate cycle counts, but does not model the external memories or peripherals. However, the information provided by it is sufficient to develop the first executable version of an application.

The simulator is based on Just-in-Time code generation technology, which has been developed in house [14]. This technique is different than the interpretive techniques used in other DSP simulators. In the interpretive technique, the simulator models the target architecture, may mimic the implementation pipeline, and has data structures to reflect the machine resources such as registers. The simulator contains a main driver loop, which performs the *fetch, decode, data read, execute and write back* operations for *each* instruction in the target executable code. Note that these actions are performed *every time* the instruction is executed. In addition, numerous conditional statements have to be executed within the main driver loop as all combination of opcodes and operands have to be accounted for.

Our simulator uses the Just-in-Time dynamic translation technique. In this technique, the simulator takes advantage of the any apriori knowledge of the target executable and converts the target assembly code to host assembly code before executing any piece of code. Using this approach, the simulator generates host machine code for instruction fetch, decode and operand reads at the beginning of program execution (called the translation phase). The host instructions are then executed at the end of the translation phase. This approach eliminates the overhead of repetitive target instruction fetch, decode and operand read in the interpretive simulation model.



▨ SB	24.639
■ TI C64x (Code Composer)	0.114
□ TI C62x(Code Composer)	0.106
□ SC140(Metrowerks)	0.002
■ ADI Blackfin (Visual DSP)	0.013

Figure 5. Simulation speed of ETSI AMR encoder

Just-in-Time dynamic translation provides very fast simulation times. Figure 5 shows the post-compilation simulation performance of the same AMR encoder for a number of DSP processors. All programs were executed on the same 1GHz laptop Pentium computer. The Sandbridge simulator is capable of simulating 25 million instructions per second. This is more than two orders of magnitude faster than the nearest competitor and allows real-time execution of GSM speech coding on the Sandblaster simulator running on a 1 GHz Pentium. To further elaborate, while some DSPs cannot even execute the out-of-box code in real-time on their native processor, Sandbridge achieves multiple real-time channels on a simulation model of the processor.

4. RESULTS

Sandbridge Technologies has functional silicon for our multi-threaded 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. As shown in Figure 6, 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. As shown in Figure 7, using our internally developed compiler, a 768 kbits/s transmit chain and a 2 Mbits/s receive chain is supported in real-time. 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 7.

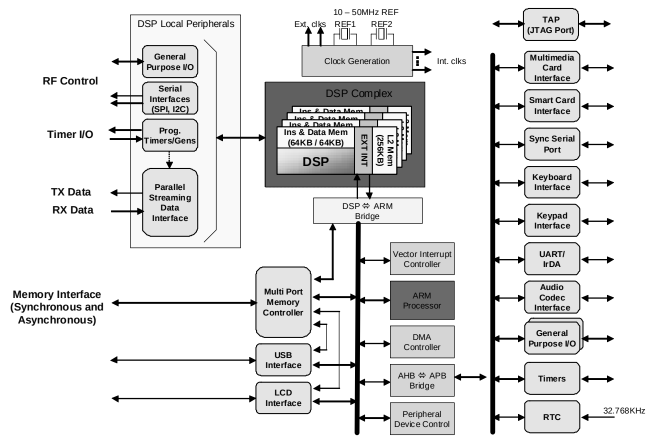


Figure 6. SDR SB3000 Baseband Processor

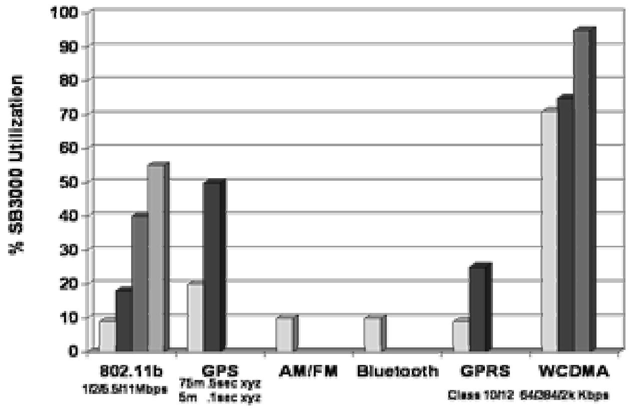


Figure 7. SB3000 bas band performance

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 multi-threaded 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 vectorizing compiler with the ability to automatically analyze programs and generate DSP instructions. The compiler also automatically parallelizes and multithreads programs. This obviates the need for assembly language programming and significantly accelerates time-to-market for 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. We then execute identical object files on our hardware. 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] G. Blaauw and F. Brooks Jr., *Computer Architecture: Concepts and Evolution*, Addison-Wesley, Reading, MA, 1997.

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

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

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

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

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

[8] J. Glossner, T. Raja, E. Hokenek, and M. Moudgill, "A Multithreaded Processor Architecture for SDR," *The Proceedings of the Korean Institute of Communication Sciences*, Vol. 19, No. 11, November, 2002, pp. 70-84.

[9] K. Jarvinen et al., "GSM Enhanced Full Rate Speech Codec," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1997, pp. 771-774.

[10] P. Balzola, M. Schulte, J. Ruan, J. Glossner, and E. Hokenek, "Design Alternatives for Parallel Saturating Multioperand Adders," *Proceedings of the International Conference on Computer Design*, September, 2001, pp. 172-177

[11] 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*, 1990, pp. 1065-1068.

[12] 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.

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

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