A SOFTWARE DEFINED COMMUNICATIONS BASEBAND DESIGN

John Glossner, Daniel Iancu, Jin Lu, Erdem Hokenek, and Mayan Moudgill
Sandbridge Technologies, Inc.
White Plains, NY
914-287-8500
glossner@sandbridgetech.com

ABSTRACT

Software Defined Radios (SDRs) offer a programmable and dynamically reconfigurable method of reusing hardware to implement the physical layer processing of multiple communications systems. An SDR can dynamically change protocols and update communications systems over the air as a service provider allows. In this paper we discuss a baseband solution for an SDR system and describe a 2Mbps WCDMA design with GSM/GPRS and 802.11b capability that executes all physical layer processing completely in software. We describe the WCDMA communications protocols with a focus on latency reduction and unique implementation techniques. We also describe the underlying technology that enables software execution. Our solution is programmed in C and executed on a multithreaded processor in real-time.

INTRODUCTION

Traditional communications systems have typically been implemented using custom hardware solutions. Chip rate, symbol rate, and bit rate co-processors are often coordinated by programmable DSPs but the DSP processor does not typically participate in computationally intensive tasks. Even with a single communication system the hardware development cycle is onerous often requiring multiple chip redesigns late into the certification process. When multiple communications systems requirements are considered, both silicon area and design validation are major inhibitors to commercial success. 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 many billions of operations per second at power efficiency levels appropriate for handset deployment. This has brought Software Defined Radio (SDR) to prominence and addresses a difficult portion of SDR implementation.

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.

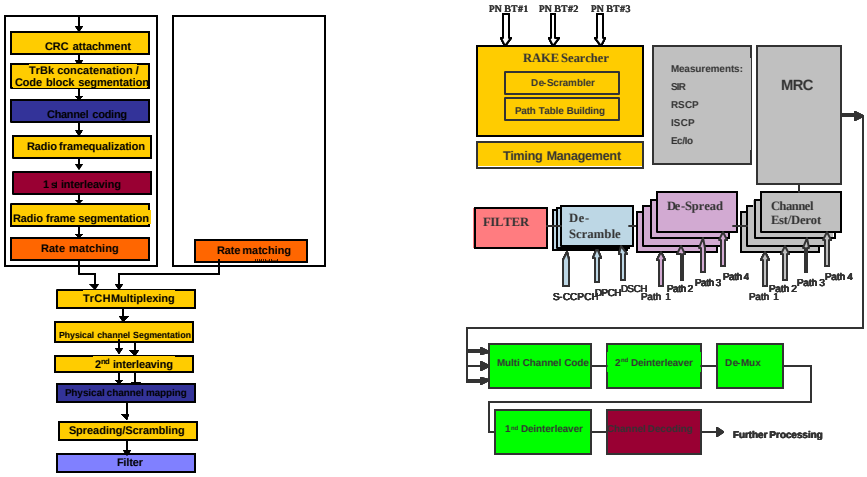

Figure 1. WCDMA Transmission System

In this paper we discuss a Tier-2 implementation of SDR that implements baseband processing in software. We first describe specific communications systems and elaborate on implementation techniques. Next we describe a multithreaded processor architecture capable of executing multiple baseband communications protocols. We then describe the software development environment including new compiler technologies that automatically generate signal processing instructions from ANSI C code. Finally, we describe an implementation of a multicore, multithreaded processor capable of executing multiple simultaneous communications systems completely in software.

COMMUNICATIONS SYSTEM DESIGN

Figure 1 shows the major blocks for both a transmitter and receiver for the UMTS WCDMA FDD-mode communication system. We choose to focus on WCDMA because it is computationally intensive with tight constraints on latency.

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 1 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, nearly every system implements this function 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.

High throughput Turbo Decoding

Turbo encoders and decoders are used in WCDMA communication systems due to their superior error correction capability. A turbo decoder has been demonstrated to approach the error correcting limit on both AWGN and Rayleigh fading channels.

A standard WCDMA turbo decoder consists of two concatenated SISO (Soft Input Soft Output) blocks separated by an interleaver and its inverse block, the deinterleaver. Upon reception of observations $y_1,...,y_K$ from the channel and prior information (a measure of the prior knowledge of a bit being 1 or 0), each SISO block computes log posterior ratios of each bit, a probabilistic measure of a bit being 1 or 0, with well-known forward and backward algorithms. The forward algorithm starts from a known initial state and calculates an intermediate variable $a_k(.)$ (the joint probability of the observations $y_1,...,y_K$ and the state at time k) from 1 to K. The backward algorithm starts from a known end state and calculates an intermediate variable $b_k(.)$ (the conditional probability of future observations given the state at time k) from K to 1. A SISO block computes the log posterior ratios of all bits and passes it to the other SISO blocks as a probabilistic estimate. This probabilistic estimate is called the extrinsic information. Additional SISO blocks use this as a prior information estimate. The two SISO blocks run in an iterative scheme, mutually exchanging extrinsic information and improving on the log posterior ratios. After the required number of iterations is completed, a hard decision about a bit being a 1 or 0 is made based on the log posterior ratios or soft information.

Simulations show that more than 90% of the computation of a turbo decoder is spent on the forward and backward algorithms. If the size of an observation sequence K is large, the time required for the computation of the forward and backward variables grows, creating a long latency as we go through the forward and backward algorithms. To reduce the latency of forward and backward algorithms for a software radio implementation, we divide the input data into M segments and simultaneously calculate the $a_k(.)$ and $b_k(.)$ for the M segments. In theory, this parallel scheme reduces the computation to 1 out of M in comparison to the original forward and backward algorithms (e.g. ½ for M=2).

An important issue in calculating $a_k(.)$ and $b_k(.)$ in parallel is the estimation of starting states. For the standard "one-shot" forward and backward algorithms, initial states (the state at the beginning and end) are known. But for multiple segments, the initial states for some segments must be estimated. We developed various fast initial state estimation methods that have little impact on the latency. Our simulation and experiment shows comparable results (within 1%) between the regular and parallel turbo decoders in terms of BER (bit error rate) with a significant improvement in latency.

SDR PROCESSOR DESIGN

Execution predictability in DSP systems often precludes the use of many general-purpose design techniques (e.g. speculation, branch prediction, data caches, etc.). Instead, classical DSP architectures have developed a unique set of performance enhancing techniques that are optimized for their intended market. These techniques are characterized by hardware that supports efficient filtering, such as the ability to sustain three memory accesses per cycle (one instruction, one coefficient, and one data access). Sophisticated addressing modes such as bit-reversed and modulo addressing may also be provided. Multiple address units operate in parallel with the datapath to sustain the execution of the inner kernel.

In classical DSP architectures, the execution pipelines were visible to the programmer and necessarily shallow to allow assembly language optimization. This programming restriction encumbered implementations with tight timing constraints for both arithmetic execution and memory access. The key characteristic that separates modern DSP architectures from classical DSP architectures is the focus on compilability. Once the decision was made to focus the DSP design on programmer productivity, other constraining decisions could be relaxed. As a result, significantly longer pipelines with multiple cycles to access memory and multiple cycles to compute arithmetic operations could be utilized. This has yielded higher clock frequencies and higher performance DSPs.

In an attempt to exploit instruction level parallelism inherent in DSP applications, modern DSPs tend to use VLIW-like execution packets. This is partly driven by real-time requirements which require the worst-case execution time to be minimized. This is in contrast with general purpose CPUs which tend to minimize average execution times. With long pipelines and multiple instruction issue, the difficulties of attempting assembly language programming become apparent. Controlling instruction dependencies between upwards of 100 in-flight instructions is a non-trivial task for a programmer. This is exactly the area where a compiler excels.

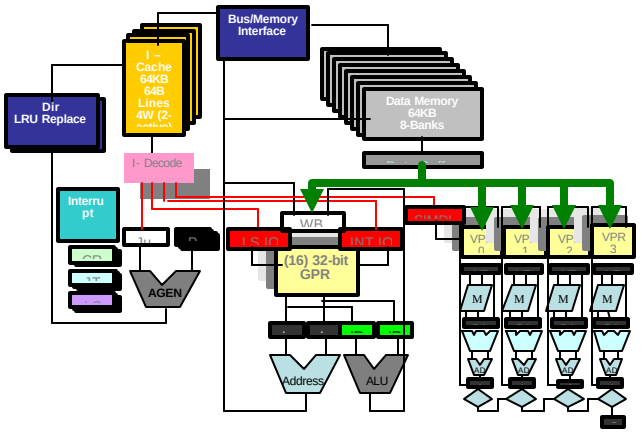

Figure 2. SDR Multithreaded Processor

A challenge of using VLIW DSP processors include large program executables (code bloat) that results from independently specifying every operation with a single instruction. As an example, a 32-bit VLIW 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).

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 results in high power dissipation that may not be acceptable 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 can not be restored. This requires programmers to break apart loops so that worst case timings and maximum system latencies may be acceptable.

Signal processing applications often require a mix of computational calculations and control processing. Control processing is often amenable to RISC-style architectures and is typically compiled directly from C code. Signal processing computations are characterized by multiply-accumulate intensive functions executed on fixed point vectors of moderate length. An additional trend for 3G applications is Java execution. Some carriers are requiring Java functionality in their handsets.

As shown in Figure 2, Sandbridge Technologies has designed a multi-threaded processor capable of executing DSP, Control, and Java code in a single compound instruction set optimized for handset radio applications [10]. The Sandbridge 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 instructions that are grouped into packets for execution. The resulting combination provides for efficient Control, DSP, and Java processing execution.

SDR DSP SOFTWARE

Programmer productivity is also a major concern in complex DSP and SDR applications. Processors capable of performing baseband processing must perform DSP operations. Because most classical DSPs are programmed in assembly language, it takes a very large software effort to program an application. For modern speech coders 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 for DSPs were available, significant advantages in software productivity could be achieved.

A DSP compiler should be designed jointly with the architecture based on the intended application domain. Trade-offs are made between the architecture and compiler subject to the application performance, power, and price constraints.

However, there are a number of issues that must be addressed in designing a DSP compiler. First, there is a fundamental mismatch between DSP datatypes and C language constructs. A basic 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. Saturation arithmetic is analogous to a stereo dial. As you increase the volume it always gets louder until it reaches the limit of the system precision. When this limit is reached the value still remains at the maximum value. If the volume control worked like C language modulo arithmetic the volume would immediately return to “0” after overflowing the precision limit and no sound would be heard. A DSP compiler must deconstruct and analyze the C code for the semantics of the operations represented and generate the underlying fixed point operations.

A second problem for compilers is that previous DSP architectures were not designed with compilability as a goal. To maintain minimal code size, multiple operations

were issued from the same compound instruction. Unfortunately, to reduce instruction storage, a common encoding was 16-bits for all instructions. Often, three operations could be issued from the same 16-bit instruction. While this is good for code density, orthogonality[1] suffered. Classical DSPs imposed many restrictions on the combinations of operations and the dense encoding implied many special purpose registers. This resulted in severe restrictions for the compiler and poor code generation.

Early attempts to remove these restrictions used VLIW instruction set architectures with nearly full orthogonality. To issue four multiply accumulates 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 [2]. Typical additions may include special type support for 16-bit datatypes (Q15 formats), saturation types, multiple memory spaces, and SIMD parallel execution support. These additions often imply a special compiler and the code generated may not be emulated easily on multiple platforms. As a result, special language constructs have not been successful.

In addition to language extensions, other high-level languages have been used. BOPS produced a Matlab compiler which offers exciting possibilities since Matlab is widely used in DSP algorithm design. Difficulties with this approach include Matlab's inherent 64-bit floating point type not being supported on most DSPs. On DSPs which do support 32-bit floating point, precision analysis is still required.

For algorithm design, tensor algebra has been used [3]. Attempts have been made to automate this into a compilation system [5]. The problem of this approach is that highly skilled algorithm designers are still required to describe the initial algorithm in tensor algebra. However, this approach holds promise because the communications and parallelism of the algorithm are captured by the tensor algebra description.

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

Intrinsic Functions

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

An intrinsic function has the appearance of a function call in C source code, but is replaced during pre-processing by a programmer-specified sequence of lower-level instructions. The replacement specification is called the intrinsic substitution or simply the intrinsic. An intrinsic function is defined if an intrinsic substitution specifies its replacement. The lower-level instructions resulting from the substitution are called intrinsic instructions [6].

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

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

Early intrinsic efforts, like inlined asm statements, inhibited DSP compilers from optimizing code sequences [7]. A DSP C compiler could not distinguish the semantics and side effects of the assembly language constructs and this resulted in compiler scheduling hazards. Other solutions which attempted to convey side-effect free instructions have been proposed. These solutions all introduced architectural dependent modifications to the original C source.

Intrinsics which eliminated these barriers have been explored. The main technique is to represent 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 6x.

The main detractor of intrinsics is that it moves the assembly language programming burden to the compiler

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

writers. More importantly, each new application may still need a new intrinsic library. This further constrains limited software resources.

Supercomputer Compiler Optimizations

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

In addition to classic compiler optimizations, there are some advanced optimizations which have proven significant for DSP applications. Software pipelining in combination with aggressive inlining and VLIW scheduling has proven effective in extracting the parallelism inherent in DSP and general purpose applications.

Interestingly, some DSP applications (speech coding for example) do not exhibit significant data dependence. A program that is data dependent will give significantly different execution times and execution paths through the program depending upon what data input the program receives. When programs are not heavily influenced by the dataset choice, profile directed optimizations may be effective at improving performance [8]. In profile driven optimization the program is executed based on a set of data inputs. The results of the program and the execution path through the program are then fed back into the compiler. The compiler uses this information to group highly traversed paths into larger blocks of code which can then be optimized and parallelized. These techniques, when used with VLIW scheduling have proven effective in DSP compilation. However, the results may still be less than half as efficient as assembly language programs.

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

Exploiting data parallelism is an important factor in optimizing for DSP applications. While both a VLIW and Vector datapath can exploit such parallelism, extracting it from C code can be a difficult challenge. Most VLIW scheduling technique focus on exploiting instruction level parallelism from code sequences. What is often needed to reveal data parallelism is a vectorizing compiler. For a compiler to be able to vectorize loops coded in C it may have to significantly reorder the loops either splitting or jamming them together. Often loops are nested multiple levels deep. It may not be possible to vectorize the inner loop without first vectorizing the outer loops. These types of optimizations are typically only found in supercomputer compilers but they significantly assist in uncovering data parallelism from arbitrary C code.

Compiler Technologies

It is well recognized that the best way to design a DSP compiler is to develop it in parallel with the DSP architecture. Future compiler-architecture pairs will not be afforded the luxury of large numbers of intrinsic libraries. Just as modern RISC processors do not require assembly language programming, neither will future DSP applications.

A unique aspect of the modern compiler is that DSP operations are automatically generated using a technique called semantic analysis. In semantic analysis, a sophisticated compiler must search for the meaning of a sequence of C language constructs. A programmer writes C code in an architecture independent manner - such as for a micro controller - 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.

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

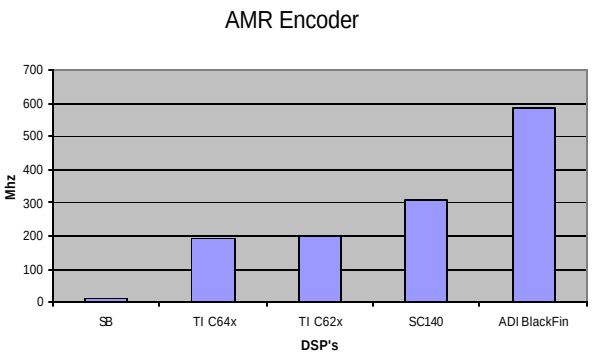

Figure 3. Out-of-the-box AMR ETSI Encoder C code results

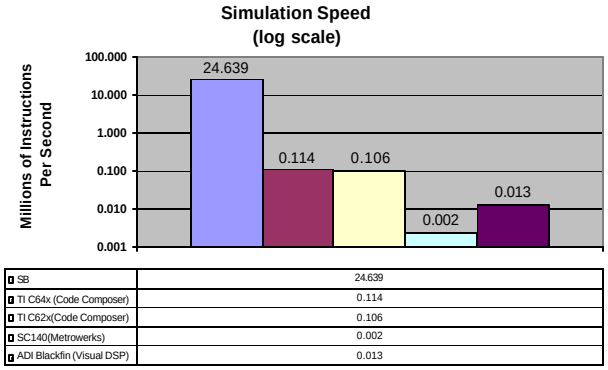

Figure 4. Simulation speed of ETSI AMR Encoder

Figure 3 shows the results of various compilers on out-of-the-box ETSI C code. The y-axis shows the number of MHz required to compute frames of speech in real-time. 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 compiler is able to achieve real-time operation on the baseband 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 ours but they only apply to the ETSI algorithms whereas the described compiler can be applied to arbitrary C code.

Ultra-fast Software Simulation

Efficient compilation is just one aspect of software productivity. Prior to having hardware, algorithm designers should have access to fast simulation technology. Figure 4 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 tools are capable of simulating 25.6 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 a Pentium simulation model. To further elaborate, while some DSPs can not even execute the out-of-the-box code in real-time on their native processor, Sandbridge achieves multiple real-time channels on a simulation model of processor. We achieved this by using our own compilation technology to accelerate the simulation.

RTOS and IDE

The programming interface for the multithreaded processor is generic ANSI C code. In keeping with an easy-to-use programming philosophy, access to multithreading is provided through the open standards of either Java threads or POSIX pthreads. Since nearly all general purpose platforms support these standards it is simple to port programs to the Sandbridge platform. An API is also supported to allow access to the underlying thread scheduler and for fast porting of 3rd party RTOS's.

An IDE is also provided based on the opensource Netbeans IDE. Our netbeans implementation has been extended to work with C programs and allows for both Java and C to be debugged using a common environment.

SDR IMPLEMENTATION

Previous 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. 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 takes a completely new approach to communications system design.

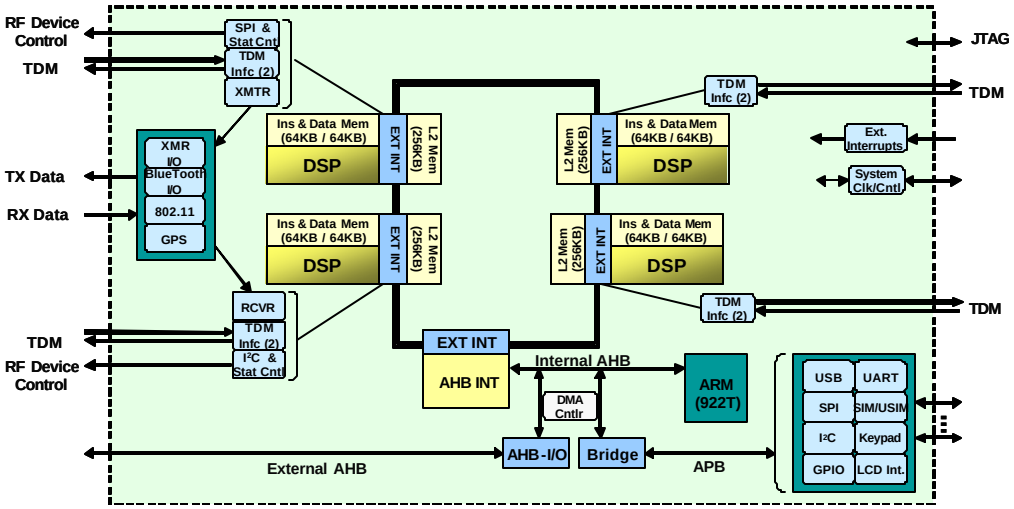

Figure 5. SDR SB9600 Baseband Processor

Rather than designing custom blocks for every function in the transmission system, an SDR implements a processor capable of executing operations appropriate to broadband communications. A small and power efficient core is then highly optimized and replicated to provide a platform for 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.

RESULTS

Sandbridge Technologies has developed complete SDR product, including baseband processor as well as C code for the UMTS WCDMA FDD mode physical layer standard. Using an internally developed compiler, real-time performance on a 768kbps transmit chain and a 2Mbps receive chain has been achieved, which includes all the blocks shown in Figure 1. The entire transmit chain including bit, symbol, and chip rate processing requires less than 400MHz of processor capacity to sustain a 768 kbps transmit capability.

Figure 6 shows the performance requirements for 802.11, GPRS, and WCDMA as a function of SB9600 utilization for a number of different transmission rates. Providing processing capability for 2Mbps WCDMA FDD-mode also provides sufficient processing capability for 802.11b and even concurrent capacity for multiple communications systems.

Handset SDR Product

Figure 5 shows the SB9600™ baseband chip. It contains multiple Sandblaster™ cores and an ARM microcontroller that functions as an applications processor. The performance of the chip is more than sufficient to sustain a 2Mbps WCDMA 3G transmission in real time. It also supports executing the digital basebands for GPRS, 802.11b, Bluetooth, and GPS.

The chip contains a number of digital peripheral interfaces for moving data in and out of the chip such as AD/DA for Tx and Rx data, TDM ports, and an AMBA bus. High speed Universal Serial Bus (USB) provided easy connectivity to external systems. Control and test busses such as JTAG, SPI, and I²C allow the chip to control RF and front end chips.

Initial silicon of the core is available. The final chip core will support 9.6 billion multiply accumulates per second at less than 500mW power consumption.

SUMMARY

A new and scalable design methodology has been introduced for implementing multiple transmission systems on a single 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 for all future processor designs. 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 2Mbps WCDMA physical layer. First, we designed a MATLAB implementation to ensure conformance to the

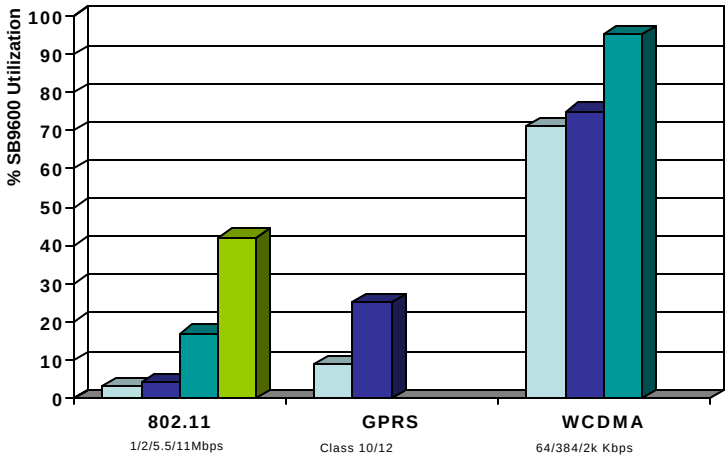

Figure 6. Baseband Communications System Performance

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 that runs at up to 100 million Sandblaster™ instructions per second on a high end Pentium thereby ensuring complete logical operation. Having designed our own 3GPP compliant RF front end using commercially available components, we execute complete RF to IF to baseband and reverse uplink processing in our lab. Our measurements confirm that our WCDMA design will execute within field conformance requirements in real time completely in software on the SB9600™ platform.

In addition to WCDMA, we have also implemented 802.11b and GSM/GPRS. These protocols also execute in real-time on the developed platform.

REFERENCES

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

[2] 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.

[3] J. Granata, M. Conner, R. Tolimieri, **"The Tensor Product: A mathematical Programming Language for FFTs and other Fast DSP Operations,"** *IEEE SP Magazine*, pp. 40-48, January 1992.

[4] C.J. Glossner, G.G. Pechanek, S. Vassiliadis, and J. Landon, **"High-Performance Parallel FFT Algorithms on M.f.a.s.t. Using Tensor Algebra."** *Proceedings of the Signal Processing Applications Conference at DSPx'96*, March 11-14, 1996, pp. 529-536, San Jose Convention Center, San Jose, California.

[5] N. P. Pitsianis, **"A Kronecker Compiler for Fast Transform Algorithms"**, *8th SIAM Conference on Parallel Processing for Scientific Computing*, March, 1997.

[6] 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.

[7] 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.

[8] 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.

[9] J. Glossner, E. Hokenek, and M. Moudgill, **"Wireless SDR Solutions: The Challenge and Promise of Next Generation Handsets"**, *Accepted for publication at the 2002 Communications Design Conference*, September 2002, San Jose, CA.

[10] J. Glossner, E. Hokenek, and M. Moudgill, **"Multithreaded Processor for Software Defined Radio"**, Accepted for publication at the *2002 SDR Forum Conference*, November, 2002, San Diego, CA.

John Glossner is CTO & EVP of Engineering at Sandbridge Technologies. Prior to co-

founding Sandbridge, John managed the Advanced DSP Technology group, Broadband Transmission Systems group, and was Access Aggregation Business Development manager at IBM's T.J. Watson Research Center. Prior to IBM, John managed the software effort in Lucent/Motorola's Starcore DSP design center. John received a Ph.D. in Computer Architecture from TU Delft in the Netherlands for his work on a Multithreaded Java processor with DSP capability. He also received an M.S. degree in Engineering Management and an M.S.E.E. from NTU. John also holds a B.S.E.E. degree from Penn State. John has more than 40 publications and 12 issued patents.

Daniel Iancu received the M.Sc. and the Ph.D. degrees in Physics and Electronics both from the University of Cluj-Napoca, Romania, in 1976 and 1986, respectively. In 1980 he took a teaching position with the Faculty of Physics, the Electronics Department of the same University, where besides teaching he spent ten years of research in various areas of high frequency Physics and Electronics with applications in DSP and Communication Systems. After arriving in US in 1990, he took several jobs in DSP and Communication field of applications. Since 2000 he is with Sandbridge Technologies as Director of Emerging Technology.

Jin Lu, graduated from Cornell University with a Ph.D. in EE, has worked in the industries of consumer electronics, telecommunication, and control system for over 10 years. He specializes in communication, interactive multimedia, digital consumer electronics, and real-time systems. He has published many papers in the area of networking, control systems, and signal processing. His current interests include real-time algorithms and implementation of wireless technologies.

Erdem Hokenek received BS and MS degrees from Technical University, Istanbul (Turkey) and PhD from Swiss Federal Institute of Technology (ETH Zurich, Switzerland). After his PhD in 1985, he joined IBM T. J. Watson Research Center where he worked on the advanced development of POWER and PowerPC processors for the RS/6000 Workstations. He also worked in various technical and management positions on the high performance compilable DSP and Cross Architecture Translations. He is co-founder of Sandbridge Technologies Inc.

Mayan Moudgill obtained a Ph.D. in Computer Science from Cornell University in 1994, after which he joined IBM at the Thomas J. Watson Research Center. He worked on a variety of computer architecture and compiler related projects, including the VLIW research compiler, Linux ports for the 40x series embedded processors and simulators for the Power 4. In 2001, he co-founded Sandbridge Technologies, a start-up that is developing digital signal processors targeted at 3G wireless phones.