Implementing Communications Systems on an SDR SoC

John Glossner[1], Daniel Iancu[1], Mayan Moudgill[1], Sanjay Jinturkar[1], Gary Nacer[1], Stuart Stanley[1], Andrei Iancu[1], Hua Ye[1], Michael Schulte[1,3], Mihai Sima[1,2], Tomas Palenik[5], Peter Farkas[1,5], and Jarmo Takala[1,4]

[1]Sandbridge Technologies	[2]Univ. Victoria	[3]UW Madison	[4]Tampere Univ.	[5]Slovak Univ.
1 North Lexington Ave.	British	Madison	of Technology	of Technology
White Plains, NY 10601 USA	Columbia	Wisconsin	Tampere	Bratislava
glossner@SandbridgeTech.com	CANADA	USA	FINLAND	SLOVAKIA

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 present techniques for implementing communications systems in software. We describe briefly the SB3011 platform and programming environment. We then present a number of useful techniques that can be used to implement working systems. We further describe in which systems these techniques are implemented.

Index Terms— *Software Defined Radio (SDR), Multithreaded Processor, Digital Signal Processor (DSP), Multicore, Communications Systems*

1. INTRODUCTION

Historically, processor-based Software Defined Radios (SDRs) have not had enough performance to implement modern communications systems. 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.

To enable physical layer processing in software, processors should support many levels of parallelism. Since many algorithms have stringent requirements on response time, multithreading is an integral technique in reducing latencies. In the Sandbridge SB3011 processor design [1] thread-level parallelism is supported by providing hardware for up to 8 independent programs to be simultaneously active on a single Sandblaster core. This approach hides the latency in physical layer processing.

In addition to thread-level parallelism, the processor also supports data-level parallelism through the use of a vector unit. In the inner kernel of signal processing or baseband routines, the computations appear as vector operations of moderate length. Filters, FFTs, correlations, etc., all can be specified in this manner. Efficient, low power support for data level parallelism effectively accelerates inner loop signal processing.

To accelerate control code, the processor supports issuing multiple operations per cycle. Since control code often limits overall program speed-up due to Amdahl's Law, it is helpful to allow control code and vector code to execute simultaneously. This is provided through a compound instruction set and multithreaded organization. The Sandblaster core provides instruction level parallelism by allowing multiple operations to issue in parallel. For example, a branch, an integer, and a vector operation may all issue simultaneously [2]. In addition, many compound operations are specified within an instruction class such as load with update, and branch with compare.

Obtaining full utilization of multiprocessor resources has historically been a difficult challenge. Much of the programming effort can be spent determining which processors should receive data from other processors. Execution cycles are often wasted for data transfers. Statically scheduled machines such as Very Long Instruction Word architectures and visible pipeline machines with wide execution resources complicate programmer productivity by requiring manual tracking of up to 100 in-flight instruction dependencies. When non-associative DSP arithmetic is present, nearly all compilers are ineffective and the resulting burden falls upon the assembly language programmer. A number of these issues have been discussed in [1].

With concurrent multithreaded hardware and a multithreaded software programming model, it is possible for a kernel to be developed that automatically schedules software threads onto hardware threads. It should be noted that while the hardware scheduling may be fixed, the software should be free to use any scheduling policy desired. The POSIX pthreads open standard provides cross platform capability as the library is compilable across a number of systems including Unix, Linux, and Windows [2].

The Sandbridge approach includes a complete parallelizing tool chain which removes the need for tedious DSP assembly language programming The compiler

automatically vectorizes data parallel operations and can even vectorize non-associative fixed-point (saturating) datatypes. The compiler is also able to automatically generate threads for the processor [3]. Furthermore, ultra-fast simulation, profiling, and debugging of code that is embodied in the Sandblaster development environment is a key enabler of fast application development.

With a tool chain capable of automatically generating parallel DSP code and a high-performance low-power chip fully functional, the physical layers of multiple communications protocols such as WCDMA, GSM/GPRS, 1xEVDO, TD-SCDMA, NTSC Video Decode, WiMax, WiFi, GPS, AM/FM radio, DVB, and SINCGARS have been implemented. Real-time response has been achieved and the effectiveness of our approach has been validated.

In the process of implementing these systems, a number of techniques have been developed for both low power optimization and efficient execution. Real-time constraints add another level of complexity. The remainder of this paper discusses the techniques and optimizations used.

2. PROGRAMMING TECHNIQUES

Rather than designing custom blocks for every function in the communication 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.

2.1. General Programming

The Sandblaster processor provides access to a fixed number of threads. These threads are called hardware threads or contexts. However, an application (written in C) can contain a virtually unlimited number of POSIX threads (pthreads also called software threads), which are scheduled on the hardware threads by the real-time operating system (RTOS).

The Sandblaster programming interface provides the ability to create, destroy, and join the threads through the common include library <pthreads.h>. This open mechanism is completely portable across multiple processors and compilers. In fact, the compiler uses the exact same pthreads mechanism when automatically generating threads. The underlying operating system supports the inherent pthreads schedule model to prioritize and schedule threads. The interface also models a number of peripherals include A/D and D/A converters, General Purpose IO (GPIO), and control peripherals. The RTOS has been kept lightweight to reduce overhead.

2.2. Coding Guidelines

We have previously published coding guidelines that allow our compiler to take advantage of parallel resources [4]. Most of these guidelines are generic in the sense that if followed, compilation on other platforms (e.g. x86) should be improved. All are high-level language practices. The most salient points are summarized:

- Minimize the use of malloc().
- Pass arrays explicitly and not enclosed within structures
- Use arrays of shorts as the widest vector datatype in the Sandblaster processor is 16-bits. The tools will also vectorize 32-bit vectors but only half as many execute in parallel.
- Use floating point only when necessary as it is emulated.
- Avoid unrolling loops manually as the compiler can do it more efficiently.

3. COMMUNICATION SYSTEMS

In designing communications systems there are often multiple algorithms that may be used in an implementation. Some algorithms provide better system performance and higher computational cost, while other techniques may involve simply changing the implementation of a component. In this section we describe some of the more interesting techniques used in our system.

3.1. Time Domain versus Frequency Domain

In the implementation of a GPS receiver we found that a time domain technique could be used very effectively, rather than a frequency domain technique used by many hardware implementations [5]. The key equation is Equation (1).

In Equation (1), we desire to minimize the detection error for a particular satellite *i.* Therefore, we force the two conditions $f - f_i = 0$ and $\varphi_i = 0$ implying a correction for the Doppler shift as well as for the phase shift for each satellite. Conforming to the Fourier transform shifting property, the condition $f - f_i = 0$ can be achieved either through frequency or time domain shifting.

$$\sum_{i=0}^{N_s-1}\int_{-\infty}^{+\infty}\chi(t)dt = \sum_{i=0}^{N_s-1}\left(\frac{D_i(k,n+n_i')\delta(f-f_i)(\cos\varphi_i + j\sin\varphi_i}{2}\right) + \sum_{i=0}^{N_s-1}E_i \qquad (1)$$

In hardware implementations, the carrier is tracked by advancing or retarding the Local Oscillator (LO) frequency and phase (frequency domain shift) conforming to the output of a Phase Locked Loop (PLL) circuit. For each visible satellite the integral defined in Equation (1) is calculated separately resulting in multiple parallel processes with each process executed by a separate

dedicated hardware block called a channel. Each channel must have its own LO, PLL and Pseudo Number (PN) generator.

A software implementation may be more efficiently implemented in the time domain. The sampled data is time domain shifted by pointer manipulation; the pointer is shifted forward or retarded depending on the direction of the Doppler shift. The LO in this case is a sin-cosine table. Even though each satellite has a different carrier frequency due to different Doppler shifts, the table is the same for all satellites. The phase condition is achieved by a digital (software) PLL, resulting in an additional shift for each carrier on the already shifted data. The PN sequences are stored in memory.

With a multithreaded processor, all the integrals in (1) are executed in parallel. The number of virtual channels is dynamically allocated by the processor, depending on the available resources and real time criteria.

Hardware implementations can easily implement the condition $f - f_i = 0$ with a Voltage Controlled Oscillator (VCO) used for LO. In software, the frequency condition cannot be exactly achieved due to a limited sampling rate. Shifting in the time domain will also induce spurious frequencies which will increase the detection error. To reduce detection errors, over-sampling is required.

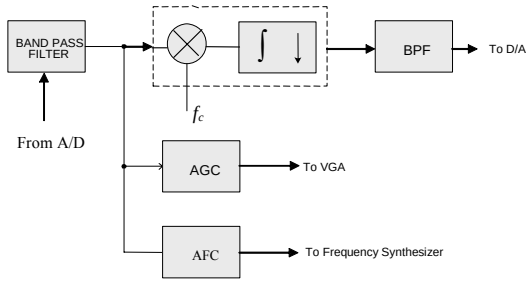

Figure 1.Software Blocks for AM Receiver

3.2. Software Generated Control Signals

Figure 1 shows the blocks that were implemented in software for an Amplitude Modulated (AM) receiver [7]. The eight times over-sampled signal from the A/D is first filtered using a second order IIR band-pass filter, centered at the carrier frequency, with 3 dB attenuation bandwidth of 5 KHz. The filtered signal is then multiplied with the cosine of the sampled signal f_c and integrated over eight samples. After integration, the data goes through 1:16 decimation, filtering using a 96 tap 80dB FIR band-pass filter, rescaling, and DC removal. Finally, the data is sent to the D/A. The automatic frequency control (AFC) and the automatic gain control (AGC) functions are also implemented in software. The coefficients for the filters are pre-computed and stored in nonvolatile memory for each carrier in the AM frequency band. This is a new algorithm for implementing an AM radio. Using control signals generated by software, we are able to control the demodulation of AM signals using significantly less complex operations than otherwise achievable.

3.3. Using Lookup Tables

Converting cycle intensive divisions and trigonometric functions into lookup tables is a technique used in our DVB receiver. One important function in communications is receiver synchronization. There is always a difference between the carrier frequency and the Local Oscillator due to reference frequency lack of precision. Minimizing the difference is the goal of frequency de-rotation. The de-rotation function is described by the following equation:

$$e^{j(w+\Delta w)nT_s} . e^{-j.\Delta w \cdot nT_s} = e^{j \cdot w \cdot nT_s}$$

where T_s is the sampling frequency and $\Delta\omega$ is the frequency error.

Conforming to the previous equation, the error $\Delta\omega$ is cancelled by multiplying the incoming sampled waveform with a locally generated sequence of frequencies $\Delta\omega$, usually stored in memory. However when memory size and access time are an issue, the LUT can be replaced by computations using the following approximation: $\sin(\omega + \Delta\omega) \cong \sin(\omega) + \cos(\omega) \cdot \Delta\omega$ and $\cos(\omega + \Delta\omega) \cong \cos(\omega) - \sin(\omega) \cdot \Delta\omega$. In both cases the error $\Delta\omega$ is supplied by the PLL or Frequency Lock Loop (FLL) block.

3.4. Using Mathematical Manipulations for Parallelization

In most communication systems the base band processing requires both I and Q orthogonal sequences. In the following we show that through mathematical manipulations it is possible to use only one input sequence, either I or Q, resulting in HW simplification without significant increase in computational complexity.

We assume an OFDM communication system such as DVB-T/H for example. The Fourier transform of the base band signals, I_{BB} and Q_{BB} yields

$$c^*_{0,0,k} = \sum_{k=0}^{M-1} \left(I_{BB}[n] + jQ_{BB}[n] \right) \cdot W^{-nk} \qquad (2)$$

Where $c^*_{0,0,k}$ is the complex amplitude of the k^{th} carrier, nT is replaced by n, $W^{-nk}=e^{-j2\pi nk/M}$ and M represents the number of samples in one symbol time interval less the guard time. Equation (2) is the most common approach for further base band processing. Another approach uses only sequence I (or Q): Suppose $I^{(1)}(t)$ and $I^{(2)}(t)$ are two consecutive symbols.

$$I^{(1)}(t) = \sum_{n=0}^{N-1} \sum_{k=K_{\min}}^{K_{\max}} c^{(1)}{}_{0,0,k} \times \Psi_{0,0,k}(nT) \cdot \cdot \delta(t - nT)$$

$$I^{(2)}(t) = \sum_{n=0}^{N-1} \sum_{k=K_{\min}}^{K_{\max}} c^{(2)}{}_{0,0,k} \times \Psi_{0,0,k}(nT) \cdot \delta(t - nT)$$

where $\Psi_{0,0,k}(nT)$ is described in [9] and δ is the Dirac function.

The complex Fourier transform of the two real symbols can be written as:

$$\sum_{k=0}^{M-1} \left(I^{(1)}[n] + jI^{(2)}[n] \right) \cdot W^{-nk} = A[k] + jB[k] \qquad (3)$$

From Equation (3), the two symbols are

$$c^{(1)^*}{}_{0,0,k} = \frac{1}{2}\left(A[k] + A[M-k]\right) + \frac{j}{2}\left(B[k] - B[M-k]\right) \quad (4)$$

$$c^{(2)^*}{}_{0,0,k} = \frac{1}{2}\left(B[k] + B[M-k]\right) - \frac{j}{2}\left(A[k] - A[M-k]\right) \quad (5)$$

Both approaches have the same final result from a mathematical point of view, except for a scaling factor.

3.5. Using Min Sum in LDPC Codes

Low-density parity-check (LDPC) codes are linear block codes with the parity-check matrix containing only a few ones compared to the number of zeros. The performance of the LDPC codes comes very close to the Shannon limit for most of the different propagation channels. Until recently the implementation of the LDPC codes was prohibitive because of the computational complexity.

The block sructure of the parity check matrix, specified in IEEE Std. 802.16e, and the organization of these blocks in tiers are suitable for multithreaded decoder implementation. The nonzero blocks are only rotated versions of the identity matrix while the degree of the columns inside each tier is either one or zero. This structure will allow multiple threads to access independently the extrinsic information of the variable nodes. The decoding algorithm consists of horizontal update (or check function evaluations) and vertical update (summations). Because the matrix is sparse, it would be very inefficient to store the extrinsic information of variable nodes in a simple 2D array of size at most 1152 x 2304. Instead, only the nonzero elements are stored. This cuts down the necessary number of rows to the number of ones in any column (at most 6 for the proposed WiMax LDPC). This way, the necessary memory space will only be (6+1) x 2304 x 2 = 32 kB. The extra row is necessary to keep track of the correct vertical index of the processed values. If handled properly, the compressed storage of variable nodes values doesn't prohibit multithreaded processing.

4. CONCLUSIONS

We have described a number of SDR techniques for implemented communications algorithms in software. These techniques range in scope from simple high level coding guidelines to sophisticated software resampling techniques that reduce hardware component counts. The ability to dynamically select communications system components provides the flexibility to reduce the computational burden (i.e. power) if the system parameters permit.

5. REFERENCES

[1] J. Glossner, M. Schulte, M. Moudgill, D. Iancu, S. Jinturkar, T. Raja, G. Nacer, and S. Vassiliadis, "Sandblaster Low-Power Multithreaded SDR Baseband Processor", Proceedings of the 3rd Workshop on Applications Specific Processors (WASP'04), pp. 53-58, Stockholm, Sweden, September 7th, 2004.

[2] B. Nichols, D. Buttlar, and J. Farrell, Pthreads Programming: A POSIX Standard for Better Multiprocessing, O'Reilly Nutshell Series, Sebastopol, CA, September, 1996.

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

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

[5] J. Glossner, D. Iancu, V. Kotylar, H. Ye, E. Hokenek, and M. Moudgill, "Software Defined Global Positioning Satellite Receiver", Proceedings of Software Defined Radio Technical Forum, pp. HW-2-001, 1-5, Orlando, Florida, November 2003.

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

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

[8] J. Glossner, D. Iancu, G. Nacer, S. Stanley, E. Hokenek, and M. Moudgill, "Multiple Communication Protocols for Software Defined Radio", IEE Colloquium on DSP Enable Radio, pp. 227-236, September 22-23, 2003, ISIL, Livingston, Scotland.

[9] TBD.