Multiple Communication Protocols for Software Defined Radio

John Glossner[†,*], Daniel Iancu[*], Gary Nacer[*], Stuart Stanley[*],
Erdem Hokenek[*], and Mayan Moudgill[*]

[*]Sandbridge Technologies, Inc.
1 North Lexington Ave.
White Plains, NY 10601 USA
jglossner@sandbridgetech.com

[†]Delft University of Technology
Computer Engineering
Electrical Engineering, Mathematics and Computer Science
Delft, The Netherlands

Abstract

We present an inexpensive multi-protocol communication system implemented entirely in software comprised of Global Positioning System (GPS), Wireless Local Area Network (WLAN) and Bluetooth (BT). All three communications protocols share the same Analog to Digital converter and are sampled at the same hardware sampling rate. The hardware sampling clock is provided by a high stability crystal oscillator at a sampling frequency suited for the GPS receiver. For the WLAN and BT systems, the sampling rate is modified with an interpolation filter and a Phase Lock Loop (PLL) both implemented in software. The communication system is able to execute simultaneously in software the physical layer processing and data sample timing for GPS, WLAN, and BT employing Sandbridge Technologies' multithreaded Sandblaster SB9600 Processor.

1. Introduction

The concept of Software Defined Radio (SDR) has been considered for some time but the speed and cost of existing processors has prohibited commercial implementation. Today, with the increasing capabilities of Digital Signal Processors (DSPs) and the requirements for accelerated time to market, SDR is emerging as an important commercial technology. The concept of reusing hardware (H/W) blocks by reconfiguring the Radio Frequency (RF) front end while also reconfiguring the baseband software (S/W) remains the touchstone of SDR [1].

When discussing the ultimate goal of an ideal SDR-based communications systems, there are three key enabling technologies that determine the viability of a pure, multi-protocol, 'any standard' commercial SDR hand held device:

1) An ultra-wideband, low power, high dynamic range RF front end
2) A high rate, high dynamic range, low power data converter section and
3) A powerful, computationally efficient, low power baseband signal processor

An ideal SDR must be capable of low noise RF reception over a large bandwidth, with the ability to discern the weakest signals at system sensitivity from all other received signals within all desired bands, at power levels up to their respective maxima. Efficient, highly linear broadband transmission of simultaneous, multi-carrier signals at differing power levels across the entire dynamic range encompassing all applicable protocols is required. Additionally, a high sampling rate intermediate frequency (IF) signal data conversion to capture (or produce) the combined downlink (uplink) spectra of all the simultaneous waveforms, operating across the multiple communications systems is required. Finally, sufficient computational capacity to extract and process multiple simultaneous baseband

waveforms in real time, perform all the Layer 2 and 3 signaling and control functions, and run the commercial device's Operating System (OS), and software applications presents many challenges. The difficulty is further enhanced in a mobile wireless environment where Doppler shifts and multi-path effects are prevalent.

Current SDR systems fielded or in development today possess some of the important attributes of an ideal SDR system. For example, Wireless Base Stations may generate multi-user downlink data for a single protocol standard in software and then deliver it into a very high rate, high dynamic range data converter. The resulting aggregated analog multi-carrier signal is then up-converted onto an RF carrier frequency and fed into an ultra linear Multi Carrier Power Amplifier (MCPA) with some form of linearization (feed-forward or pre-distortion error correction) to produce tens of watts of RF output for the Tx antenna. This system constitutes a highly over-sampled SDR application, whereby all signal waveforms are brought together and sent through a single data converter channel. This has become possible recently due to SiGe Data converters such as Maxim's 500Msps MAX5195 SiGe 14-bit DAC with LVPECL digital inputs. While RF front ends are limited in re-configurability, flexible SDR solutions for baseband processing are an attractive choice in base-station applications enabling easy protocol and feature updates.

Another diverse application of SDR technology is the United States Department of Defense's Joint Tactical Radio Systems (JTRS) program. JTRS is developing hardware that can reconfigure and run multiple channels of various waveforms of differing protocols over a broad RF spectrum. In contrast to the base station transmit chain, greater emphasis is placed on phase coherence between the multiple RF front end's local oscillator (LO) frequencies and sampling clocks. This facilitates detection and tracking of multiple waveforms across multiple RF front ends [2]. Typically, this requires that all sub-system clocks be either synthesized from a common reference clock, or if independent, they must each be used to control their respective data converters and fed to a DSP to permit tracking. In this architecture, each stream of baseband data must be buffered and multiplexed into the DSP.

Commercial hand held SDR devices pose significant challenges for low-power, low-cost implementation. The simultaneous combination of all the desired characteristics for the RF front end and data converter does not yet exist. However, multiband subsets of key frequencies are attainable. The migration from component intensive super heterodyne transceivers toward Direct Conversion Receivers (DCR) has reduced the number of components necessary to implement multiband receivers. Sigma-delta converters have been limited to 100 Msps, rendering them best suited to single mode devices. Migration to advanced pipelined converters that are tailored to lower power consumption is anticipated to provide significant SDR advantages.

Software baseband processing (i.e. DSP processing) has improved dramatically with process technology improvements. Power efficient architectures can now provide real-time baseband execution of simultaneous waveforms suitable for handheld device applications. A particular problem has been the capability of generating multi-protocol clocking and signal tracking. Since each waveform may have different sampling rates, it is desirable to minimize the number of hardware generated clocks. In this paper, to accommodate various protocols, we implement a hybrid approach where a single hardware clock is used to receive samples. We then employ software techniques to resample and track the signals. Using this technique it is possible to provide support for multiple waveforms with minimal hardware.

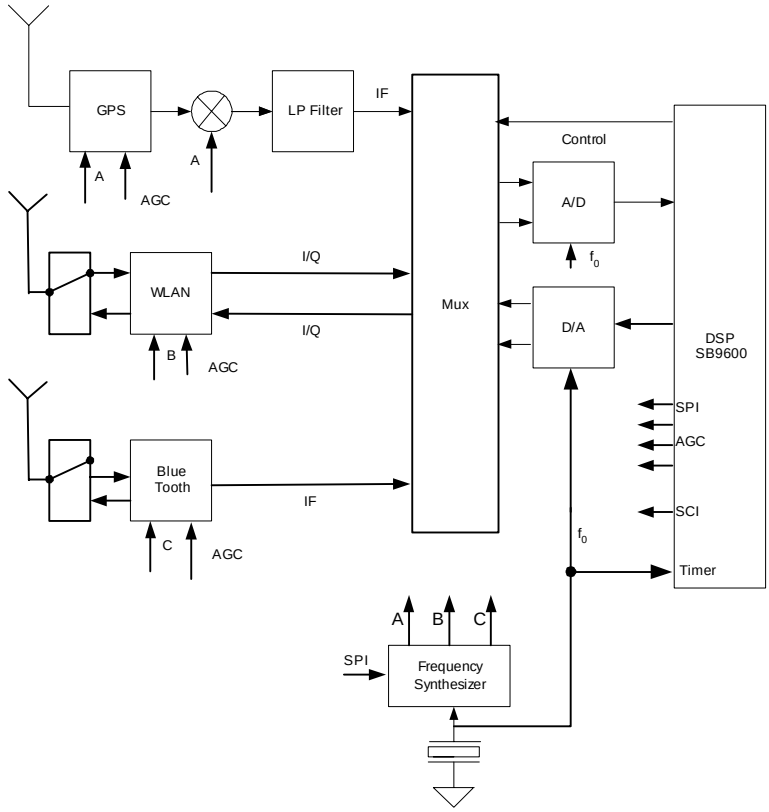


Figure 1. HW diagram of the multi-protocol communication system.

2. Overall Hardware Architecture

Figure 1 shows the hardware to validate our SDR design. Three RF blocks are shown corresponding to GPS, WLAN, and Bluetooth waveforms. We use a dedicated RF front end for each communication protocol and select under software control which waveform to execute. Each RF front end typically consists of an antenna feeding through a diplexer to one or more dedicated LNA / PA / RF conversion chips. As an example, for the receive chain, each of the three RF front end signals are received from separate antennae and then data multiplexed into a single stream. The RF single stream is then converted into a digital I/Q signal that is oversampled. The amount of oversampling varies based on the waveform being processed. This effectively allows the baseband processor to view the RF receiver as a broad spectrum, multiband receiver. A similar process is used for the transmit chain.

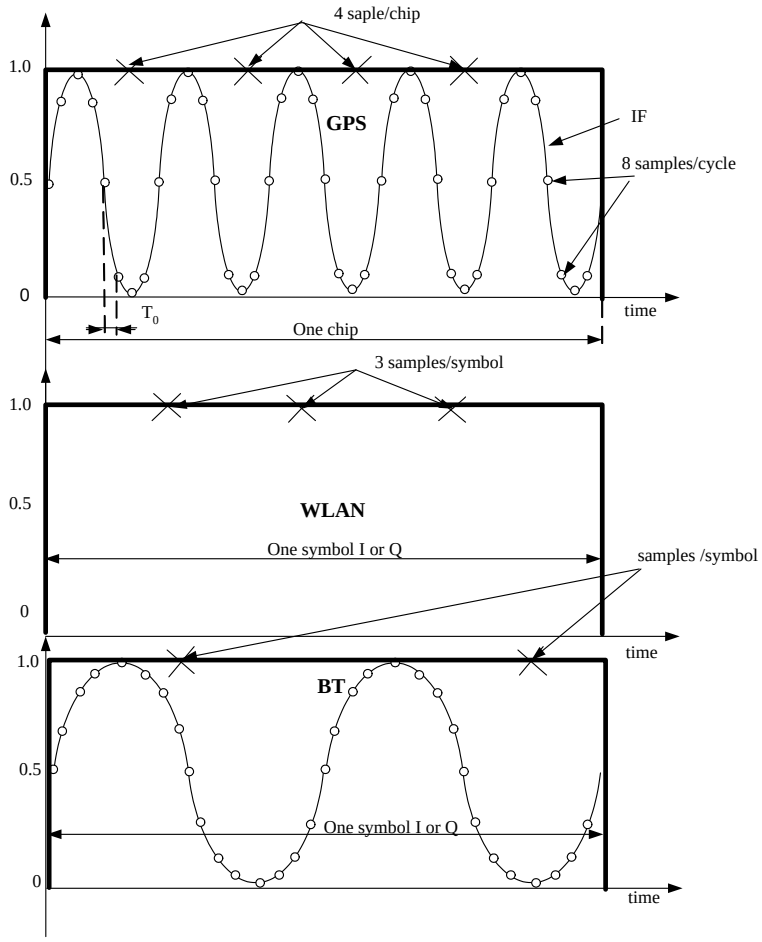
2.1. RF Hardware

The top of Figure 1 shows the GPS RF front end. It consists of a single chip receiver including a Low Noise Amplifier (LNA) and down-conversion stages. An external Surface Acoustic Wave (SAW) Band-pass Filter (BPF) performs cleanup after the LNA. The last stage of down-conversion and subsequent Low Pass Filtering (LPF) are done separately off chip to produce a band limited 4.092 MHz Intermediate Frequency (IF) signal. This IF is then multiplexed into the data converter. The digitized IF is then demodulated and subsequently all chip rate processing is performed in software on the Sandblaster processor.

The middle of Figure 1 shows the WLAN 802.11b RF front end. It consists of a single chip LNA / down-converter / demodulator chain, and a modulator/ up-converter / PA chain, both

joining at the Transmit / Receive (T/R) switch that feeds into the antenna port. An external SAW BPF (not shown) is used at the IF to remove the image frequencies and spurious mixing products, and an external PLL synthesizer chip is used to generate the LO for up / down conversion. The demodulator provides analog filtered downlink differential I and Q signals to the A/D data converter, through the multiplexer. The D/A portion of the data converter feeds the multiplexer which drives data samples into the modulator for the uplink.

The bottom of Figure 1 shows the Bluetooth RF front end. Architecturally it is similar to the WLAN RF front end. The Bluetooth receiver down converts the RF carrier frequency to an IF signal which is again fed to the A/D through the multiplexer. The D/As are also shared through the multiplexer, which sends the transmit data to the modulator. The IF signal is then up-converted and fed to the driver amplifier or PA.



Protocol	GPS	BT	WLAN
IF sampling rate	32.736 MHz	32 MHz	33 MHz
IF over sample	8	16	—
Chip/Symbol rate over sample	4	2	3

Figure 2. Waveform Sampling Rates Generated by Software

2.2. Reference and Sampling clocks for the RF Front Ends

A high stability reference frequency source is used to lock the synthesizers and generate the sample rate clock. A common sample rate is employed across all three RF front ends and its frequency was selected as an integral multiple of the chip rate for the GPS protocol to facilitate satellite signal acquisition and tracking. Since the GPS receiver must deal with the reception of the time stamped data from the multiple satellites along with the process of deriving the time offsets from Doppler shifts, the sample rate was chosen to minimize GPS processing. WLAN and Bluetooth data both get sampled by the data converter at the GPS frequency. The proper sampling rates for both WLAN and Bluetooth are then generated through software interpolation in baseband processing. The interpolation is linear and done at each sample. The formula is given by [3]:

$$Y_n=X_n+n\,(T_0-T_s)/T_s(X_{n+1}-X_n) \qquad \text{(Eqn. 1)}$$

where Y_n is the interpolated sample, X are the acquired samples, T_0 is the desired sampling rate, and T_s is the constant sampling rate. The waveform sampling rates chosen for each communications protocol is shown in Figure 2.

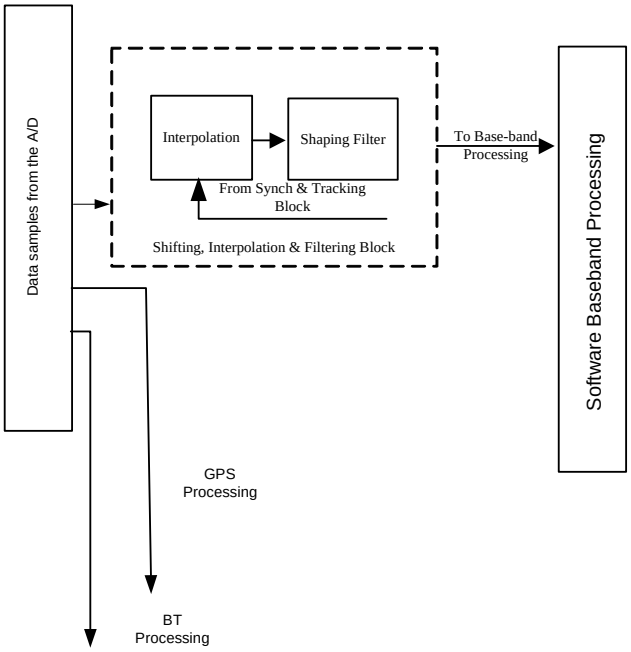


Figure 3. WLAN Shifting Interpolation Filtering Block

2.3. Baseband Processing

The Sandbridge Sandblaster processor executes all of the baseband processing for each of the communications waveforms in software. Figure 3 shows the wireless LAN shifting intrerpolation filtering block. All three protocols use this software block prior to physical layer processing. It consists of the interpolation block that executes the interpolations for the desired sampling rate based on Equation (1). It also takes input from the tracking and synchronization and based upon the decision of this block, it can advance or retard an additional sample in order to keep the phase locked. Waveform shaping is then performed by an FIR filter. The filter coefficients are pre-calculated and stored in the memory. After the filtering, the data is further processed by physical layer software. The WLAN implementation

is different from both the GPS and Bluetooth implementations in that the sampled signal is already in I/Q format.

Figure 4 shows the block diagram describing the GPS software baseband processing. The GPS IF of 4.096 MHz arrives digitized from the A/D converter 8x over-sampled at 32.736 MHz. A buffer of data is collected, filtered, and processed separately for every satellite carrier received to determine each Doppler shift [4]. The Doppler shift compensation is realized in software by a combined shift - PLL (Phase Lock Loop) function executed on the data samples in the shifting, filtering and interpolation block. A local oscillator (LO) function, also implemented completely in software, then modulates the samples. This LO implementation utilizes a lookup table; the multiplied samples are then integrated over one sine/cosine cycle period. The resulting down-sampled waveform is correlated against a 1023 chip pseudo number (PN) sequence. The resulting I and Q data is further processed by the upper layers in the searching, tracking and control processing. The I/Q data is then squared and summed to provide tracking synchronization. When the WLAN or BT waveforms are executing, GPS synchronization is maintained by counting the number of incoming samples to serve as a time reference.

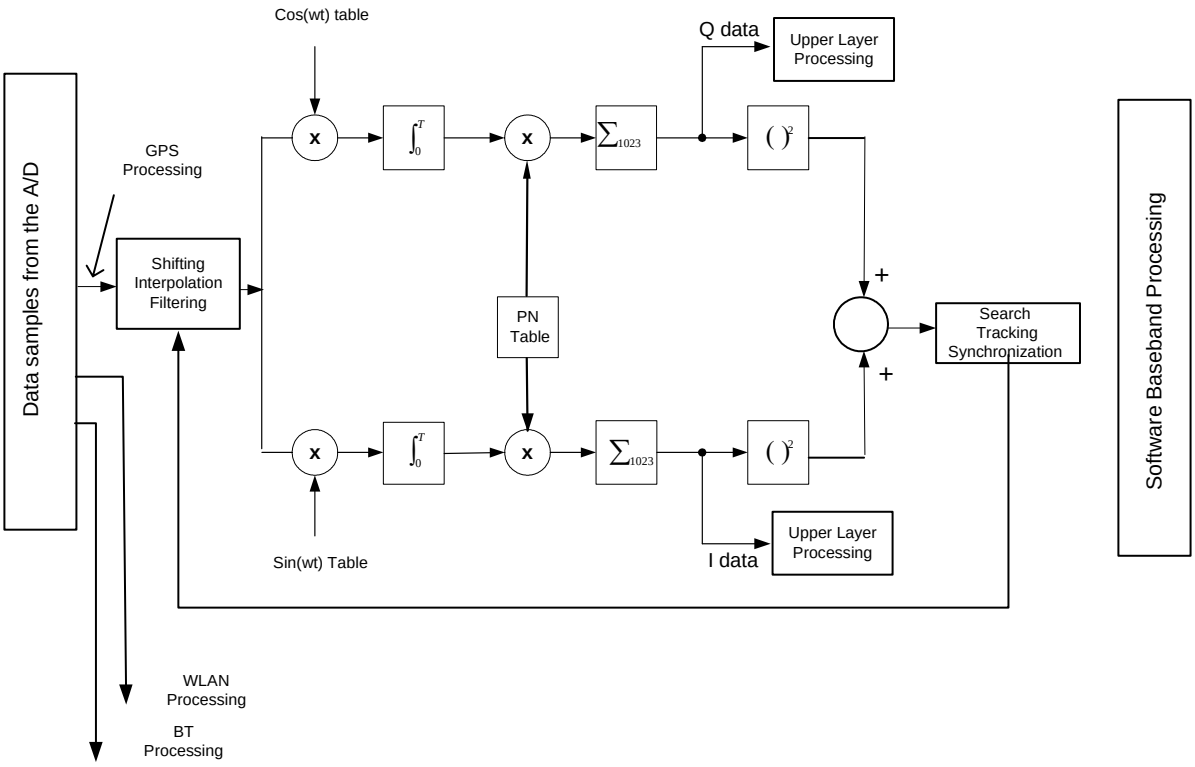


Figure 4. GPS signal processing block diagram.

Figure 5 shows the Bluetooth baseband processing chain which is similar to the GPS processing chain. The blocks are again implemented completely in software. Data from the A/D converters are received by the Sandblaster processor and are interpolated and filtered. The 2 MHz IF is then demodulated by a Gaussian Frequency Shift Key (GFSK) demodulator. The demodulated data is used for synchronization, for determining a Received Signal Strength Indication (RSSI), and for use by the packet processing block. The hopping and Automatic Gain Control (AGC) controls are also executed in software. For the transmitter,

the data packets are GFSK modulated and interpolated for the appropriate sample rate. The digitized data is then sent to an external D/A.

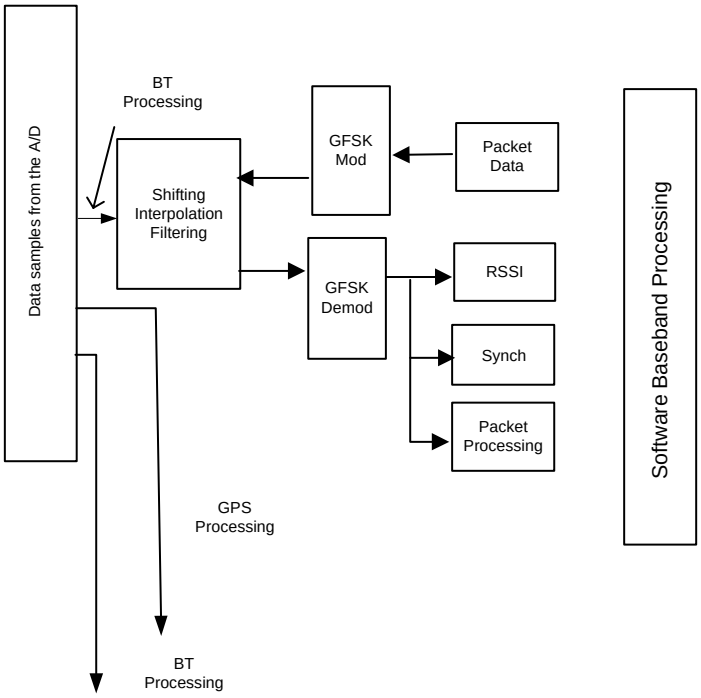


Figure 5. Blue Tooth signal processing block diagram.

3. Software Implementation

In the previous section we focused on the software for timing and resampling of the waveforms. Developing software on the Sandbridge platform is similar to developing applications code for general purpose computing platforms. Using open standards and common programming methodologies combined with sophisticated software tools, a programmer can write in a high-level language and the Sandbridge tools automatically optimize the code for the Sandblaster platform. In this section we describe the tool chain and methodology for generating the complete physical layer in software.

3.1. Processor Tools

Figure 6 shows the Sandblaster tool chain generation [5]. The platform is programmed in a high-level language such as C, C++, or Java. The program is then translated using an internally developed supercomputer class vectorizing, parallelizing compiler. In addition to standard optimizations, the compiler automatically recognizes DSP (e.g. saturating fixed point) operations and generates optimized code. The compiler is also capable of automatically creating threads for parallel execution. The tools are driven by a parameterized resource model of the architecture. The source input to the tools, called the Sandbridge architecture Description Language (SaDL), is a collection of python source files that guide the generation and optimization of the input program and simulator. The compiler is retargetable, in the

sense that it is able to handle multiple possible implementations specified in SaDL and produce an object file for each implementation. The platform also supports many standard libraries (e.g. libc, math, etc.) that may be referenced by the C program. The compiler generates an object file optimized for the Sandblaster architecture. The object file may then be directly executed on a Sandblaster processor.

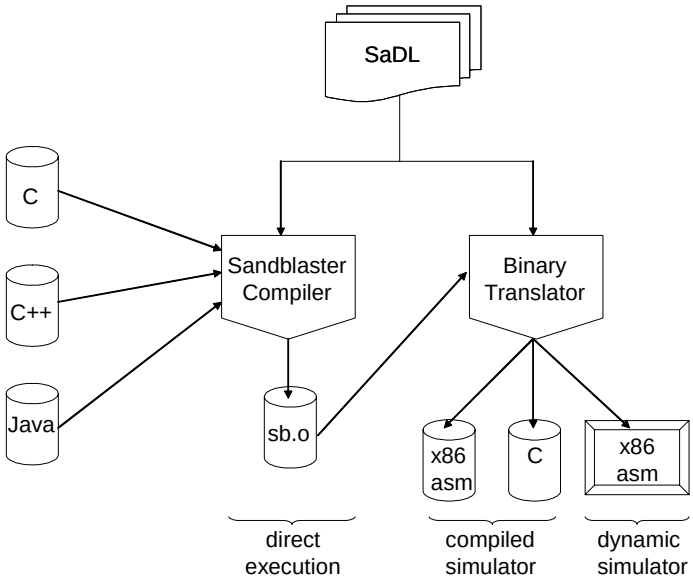


Figure 6. Tool Chain Generation

The tools are also capable of producing dynamic and static simulators from the same executable object file. A binary translator/compiler is invoked on the host simulation platform. The inputs to the translator are the object file produced by the Sandblaster compiler and the SaDL description of the processor. From these inputs, it is possible to produce a statically compiled simulation file. If the host computer is an x86 platform, the translator may directly produce x86 optimized code. If the host computer is a non-x86 platform, the binary translator produces a C file that may subsequently be processed using a native compiler (e.g. gcc). Using the dynamic simulator, we achieve 25 to 100 million Sandblaster instructions per second of simulation speed on a 1GHz x86 computer.

The Sandblaster DSP is a multi-threaded processor [6][7]. A programming interface to the resources (multiple threads, peripherals, memories etc.) on the processor is provided via POSIX pthreads [8]. Keeping with the philosophy of having a standard, user friendly and efficient programming model, this interface is based on ANSI C and is commonly used in many operating system environments including Linux and Windows.

A multi-threaded application has a number of advantages over a single threaded application. For example, one thread may be waiting for data from a peripheral while the other can perform some important computation. Thus, the latency of access from the peripheral is hidden.

The first implementation of the Sandblaster processor provides access to a fixed number of threads. These threads are called hardware threads (HT) or contexts. However, an application (written in C) can contain an arbitrarily large number of POSIX threads (called software threads or just threads), which are scheduled onto the hardware threads by the Sandblaster kernel.

3.2. Physical Layer Software

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 systems there may be over 2 million logic gates required to implement physical layer processing. These systems 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 development cycles. An SDR design takes a completely different approach to communications system design.

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

In our implementation of GPS, WLAN, and Bluetooth, each communication protocol has a number of software threads that may execute on hardware thread units. Each software thread implements a portion of a specific physical layer of the communications protocol. As described in this paper, a common input buffer of the sampled data is used for all three communication protocols. Control software executing as a thread on the Sandblaster processor coordinates which RF front end is active and signals each physical layer software implementation as to when data is available for that protocol. The data is multiplexed and read by the correct software communications program under the coordination of the control software.

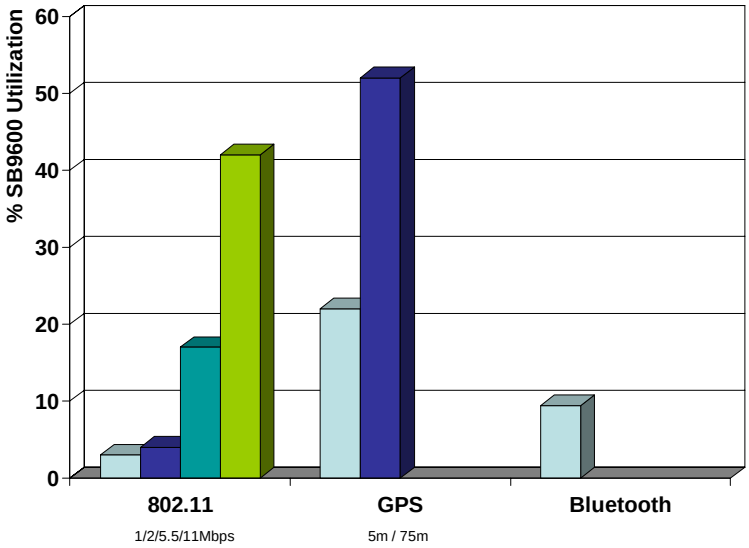


Figure 7. Communication System Performance

4. Results

Figure 7 shows the utilization on the Sandblaster SB9600 platform for each complete communications physical layer implementation. The SB9600 has four integrated Sandblaster cores each with eight thread units supplying 32 hardware thread units available for concurrent

execution of software threads. The physical layer performance requirements of WLAN is shown at 1,2, 5.5 and 11Mbps. The worst case utilization is about 40% of the SB9600. The GPS performance is dependent upon the accuracy desired. For 75 meter accuracy in the x,y, and z axes, about 20% of the SB9600 is utilized. For high performance 5 meter accuracy, slightly over 50% of the SB9600 is utilized. Bluetooth requires significantly less processing capacity and utilizes less than 10% of the SB9600.

5. Conclusions

We have described a communications platform capable of running multiple waveforms concurrently. A key aspect of this system is that all of the waveforms are digitized at the same hardware sampling rate. This decreases the amount of hardware required for integration. The sampled data is then resampled in software prior to further processing. In addition to the RF design, we described both the timing and data sample generation and provided results for the complete baseband processing for GPS, WLAN, and Bluetooth. Using a software implementation of the physical layer, it will be possible to execute these waveforms concurrently on the SB9600 platform.

6. References

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

[2] R. S. Bovard and A.C. Bush, "Multi-Channel SDR Architectures for C4ISR Applications" RF Design Magazine, pp. 50-60, May, 2003

[3] E. Kreyszig "Advanced Engineering Mathematics", John Wiley and Sons, Inc., 1972.

[4] D. Iancu, J. Glossner, E. Hokenek, M. Moudgill, V. Kotlyar, "Software GPS receiver", Accepted for publication is the 2003 Software Defined Radio Technical Conference and Product Exposition November 17-19, 2003 - Orlando, Florida.

[5] J. Glossner, S. Dorward, S. Jinturkar, M. Moudgill, E. Hokenek, M. Schulte, and S. Vassiliadis, "Sandbridge Software Tools", in Proceedings of the 3rd International Worksop on Systems, Architectures, Modeling, and Simulation (SAMOS'03), July 21-23, 2003, pp. 142-147, Samos, Greece.

[6] 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, pp. 70-84, November, 2002.

[7] J. Glossner, E. Hokenek, and M. Moudgill, "Multithreaded Processor for Software Defined Radio", Proceedings of the 2002 Software Defined Radio Technical Conference, Volume I, pp. 195-199, November 11-12, 2002, San Diego, California.

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