SOFTWARE DEFINED RADIO IMPLEMENTATION CONSIDERATIONS AND PRINCIPLES USING THE SANDBLASTER™ SDR BASEBAND PROCESSOR

Babak D. Beheshti, (B.Beheshti@ieee.org) ,
Tanuj Raja (Tanuj.Raja@Sandbridgetech.com), Sandbridge Technologies Inc.

ABSTRACT

The design flow and methodology for hardware centric radio baseband processors is well established and understood. However, migration to a software centric baseband processor approach is still new to many designers. The paradigm shift of viewing real time events from the point of view of gates and registers to pointers and memory offsets is not often easily grasped. This paper covers a new design flow appropriate for a complete software driven baseband processor using the Sandblaster™ SDR baseband processor. Topics covered in this paper include programming for the RF-Baseband Interface, Programming the entire physical layer processing in C, handling TDMA/CDMA time-critical events, Code/Data memory considerations and layout, Multi-program support for multi-mode radios, System Software Pseudo-code, Terminal Reconfiguration Management, and Taking advantage of Low Power Architectural features.

1. INTRODUCTION

Software Defined Radios (SDRs) offer a dynamically reprogrammable method of reusing hardware to implement the physical layer processing of multiple communications systems and applications. SDRs can dynamically change protocols and accept communications systems and applications updates over the air as quickly as a service provider requires this update. Rapid implementation of numerous multimedia applications and multiple wireless communication protocols is easily accomplished on a single programmable platform utilizing an SDR baseband processor. Furthermore, enhancing the terminal capabilities with new protocols, applications, and functions is achieved through over-the-air dynamic software downloads. This capability reduces product feature support cost, time-to-market, and risk while increasing handset OEMs and ODMs R&D and manufacturing productivity.

The basic terminal model assumed in this paper is one with a block diagram shown in **Error! Reference source not found.**. The interface between the RF front end and the baseband processing is a parallel n-bit or a serial digital interface. The sample word size, in bits, is determined based on BER and channel requirements. Extra bits increase the overall dynamic range of the sampled data. It is assumed that the RF front end block includes the down conversion and the A/D and D/A circuitry to provide a purely digital interface to the baseband processor. It is furthermore open to implementation whether this digital interface is DMA driven or directly controlled by the processor. In this discussion, a DMA driven interface is assumed. The Sandbridge baseband processor, SB3010 provides a 16-bit parallel, DMA based I/Q interface for transfer of digital Baseband data to/from the RF front end.

2. DESIGN FLOW

The design flow for a SDR bases baseband processor is somewhat different from the traditional hardware based designs. This design flow relies heavily on host based development throughout the development cycle removing the sequential nature of dependency on the hardware platform availability to make fine-tuning of system performance. The general steps in the development of any air interface waveform using this methodology are listed below:

1. Physical Layer Algorithm Development (floating point) using MatLab, …
2. Translation to fixed point ANSI C
3. Simulate for algorithm accuracy in fixed point/ use debugger to debug code
4. Modify code as necessary
5. Profile code for MIPS/MHz requirements
6. Optimize specific functions as necessary (only in C)
7. High level partition the code into software threads (tasks) using the API provided by the operating system used.
8. Using the simulator supplied with the tool chain, and possibly an event viewer examine latency requirements
9. Repartition threads as necessary - balance system load
10. Final system integration and hardware testing

As can be seen in Figure 1 below the development effort is front loaded, where the cost and risk to system changes are significantly lower than the back end. This significantly reduces risk and development time as well as allowing for tremendous visibility into the implementation because of the host base development environment.

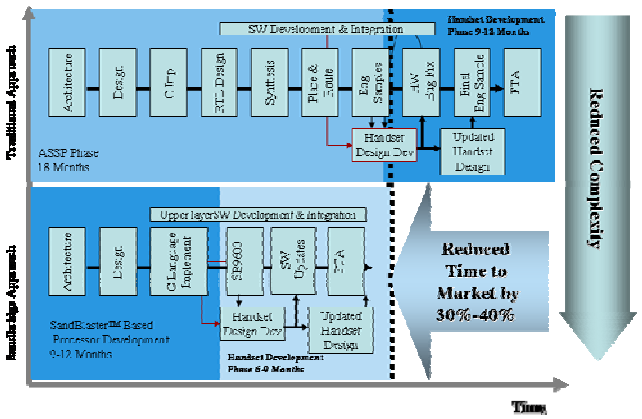

Figure 1 – SDR Based terminal Design Flow

3. PROGRAMMING THE ENTIRE SYSTEM IN C

Figure 2 shows the Sandblaster® tool chain. 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. The tools are driven by a parameterized resource model of the architecture that may be programmatically generated for a variety of implementations and organizations. 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.). The tools are then capable of producing dynamic and static simulators. A binary translator/compiler is invoked on the host simulation platform. 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).

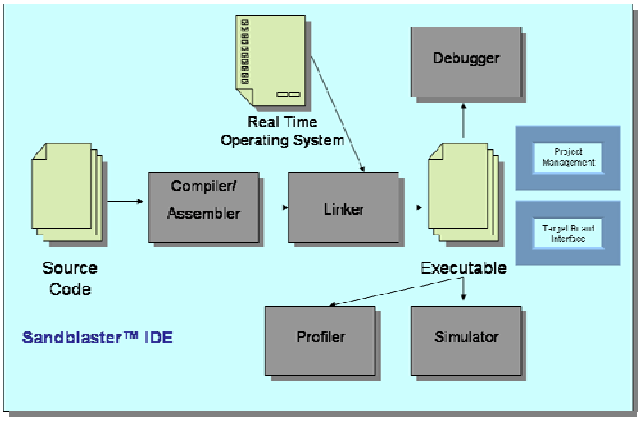

Figure 2 - Sandbridge Software Development Tool Chain

For the dynamically compiled simulator, the object file is translated into x86 assembly code during the start of the simulation. In single-threaded execution, the entire program Dynamically compiled single threaded simulation translation is done at the beginning of the execution phase. Regions of target executable code are created. For each compound instruction in the region, equivalent host executable code is generated. Within each instruction, sophisticated analysis and optimizations are performed to reorder the host instructions to satisfy constraints. When changes of control are present, the code is modified to the proper address. The resulting translated code is then executed.

4. CODE/DATA MEMORY LAYOUT

With the non-volatile memory size of most 2.5G mobile phones in the several megabyte range, the code size for any of the individual waveform physical layers, the constraint is not the code size itself rather the cache hit rate. The cache hit rate indicates how often the code is available in the instruction cache of the baseband processor, eliminating wait states to fetch the instructions from the external flash memory.

As for volatile memory (for scratch pad data memory), the physical layer demands on the data storage needs to be analyzed to assure that the needs of the program do not exceed the handset's available RAM. On the other hand, the RAM requirements are to be kept at a reasonable level not to exceed the budgeted power consumption for the handset. As two examples, the Sony Ericsson P900 GPRS, HSCSD, WAP, Java, MMS, HTML browser, email client, built-in digital camera, touch-sensitive display, polyphonic ringtones, MP3 player, MPEG 4 video, handwriting recognition, Bluetooth, infrared, USB port and many more. The phone is equipped with 48MB of RAM. The Nokia 9300 supports following bearers: CSD, HSCSD, GPRS multislot class 10, class B, EGPRS class 10 Bluetooth, IR (115 kbps). It has 64 MB SD RAM, 128MB Flash. Out of the on-baord RAM, the free user memory is 80MB, indicating amount used by the physical layer and the protocol stack).

5. PROGRAMMING FOR THE RF-BASEBAND INTERFACE

There is a significant paradigm shift in thinking of and implementing the real-time data flow in a software based transceiver. The offsets in time are to be thought of as offsets in memory buffers. Activation of specific events at specific time instances are then computed in software and then implemented using on-board timers. For example in a TDMA based system, the time slots received in the receive buffers are collected. Subsequently "cell-search" is performed on the collected data, determining the start position of the time slots and the transmission frame. Now the indicator for the start of frame is simply a memory pointer. Since the control software is aware of the correspondence of the actual time of arrival of the data and its location in the receive buffer, an offset into the buffer can simply be converted to a calculated offset in time. This offset in time is used to turn on an on-board timer. This timer is programmed to initiate transmit of a frame to the base station at the precise instant governed by the receive frame start time. As software adjusts timing during operation, the transmit time will also be adjusted to maintain the critical strict timing relation between receiving and transmitting. This idea is further developed later in this paper.

6. MULTI-PROGRAM SUPPORT

The Sandbridge software tools have the ability to combine multiple executables together, without going through the difficult task of manual name mangling of conflicting names in the two executable files. The user has to follow the following steps to combine two executable files into a single executable file.

1. Compile all the object files for a program (say 'prog1') and gather them into an archive (say 'prog1.a'}
2. This will put all the prog1.o files in to the archive 'prog1 a', instead of into an sbx file.
3. Repeat the above steps for all programs so that you have archives 'prog1 a', 'prog2.a' etc.
4. Indicate the main functions for these programs. For example, prog1 0, prog2 () etc.
5. Write a program that will invoke these (set of main functions) as and when needed, say driver.c

Link them together.

As a result, the executables for several waveforms can be combined into a single executable which will be resident in the flash memory of the wireless terminal. This convenient arrangement reduces the task of invoking different waveforms to simply "calling a function" associated with that waveform. Therefore even though traditional approaches such as marking a flag in the non-volatile memory and "re-booting" to the new waveform are still possible and available, the process can be streamlined significantly by this technique.

7. SYSTEM SOFTWARE PSEUDO-CODE

Using the convenient multi-program support of the tools, one possible pseudo-code for the main operational sequence of the handset software can be composed as follows. It is noteworthy that in the following sequence, user input via a key press or menu selection would result in the reconfiguration of wireless terminal into a new waveform.

Power on/Reset
For all supported waveforms
    Perform cell search
    Mark waveform as detected
Display all detected waveforms
Perform sky search
Display GPS data
Wait for user input for waveform selection
Do forever
If waveform selected
    Call the routine for the selected waveform physical and protocol stack
If different waveform selected by the user
    Perform cell search
    If detected
        Call the routine for the selected waveform physical and protocol stack
    Else
        Display error

A typical startup of a detected waveform could be as follows:

Select a base station
Activate upper layers/protocol stack
Decode broadcast channel information
Enter idle mode
Monitor paging channel for incoming calls
Monitor keypad for user initiated out-going calls

8. TERMINAL RECONFIGURATION MANAGEMENT

Implementation of the terminal reconfiguration management, control through "privileged over the air command", or local command or event triggered command can all be easily implemented in software as the SandBlaster™ SDR baseband Processor is capable of supporting any control program as well as DSP algorithms. A typical protocol stack configuration is shown below:

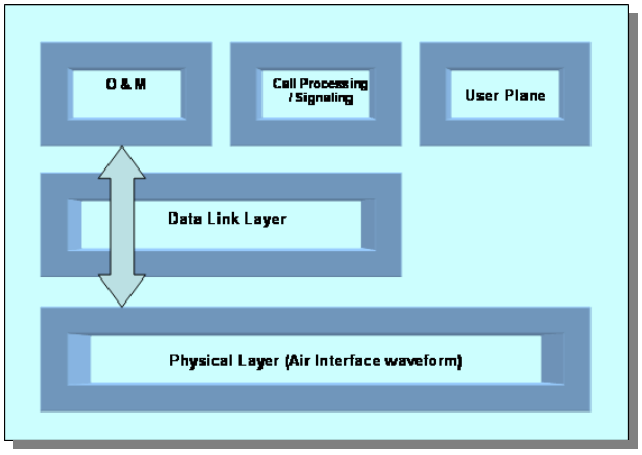

Figure 3 - Typical Protocol Stack with O&M

O & M (Operations and Maintenance) is a protocol specific to any particular implementation whereby specific logical channels carry commands and information for maintenance of the wireless terminal. As seen in figure above, the O&M protocol relies on the layers 2 and 3 that carry the signaling for call processing. Proprietary commands such as the following are typical O&M commands that can be relayed to the wireless terminal over the air:

- Status Request
- Reconfiguration Request
- Firmware Update Request
- Restart Request
- Etc.

9. PROJECTION OF POWER CONSUMPTION FOR A PARTICULAR WAVEFORM

Based on the average power consumption derived for a single core of a multi-core device, such as the Sandbridge SB3010, one can estimate the expected power consumption of any waveform once its MHz (or MIPS) requirements are determined. This calculation is very simple in that the total MHz is converted to the number of DSP cores required to run the waveform in real-time (keeping the fractional part). The number of cores is then multiplied by the nominal figure for the average power consumption for the waveform. The following example illustrates the power consumption calculations for the 384kbps WCDMA physical layer.

Total MHz demand = 1800MHz
1800 / 600 = 3 Cores
Average Power consumption = 3 Cores * N mW/Core

The following considerations are noteworthy in power consumption calculations:

1. The calculated power is average. It can be further lowered by taking into account duty cycle of the running code as per sleep mode and discontinuous reception/transmission specifications of a standard waveform.

2. Many power management techniques can be employed to significantly reduce the projected power number derived above. These techniques include frequency and voltage scaling, operating system based core/thread clock gating etc.

3. The power consumption discussion above pertains to baseband only and therefore excludes RF section power consumption.

12. CONCLUSION

Several techniques and approaches to implement SDR based terminals were discussed in this paper. As can be seen, the techniques are evolving to be significantly different from the "logic gate" approach. The success of a SDR based terminal mostly depends on a powerful SDR processor platform and the software development tools that enable the designers in a timely design/implementation/debug cycle. The Sandblaster™ platform along with its software development environment used as an example in this paper provide such a platform.

13. REFERENCES

[1] B. Beheshti, J. Glossner, D. Routenberg, L. Zannella, and P. Steensma, "Evaluation of Military Waveform Processing on a COTS Reconfigurable SDR Processing Platform", Proceedings of Software Defined Radio Technical Forum, Volume A, pp. 147-151, 16-18 November, 2004, Scottsdale, Arizona.

[2] B. Beheshti, "A Study of the Technology Migration Path of the Cellular Wireless Industry from 3G to 3.5G and Beyond", Proceedings of 2005 IEEE Long Island Systems, Applications and Technology Conference (LISAT2005), May 2005, Farmingdale, New York.http://www.sdrforum.org

[3] D. Iancu, J. Glossner, V. Kotlyar, H. Ye, M. Moudgill, and E. Hokenek, "Software Defined Global Positioning Satellite Receiver", Proceedings of the 2003 Software Defined Radio Technical Conference (SDR'03), HW-2-001, 6 pages, Orlando, Florida, 2003.

[4] J. Glossner, D. lancu, 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.

[5] D. lancu, 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.

[6] 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.p3), July 21-23,2003, pp. 142-147, Samos, Greece.

[7] 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.

[8] 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.