OPTMIZED SOFTWARE DEFINED RADIO

Bryan Chen, Vijay Chandrasekhar

ABSTRACT

A software radio is a radio whose channel modulation waveforms are defined in software. A software radio receiver is an embedded platform that can support different communication standards. All the base-band processing is performed on a DSP. On the receiver side, the base-band processing encompasses matched filtering of the incoming data, frame synchronization, frequency offset and phase offset correction, timing offset correction and finally the demodulation of the digital base-band signal. For the system in consideration, the base-band processing is done on a TI DSP (TMSC6701), a DSP with stringent speed and memory constraints. For the software radio to function in real-time, it is imperative that each of the software components be highly optimized for time and space. This paper focuses on optimizing the three core modules of a software radio: filtering, correlation and frequency offset correction. A variety of optimization techniques such as Karatsuba, rhombing and overlap add using Spiral FFTs were employed to optimize the software components mentioned above. We managed to achieve a speed of 5X by employing the above-mentioned optimizations. We conclude on the basis of our timing results and analysis that a real-time implementation of our software defined setup is achievable.

1. INTRODUCTION

Promising wireless applications can be broken down into three groups: cellular communications, wireless LANS, and defense/aerospace technologies. Each of these employs a different communications standard. All these contemporary communications signal an industry wide desire for Radio Frequency (RF) technology that can be rapidly upgraded, reconfigured, and deployed at minimal cost and retaining use of existing technologies. However, most RF technology in use today requires specially designed hardware to process signals and data at sufficient speeds for effective use. The concept of the software radio is to replace these high-priced components with object-oriented software architectures that not only meet the intense computation requirements of digital communications, but allow dynamic reconfiguration and minimal cost, on-demand upgrades of products.

Once hardware can be reconfigured on-demand, an entire cellular network could be upgraded by distributing a new program, in much the same manner that software patches and updates respond to virus threats. Another potential use of this technology is creating bridges between networks operating on different standards.

1.1 Motivation

Our software radio uses a TMSC6701 TI DSP for base-band signal processing. The TMSC6701 has stringent speed and memory constraints. The peak performance that can be achieved on the TMSC6701 is 900 MFLOPS. Also, the TMSC6701 has only 1 M-bit of on-chip memory.

For our software radio to operate real-time, several real-time timing constraints need to be met. It was determined that the core software components (filtering, correlation, frequency offset correction) needed to be highly optimized, for the system to operate under real-time conditions. As a result of the TMSC6701 system limitations listed above, it is very important that the core software modules of the system be highly optimized so that our system can support high input data rates

1.2 Previous Work and State-of-the-art

The TMSC6701 TI DSP embedded platform has been used by the FEAST [1] software radio project at Western Michigan University. FEAST also highlights the importance of optimizing software modules for the embedded TI platform. FEAST uses C’6000 TI DSP profiling tools to identify inefficient code segments. These time-critical areas are then rewritten using linear assembly and optimized using TI assembly optimizers.

Another popular software radio platform is the open source GNU Radio [2]. The GNU Radio provides a library of signal processing blocks and details on how to link them together. The programmer builds a radio by creating a graph where the vertices are signal processing blocks and the edges represent the data flow between them. The signal processing blocks are implemented in C++. Conceptually, blocks process infinite streams of data flowing from their input ports to their output ports

1.3 Overview

Our software radio receiver set-up consists of a Maxim RF2410 evaluation board, AD6644 Analog to Digital Converter (ADC), AD6620 Digital Down Converter (DDC) and a TMSC6701 TI DSP. All the base-band signal processing is performed on the TI DSP.

We began by implementing the core modules of the SDR on a Pentium 4 3.2 GHz processor. We identified the critical software modules of our software radio implementation i.e. filtering, correlation for frame detection and frequency offset correction. The filtering module was best optimized by implementing rhombing and loop-unrolling optimization techniques on the straightforward convolutional implementation. The correlation module was best optimized by implementing the overlap-add FFT algorithm using smaller 1024 Cooley-Tukey FFTs generated by SPIRAL [3] for our x86 test platform. We determine the frequency offset statistically by searching over a range of possible frequency offset values. This module required a large number of constants to be pre-computed and stored in memory. Hence, this module was optimized extensively for both time and space. We optimized this module using basic optimization techniques like loop-unrolling and scalar replacement. We reduced the number of constants so that they could all be stored on the TMSC6701 on-chip memory.

We used the results that we obtained for our x86 implementation to draw conclusions on whether or not real-time constraints could be met on the TMSC6701 DSP platform. Based on our timing analysis, it was determined that meeting real-time constraints was achievable on the TI DSP embedded platform for our software radio set-up.

1.4 Organization of paper

In Section 2, we provide a detailed description of the hardware components and core software modules of our SDR. In Section 3, we provide details of the embedded DSP platform used in the SDR. Section 4 includes details of our implementation and the optimization techniques that are used. A detailed timing analysis is provided in this section. In Section 5, we provide details of porting the preliminary x86 implementation to the embedded DSP platform. Section 6 concludes with a summary of our results.

2. BACKGROUND OF SOFTWARE DEFINED RADIO

2.1 Hardware Set-up

The Maxim 2410 Evaluation boards are used for receiving and transmitting RF signals. The up conversion from IF to RF and the down conversion from RF to IF is done using Analog Devices evaluation boards AD6620 and AD6623 respectively. Analog Devices evaluation boards AD6644 and AD9723 are used for A/D and D/A respectively. The base-band processing is carried out on a TMSC6701 DSP (Texas Instruments).

Our current goal is to optimize the software components on a x86 platform and then port it to the Texas Instruments DSP platform (TMS C6701). The results obtained by implementing the core software modules on a x86 platform are discussed in Section 4.

2.2 Software components

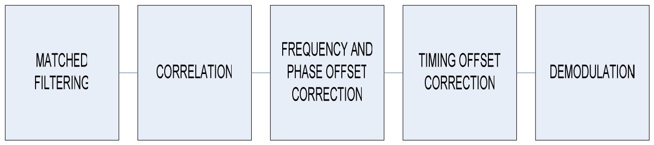

Figure 1: Core modules of a software radio

The key software components of the SDR are illustrated in the diagram above. For our test system, the data is processed in blocks of 7500 complex samples. The maximum block size that can be stored in the on-chip memory of the TI DSP is 8000 complex samples. Also, the test data that we use is 10 times over-sampled.

The input to the above system is 7500 QPSK (Quadrature Phase Shift Keying) complex samples and the final output of is 7500/5*2 = 3000 information bits.

2.2.1 Filter

The input to this block is 7500 complex samples and the output of this block is 7500 complex samples after the filtering process. The incoming data is first filtered to remove noise components. The filter in consideration here is a raised cosine matched filter, which is given by the equation below:

$$Q(f) = \begin{cases} T, & 0 \le |f| \le \frac{1-\alpha}{2T} \\ \frac{T}{2}\left(1-\sin\left(\frac{\pi T(|f|-\frac{1}{2T})}{\alpha}\right)\right), & \frac{1-\alpha}{2T} \le |f| \le \frac{1+\alpha}{2T} \\ 0, & else \end{cases}$$

where T is the sampling period and alpha is the roll-off factor. The Matlab function firrcos is used to generate a 100 tap filter.

2.2.2 Correlation

The input to this block is 7500 complex samples obtained from the filtering block. The output of this block is the starting index of a frame.

The incoming data is received in packets. The start of each frame or packet is determined by correlating the incoming data sequence with a known correlation sequence. The correlation sequence used here is a QPSK sequence consisting of only symbols 00 and 11.

2.2.3 Frequency offset and phase offset correction

The correlation block above is used to determine the starting index of a frame. The header of each frame is a known BPSK (Binary Phase Shift Keying) sequence. The header is used to estimate the frequency offset. The input to this block is 630 samples (header of one frame) and the output of this block is an estimate of the frequency and phase offset of the data.

The down conversion of the RF signal to the base-band domain is not ideal and hence leaves a frequency offset in the base-band signal. This module corrects the offset empirically by trial and error of a range of possible values of frequency offset and phase offset. The set of frequency offset values over which the search is done is a function of the carrier frequency.

Let the BPSK header sequence of length 630 be given by x[n]. The values for w0 and alpha that minimize the standard deviation of the scalar product x[n] and exp(j*w0*n+alpha) are the frequency offset and phase offset respectively. This can be characterized by the equation below:
{alpha, w0 | min(std deviation (x[n].*exp(j*w0*n + alpha for n = 0: 629))}, for a range of values alpha and w0 determined empirically.

2.2.4 Timing offset correction

The input to this block is 7500 complex samples obtained after correction of the phase and frequency offset in the module 2.2.3. The output of this block is 7500/M complex samples where M is the over-sampling factor.

The incoming data is over-sampled by a factor of 5 or 10 for all our test cases. This module corrects the timing offset by sampling the over-sampled data at the right instants in time.

2.2.5 Demodulation

The input to this block is 7500/M complex samples obtained after the correction of the timing offset in module
2.2.4. The output of this block is the final 7500/M*2 information bits in the signal. Our SDR is primarily used for demodulating ASK/PSK data. The modulation scheme used in all the test cases in consideration is QPSK IS-95.

Finally, the timing offset correction and demodulation modules are O(N) run-time operations. These two modules take significantly less time than the first three modules and are hence ignored in the timing analysis in the following sections. Also, equalization for removing multi-path components in the signal is also a core component of a SDR. Since, the test environment is free of multi-paths, the equalization module is also ignored.

3. TI DSP PLATFORM

The TI C6701 DSP is a VLIW processor with up to eight parallel operations occurring on each clock cycle. The TI C701 operates at a clock frequency of 167 MHz. The peak performance that can be achieved on this platform is 900 MFLOPS. A performance of 700 MFLOPS can be sustained on this platform [4].

The TI C6701 has stringent on-chip memory constraints too. This embedded platform has 1 Mbit of on-chip memory, 256 kb of synchronous 133 Mhz SBSRAM and 32 Mb of 100 Mhz synchronous SDRAM [5]. However, access to data in the SBSRAM/SDRAM is significantly slower than accessing on-chip memory. Hence, several optimizations for space have been performed to try to fit all the pre-computed constants in the TI on-chip memory.

4. IMPLEMENTATION AND OPTIMIZATIONS

Since our system consists of modular components, we decided to optimize each component individually since an individual component's performance shouldn't affect another component's performance. The MFLOPS were computed using the pin tool [6].

All of the implementations were compiled and run on a Intel Pentium 4 with 8K L1 cache and 512K L2 cache.

The code was compiled with gcc version 3.2.1 with –O4 flags. The experiment below was carried out on an input data stream of 66 kilo symbols per second. The IF carrier frequency is set to 1 MHz. Processing of data is done in blocks of 7500 complex samples and the size of each frame or packet is 288 bits.

4.1 FIR Filter

The matched filter in consideration is a FIR filter with 100 taps. The raised cosine filter is generated in Matlab using the command firrcoss. The filter taps are constant for a given input data rate. The FIR coefficients were hard-coded into our C program. If implemented by definition, filtering has O(LN) run-time, where L is the number of taps and N is the size of the input vector. On the other hand, if implanted using the Cooley-Tukey FFT, filtering takes O(N log N) run-time. The optimizations performed on the filtering module include rhombing and a one-level Karatsuba implementation.

The fact that the filter length is significantly shorter than the input implies that DFTs should be avoided as they would result in a lot of wasted space.

Algorithm	Time taken (seconds)	MFLOPS
By Definition	0.006	500
Rhombing	0.005	615
Unroll by 4	***0.0025***	***1230***

Figure 2: Runtimes for Filter Module

Figure 2 shows the runtimes of the different implementations. We tried to implement the one- level Karatsuba approach, but it did not have any significant improvement in runtime due to the additional overhead of copying. The rhombing implementation seemed to perform better than the implementation by definition due to cache locality. This implementation is further improved by unrolling the inner for loops in the rhombing implementation by a factor of 4.

4.2 Correlation

The correlation is used to detect the start of each frame. The length of the correlation sequence varies with input parameters like the data rate, over-sampling factor and the packet size. The length of the correlation sequence is determined empirically. The correlation sequence is a BPSK sequence, consisting only of the complex symbols 00 and 11. For the test data sets in consideration, the length of the correlation sequence varies from 630 to 1260. Correlation, by definition, has O(LN) run time, where L is the number of taps and N is the size of the input vector. Like filtering, correlation has O(N log N) run time if implemented using the Cooley-Tukey Algorithm. Since the length of the correlation sequence is significantly large, implementation using FFTs speeds up the run-time considerably. The overlap-add FFT algorithm is used to improve the run-time. The overlap-add algorithm breaks down the correlation module into smaller Cooley-Tukey FFTs.

This module was first implemented using the straightforward convolutional definition. We then tried a variety of FFT implementations including FFTs using SPRIAL [3]. Next we finally exploited cache locality using the overlap-add FFT method along with further optimizations such as loop unrolling and scalar replacement.

Algorithm	Time taken (seconds)	MFLOPS
Normal definition, convolution	0.0074	1170
Exact size Stanford DFT (8129)	0.03840	963
8192 DFT Spiral FFT	0.00200	872
Overlap add using 1024 Spiral FFT	0.00500	755
Overlap add unrolled	***0.002245***	***1172***

Figure 4: Runtimes for Correlation Module

The runtimes for the implementations did get faster as we used FFTs. Spiral generated FFTs proved to be faster than the Stanford FFT implementation [7]. We got further improvements in runtimes when we implemented overlap add along with unrolling. One thing to note was an exact FFT size of 8129 caused performance to suffer a lot.

4.3 Frequency Offset Correction

The frequency offset can be defined as the difference between the frequency of a source and a reference frequency. In a software defined radio, the data is down-converted from RF to base-band and then processed on a DSP. However, the down conversion process leaves a residue frequency offset, which needs to be corrected before the data can be demodulated correctly. In our system, the frequency offset and phase offset are determined empirically. For our software defined radio, the order of the frequency offset was first determined empirically as a function of the input carrier frequency.

The frequency offset algorithm described in 2.2.3 takes O(LMN) run-time where L is the number of trial frequency offset values, M is the number of phase offset values and N the length of the header sequence.

For the frequency offset correction module, we first implemented a straightforward brute force search over a large range of values. Next, the search was done using fewer constants and some math simplifications. We next implemented the frequency offset correction algorithm using a smarter search determined empirically on the basis of the input carrier frequency. This implementation searches over fewer values and yields the same results as the previous implementations. For our software radio system, it was determined empirically that the frequency offset is of the order 10^-10 of the carrier frequency. This knowledge is used to search over a narrower range of values to determine an estimate of the carrier frequency offset. Our last optimization uses a technique which decreases the number of constants that need to be stored from N to square root of N.

Algorithm	Time taken (seconds)	MFLOPS	Constants
Brute force search over 200 frequency values	0.04	1891	126050
Unroll by 8	0.03	2521	126050
Smarter search based on input frequency	0.007	2161	25250
Reduce Constants	***0.014***	***1607***	***2050***

Figure 4: Runtimes for Frequency Offset Module

The runtimes got faster as we implemented algorithms which used less floating point calculations. Also it can be noted that unrolling helped to speed up runtime. The biggest performance gain though was achieved from a smarter search which searched over a narrower range of values. One also notices, that the run time increases by a factor of 2 when the number of constants is reduced from N to square root of N. This happens because of a trade-off between storing one constant in memory versus computing one during run-time. Due to the memory constraints of the DSP, the last implementation was chosen as the ideal one.

5. MOVING TO THE TI PLATFORM

Our total best runtime on our x86 platform took 0.0025 + 0.002245 + 0.014 = 0.018745 seconds. The DSP has a maximum flop rate of 900 MFLOPS [1] which is roughly a factor of about 3 slower than our maximum x86 MFLOPS. Interpolating this, we predict that the runtime on the DSP should take about 0.056 seconds. The real-time constraints that need to be met on the TI DSP based on some back of the envelope calculations are shown below:

Data rate/ksps	**Upper bound on time to process one block of 7500 complex samples**
66	0.114
100	0.075
250	0.03

Based on these back of the envelope calculations, it seems that our current implementation on the DSP would be able to support a constant input data stream of 100 kilo samples per second.

6. CONCLUSION

.

Our software radio's 3 main components were each optimized a significant amount and all had significant runtime improvements from the initial implementation. Using different algorithmic implementations, along with optimizing for cache locality and loop unrolling played an important role in the optimization process. In an effort to fit as many pre-computed constants into the TI DSP on-chip memory as possible, we implemented optimizations that reduced the number of constants that needed to be stored from the N roots of unity to square root of N roots of unity. Our next step would be to port our code to the TMSC6701 DSP. This will also involve relocating many of the constants we use to the different memory banks on the DSP.

7. REFERENCES

[1].Bazuin, Bradley. *Flexible Electrical and SoftwareProgrammable Transceivers (FEAST) for Wireless Communications*
[2].GNU Radio: http://www.gnu.org/software/gnuradio/
[3]. Spiral Code Generator: http://www.spiral.net
[4].TMS320c67x Floating Point DSP Performance: http://www.techonline.com/community/ed_resource/feature_article/20124
[5]. TMS320C6201/6701 Evaluation Module: User's Guide
[6].Pin tool: http://rogue.colorado.edu/Pin/index.html
[7]. R.C. Singleton, Stanford Research Institute