

A NOISE ADAPTIVE SYMBOL TIMING SYNCHRONIZATION ALGORITHM FOR SOFTWARE RADIO RECEIVERS

Declan Flood, Linda Doyle, Philip Mackenzie, Keith Nolan, Donal O'Mahony
Centre for Telecommunications Value-Chain-Driven Research (CTVR),
Trinity College Dublin, Ireland.

ABSTRACT

Software radio promises tremendous benefits from both commercial and military standpoints. Currently, there are a number of different 'shades' of software radio based on the use of FPGAs, DSP chips or general-purpose processors to perform the signal processing. There are also hybrids between these three platforms. The Network and Telecommunications Research Group (NTRG) approach is to focus on the use of a general-purpose processor (GPP). This maximizes the reconfigurability of the system. The use of a GPP to perform signal processing for communications applications presents the developer with challenges but it also presents some opportunities. We argue new classes of algorithms are required which will exploit the advantages and negate the disadvantages of using a GPP. Indeed other researchers have already started this programme of 'algorithmic advances'. This paper discusses the issues involved and reviews some existing developments. We present our own progress in developing a noise adaptive symbol synchroniser.

1. INTRODUCTION

In a software radio receiver the analog-to-digital converter (ADC) is placed as close to the aerial as possible, Figure 1 [Tut99]. In a pure software radio it would connect directly to the aerial. However despite recent improvements, the restrictive sampling rate of the ADC make this impossible and so a generic radio frequency front-end is placed between the aerial and converters. The front-end bandpass filters the signal of interest from the received signal and downconverts it to an intermediate frequency the ADC is capable of sampling. Finally, a signal-processing unit performs the remaining radio functions on the digitized signal.

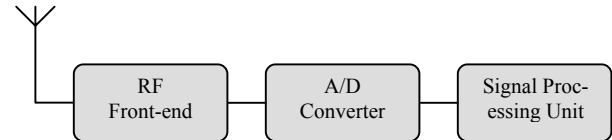


Figure 1: A software radio receiver

Previous work in software radio has concentrated on using reconfigurable hardware such as a FPGA or DSP chip as the processing unit. However there are alternative hardware possibilities and the Network and Telecommunications Research Group (NTRG) approach focuses on the use of high-level languages running on a General Purpose Processor (GPP) such as a Pentium IV. The use a GPP has enormous potential since it maximises the adaptability and reconfigurability of the system [Mac01].

It is important to note that recent developments have further complicated the divide between ASIC, FPGA, DSP and GPP. Hybrid products from companies such as Chameleon Systems Inc. and MorphICs Technology have blurred the distinction between the various platforms.

Software radio has both commercial and military applications. Commercially, software radio promises a range of improvements such as increased spectrum efficiency through spectrum rental, faster development times for telecommunications equipment, Over The Air Reconfiguration (OTAR) for roaming users and sophisticated DSP techniques such as adaptive antennas. Software radio offers the military all these benefits plus the holy grail of interoperability between military branches without susceptibility to jamming [Mel02].

Military users have not been oblivious to the potential of software radio. Indeed one of the original software radio projects was SPEAKeasy [Lac95]. This project developed a single portable radio capable of emulating over 10 existing military radios operating anywhere from 2MHz to 2GHz.

Traditional communications algorithms have been developed with a view to implementation with analog hardware or on an ASIC, FPGA or possibly DSP chip. Algorithms have been tuned for optimum performance on these platforms. The characteristics of the resources available in a GPP innately differ to those of analog hardware, ASICs, DSP chips or FPGAs.

Section two discusses the salient characteristics of the GPP when used in signal processing applications and demonstrates the need for algorithm development. Section three reviews work undertaken by other researchers and section four presents our own developments.

2. GENERAL PURPOSE PROCESSOR FEATURES

The use of GPPs represents a paradigm shift for signal processing. The salient features of a GPP relevant to signal processing are:

- *Large Cache.* Most GPPs have a large cache. This means algorithms that exhibit a high locality of reference (both spatially and temporally) for memory accesses will have a lower average execution time. It is important to note that although a cache reduces the average execution time it can increase the variability of the execution time. This can have a significant effect for algorithms running in real time.
- *Large Data Memory.* GPPs generally have large amounts of RAM. This RAM can be exploited to implement a lookup table for calculating awkward mathematical functions. Less obviously, in a GPP segments of the received waveform can be easily stored in memory. This is in contrast to a purely hardware implementation where samples must be dealt with as they arrive from the ADC. Even in a DSP or FPGA implementation there are only small amounts of RAM available limiting the amount of the signal that may be stored in memory. In GPP implementations there is little penalty for algorithms that make multiple passes over the received signal or which do not process the received samples in the same order they were received.
- *Lower Cost of Copying Data.* Relative to a FPGA / ASIC there is little penalty for making multiple copies of a piece of data. Consider the statement $A = B = C$. The contents of register C must be copied to registers B and A. In a FPGA / ASIC each register will be represented by an array of flipflops. Each of the flipflops of register C must

be connected to the corresponding register of both B and A. Physically, this means there are a large number of connections going to the register C. It is possible that the interconnect resources of the FPGA will not be capable of dealing with this and that the design will fail the Place & Route stage.

- *Ability to Dynamically Choose Algorithm at Runtime.* One positive feature of a GPP is the ability to install a number of algorithms and to dynamically select which algorithm is actually used at runtime. It is also possible to modify or even halt execution midway through processing.
- *Sequential operation.* GPPs execute programs in a purely sequential fashion. Therefore algorithms designed for FPGA / ASICs that have been used in the past because of their parallelizable nature are of no benefit in GPP implementations.

These points illustrate that the resources available when implementing radio functions on a GPP innately differ to those of other platforms. The advent of software radio on a GPP platform requires the development of new classes of communication algorithms. These will exploit the strengths and diminish the weaknesses of GPP platforms.

3. ALGORITHMIC ADVANCES

This section discusses existing work in this field but before doing so it is worth considering how a 'good' signal-processing algorithm running on a GPP should behave. It should have a fast execution time, be capable of adapting to its environment and if it fails it should fail gracefully.

- *Fast execution time.* Algorithms should be a minimal burden on scarce computing resources.
- *Adapt to environment.* They should respond to changes in the environment such as signal-to-noise ratio (SNR) and processor architecture. In high SNR environments algorithms should throttle back the amount of CPU time they are using. Other applications running such as web browsers etc may use the saved cycles. Alternatively the saved cycles may be used to reduce the power consumption by lowering the clock frequency. An algorithm should adapt itself to fully exploit the features of the current PC workstation it is running on. PC workstations vary in processor architecture and cache size and layout.
- *Fail Gracefully.* Where algorithms cannot complete processing due to real time constraints they should at least degrade gracefully. For example when demodulating a bit of data the algorithm

could perform at least some processing and make an estimate at the state of the bit.

This process to develop algorithms that behave as desired but with the constraints of the GPP as described in the previous section has already started. Some of the key developments are discussed here.

Bose describes the implementation of an alternative matched filter algorithm that uses early termination to improve average execution time. Traditional implementations of a matched filter must process all the input samples before terminating. The alternative matched filter algorithm described by Bose processes the input samples in such a manner that in most cases the algorithm can terminate early. In some cases the algorithm can have an extremely long execution time but the average is reduced [Bos99].

Welborn discusses a technique relevant to software radio transmitters called Direct Waveform Synthesis (DWS). Conventional digital modulators consist of a bits-to-symbol mapper, a pulse shaping filter and an intermediate frequency mixer. DWS allows these stages to be performed in one very short step. DWS exploits the memory available on a GPP to create a very large look-up table [Wel99a] [Wel99b].

Frigo details a highly efficient implementation of the Fast Fourier Transform (FFT) called the Fastest Fourier Transform in the West (FFTW). The Fourier transform is essential for many modern communications schemes such as Orthogonal Frequency Division Multiplexing (OFDM). Traditional FFT algorithms are designed to minimize the number of multiply operations required; however on a GPP it is also important to exploit the presence of the cache. The FFTW runs tests at power up to determine the size of the cache and divides the Fourier transform into a series of smaller problems – each of which fits in the cache. The FFTW is an example of an algorithm adapting to its environment [Fri98].

Traditionally, channel separation is performed in the order: mixing, filtering and decimation. The output sample rate is much smaller than the input sample rate. Welborn argues for a novel channel separation scheme where channel separation is performed in the order: filtering, decimation and mixing. The mixing stage has been moved to after the decimation stage and will be performed at the lower and less computationally demanding output sample rate [Wel99c].

4. HYBRID SYMBOL SYNCHRONIZATION ALGORITHM

Our work on algorithmic advances aims to develop a symbol timing synchronization algorithm tailored for a GPP. Symbol timing synchronization is the process of determining the start and end of each received symbol. Symbol synchronization in sampled data receivers is a two-stage process requiring a timing error estimator and an interpolator to correct the received data. Timing synchronization must be performed accurately to ensure the detector stage works correctly [Men97].

Rice presents symbol synchronization techniques suitable for any sampled data receiver [Ric02a][Ric02b]. These techniques, such as the maximum-likelihood or early-late gate methods used in conjunction with a polyphase filter bank to perform interpolation, are suitable for DSP chips. However, none of these existing solutions exploit the GPP's ability to dynamically select at runtime which algorithm is used.

Our novel algorithm is a hybrid utilizing two standard synchronization algorithms. The first, 'light', algorithm is very fast but the quality of its output is low and can lead to a high Bit Error Rate (BER). The second, 'heavy', algorithm has a relatively long execution time but a very low BER.

The 'light' algorithm consists of an early-late gate symbol synchronizer in conjunction with a polyphase filterbank that has a relatively small number of taps. The 'heavy' algorithm consists of a maximum-likelihood timing estimator in conjunction with a polyphase filterbank that has large number of taps.

Our hybrid algorithm performs synchronization on a received block of data using the 'light' algorithm and develops a confidence in its output. If this confidence is high then the algorithm is finished. If the confidence is low, then synchronization is performed again using the second 'heavy' algorithm as shown in Figure 2.

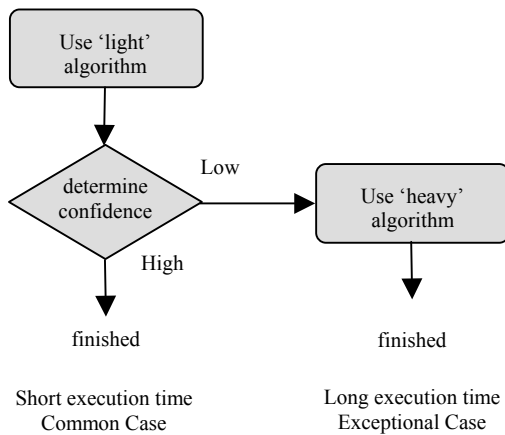


Figure 2: Hybrid Algorithm

Most blocks of data will only require the first ‘light’ algorithm. Some will require the use of both the ‘light’ and ‘heavy’ algorithm leading to a long computational time for these blocks. The overall effect is a shortened average execution time at little cost in quality of the output. The algorithm is noise adaptive; at high SNR the hybrid algorithm will rarely require the ‘heavy’ algorithm and will have shorter execution time than at lower SNR.

A key problem is to find a fast way of determining the confidence in the output of the ‘light’ algorithm. The principle of using a ‘light’ and ‘heavy’ algorithm in a hybrid mix is discussed in [Wel00].

5. CONCLUSIONS

The use of the GPP for signal processing in a communications platform is a major technical challenge. However, its use allows the creation of radios reaching levels of adaptability and reconfigurability unattainable through the use of other platforms.

This paper has discussed some of the salient features of the GPP relevant to software radio. The characteristics of the resources available for GPP implementations innately differ to those of existing implementations. We argue the advent of the software radio on a GPP solicits the development of new digital signal processing algorithms. These will exploit the strengths and diminish the weaknesses of GPP platforms. It is not enough to implement software versions of existing hardware or DSP chip algorithms – fresh thinking is required. This development process has started but further work required to extend this to other physical layer functions in a transceiver.

REFERENCES

- [Bos99] Vanu Bose. “*Design and implementation of software radio using a general purpose processor*”. PhD Thesis June 1999, Massachusetts Institute of Technology.
- [Fri98] M. Frigo and S. Johnson. “*FFTW: An Adaptive Software Architecture for the FFT*”. 1998 ICASSP conference.
- [Lac95] Raymond J.Lackey and Donald W. Upmal. “*Speakeasy: The Military Software Radio*”. IEEE Communications Magazine, May 1995.
- [Mac01] Mackenzie, P., Doyle, L., O’Mahony, D.& Nolan, K., “*Software Radio on General-Purpose Processors*”, IEI/IEE Symposium on Telecommunications Systems Research, Dublin, November 2001.
- [Mel02] Jason Melby, “*JTRS and the evolution toward software-defined radio*”, MILCOM 2002.
- [Men97] Umberto Mengali and Aldo D’Andrea, “*Synchronization Techniques for Digital Receivers*”. Plenum Pub Corp, 1997.
- [Ric02a] Michael Rice. “*Polyphase Filterbanks for Symbol Timing Synchronization in Sampled Data Receivers*”. MILCOM 2002.
- [Ric02b] Michael Rice. “*Loop Control Architectures for Symbol Timing Synchronization in Sampled Data Receivers*”. MILCOM 2002.
- [Tut99] Walter Tuttlebee, “*Software-defined radio: facets of a developing technology*”, IEEE Personal Communications, April 1999.
- [Wel99a] Matthew Welborn, John Ankcorn. “*Waveform Synthesis for transmission of complex waveforms*”. RAWCON’1999.
- [Wel99b] Matthew Welborn. “*Direct Waveform Synthesis for software radios*”. WCNC’1999.
- [Wel99c] Matthew Welborn. “*Narrowband Channel Extraction for Wideband Receivers*”. ICASSP’1999.
- [Wel00] Matthew Welborn. “*Flexible Signal Processing Algorithms for Wireless Communications*”. PhD Thesis. Massachusetts Institute of Technology 2000.