SELECTING APPROPRIATE HARDWARE FOR SOFTWARE RADIO SYSTEMS

Philip Mackenzie (Networks and Telecommunications Research Group (NTRG): Trinity College, Dublin, Ireland; mackenp@tcd.ie); Linda Doyle (NTRG; Linda.Doyle@tcd.ie); Keith Nolan (NTRG; nolanke@tcd.ie); Donal O'Mahony (NTRG; Donal.OMahony@cs.tcd.ie)

ABSTRACT

Software radio is an emerging technology that offers great opportunities for the development of future wireless systems. Many different views exist on how to build both the software and hardware required by a software radio system. This paper focuses on the use of general-purpose processors for developing software radio hardware. A software radio testbed is used to demonstrate the advantages but also limitations of using a general-purpose processor. To overcome these limitations, a particular hardware approach is presented.

1. INTRODUCTION

Wireless technology is playing an increasingly important role in global communications. Coupled with this change, software radio has emerged as an enabling technology for the implementation of existing and future wireless systems. Ultimately, software radio promises to realize wireless devices that are flexible, upgradeable and future-proof.

Software radio is a technology that draws on many different disciplines; mainly software, hardware, digital signal processing and RF design. Among these disciplines there is a general consensus that software radio will be an important and enabling technology for wireless communication. In contrast though differing views are emerging as to how these systems should be implemented. The main conflicting views are between purely hardware and purely software approaches. Many existing hardware manufacturers assume that software radio technology will consist entirely of custom hardware (ASIC, FPGA or DSP) designs. The belief is that a purely hardware approach will provide the best performance, reliability and power consumption. On the other hand, some software-based approaches are proposing the use of general-purpose processors (GPPs) and operating systems. This approach claims to offer the most flexible designs while allowing rapid system development and code reuse as commonplace software development tools can be used.

Currently, neither a purely hardware or software approach can handle all the requirements of software radio systems, in particular the requirements proposed by standards such as the US Department of Defense SCA. Purely hardware designs cannot implement some high-level software abstractions required by software architectures, e.g. CORBA. Likewise, GPP systems and operating systems cannot yet deal with the data rates and latency requirements of existing wireless schemes. This raises the question as to what hardware to use when implementing a software radio. In this paper we address this issue by analyzing the high level design typically used to realize a software radio. We present a working software radio test bed that demonstrates the advantages but also limitations of using a GPP. These limitations lead us to propose a particular hardware approach that maintains the flexibility of a GPP design.

Section 2 presents an overview of the common approach used in software radio implementations and presents a software radio test bed capable of implementing these designs. Section 3 discusses both the advantages and limitations of using a general-purpose processor as a software radio platform. Section 4 discusses various hardware designs. Section 5 presents conclusions.

2. A SOFTWARE RADIO TESTBED

A high level view of a typical software radio implementation is shown in Figure 1. Most designs are based around a common approach, [1]. In reception, the incoming IF signal is digitized, down converted to baseband where the remainder of signal processing is performed by signal processing algorithms implemented

as specially designed hardware or software. Likewise, a signal to be transmitted is modulated via signal processing,

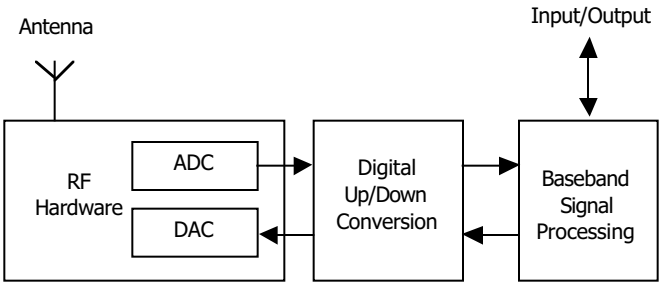

Figure 1 - A Typical Software Radio Implementation

digitally up converted to an intermediate frequency and converted into an analogue signal for transmission.

Our software radio test bed allows the rapid development and testing of such radios by breaking down a typical software radio implementation, into components. The system runs on Windows 2000 and consists of two major elements, a minimal hardware front-end to digitize radio signals and a software radio engine (SWE). The core of the test bed is built around the SWE, a component based implementation that allows the user to create different software radio implementations easily. It has at its core the ability to abstract software radio implementations from underlying hardware capabilities and has been designed to allow for maximum reconfigurability. It comprises of three modules as depicted in Figure 2, a repository of signal processing components, an XML interface and a software radio runtime environment.

The *component repository* is a collection of components that can be used to create a software radio. The SWE allows the creation of different software radios by connecting together different signal processing functions to form the specific wireless scheme of interest. All signal processing functions are implemented as components that conform to a specific interface defined by the SWE. In addition, each component (e.g. channel extractor, FIR filter) exposes a set of properties that can be used to configure the component for a particular scenario. Within the component structure, encapsulation can be used to create new components from groups of existing ones. Additional components can also be added to the repository dynamically at runtime thus offering an even great degree of flexibility.

The *XML interface* is responsible for all interaction with the software radio interface. It performs two main functions. Firstly the XML interface provides a mechanism for users to describe different wireless schemes

using Software Radio Engine ...put to the s... ...oftware radio ... ...onsible for pr...

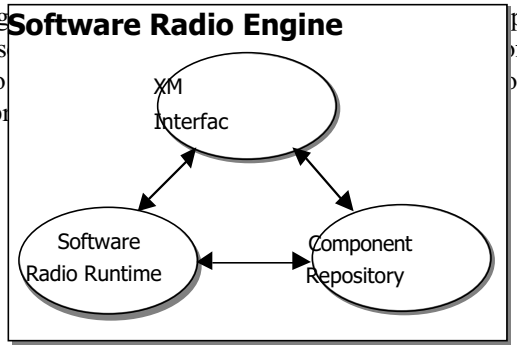

Figure 2 - Software Radio Engine

processing components according to the structure defined in the XML document. Secondly, the XML interface produces XML describing the processing capabilities and components of a particular software radio engine. This facilitates a type of service discovery whereby the capabilities of a SWE can be determined and evaluated. As different SWE's will have different hardware, processing capabilities and sets of components, different SWEs will be capable of running different wireless schemes.

The *software radio runtime* environment is responsible for the execution and control of a particular radio configuration (as defined in the XML document). It controls the flow of data and signals through the software radio and as such provides services to all signal processing components. It can use hardware specific knowledge to optimize the performance of a SWE. Abstracting the runtime as a separate module allows the runtime to monitor the execution of the software radio implementation and thus gather information on performance and component interaction.

Using the SWE environment provides a platform independent way of describing different radios. Once a specific part of a wireless scheme has been implemented as a SWE component, it can interact with all other components in the engine. SWE components can communicate through the software radio runtime and encapsulation can be used to create more complex components from sets of fundamental signal processing components. More information about the SWE environment can be found in [2][3]. This opens up many possibilities for developing novel wireless designs with dynamic capabilities not possible with traditional RF design . It follows that it is greatly advantageous to have

as many aspects of the system as possible implemented as components and thus running on a general-purpose processor. The general-purpose processor has some limitations though, which are discussed in the next section.

3. HARDWARE LIMITATIONS

When considering the flexibility of a software radio design, a constant tradeoff exists between flexibility and hardware capabilities. Ultimately the hardware choice depends on the requirements of the end application. Sometimes, a specific application limits the choice, for example a power conscious wireless device may require the use of a low power DSP. In this paper we are focusing on achieving the highest level of flexibility capable with current hardware technology.

The general-purpose processor (GPP) approach is an excellent choice when complete control and flexibility are required in a software radio design. The operating system provides abstractions such as virtual memory and multithreading which make the development of software radio systems much easier. Development environments are readily available and a GPP system can be programmed using familiar languges such as C/C++. This has a great advantage over DSP/ASIC based systems that often require specialist knowledge to fully manipulate. The drawback of using an operating system is that it is increasingly difficult to guarantee the execution time of code. This is caused by the multithreaded nature of operating system kernels that work by sharing out slices of CPU time between different threads. Operating systems such as Linux and Windows cannot guarantee that threads will always maintain execution as required by the software radio implementation. For example, delayed execution of code could result in timing glitches for digital systems or interrupted audio for analogue applications. There is the option of using a real-time operating system but this can often require specialist programming tools and knowledge which contradicts the advantage of being able to use standard programming tools and maintaining cross platform code. Another limiting factor of the GPP is processing power and more importantly the typical board architectures used by PCs. Although processor speeds greater than 2GHz are commonplace today, the limiting factor in these implementations is the speed of the system bus, PCI bus and memory.

Figure 3 shows the advantages and disadvantages of moving towards a GPP design. To summarize, a GPP is a great advantage if the application meets the following requirements:

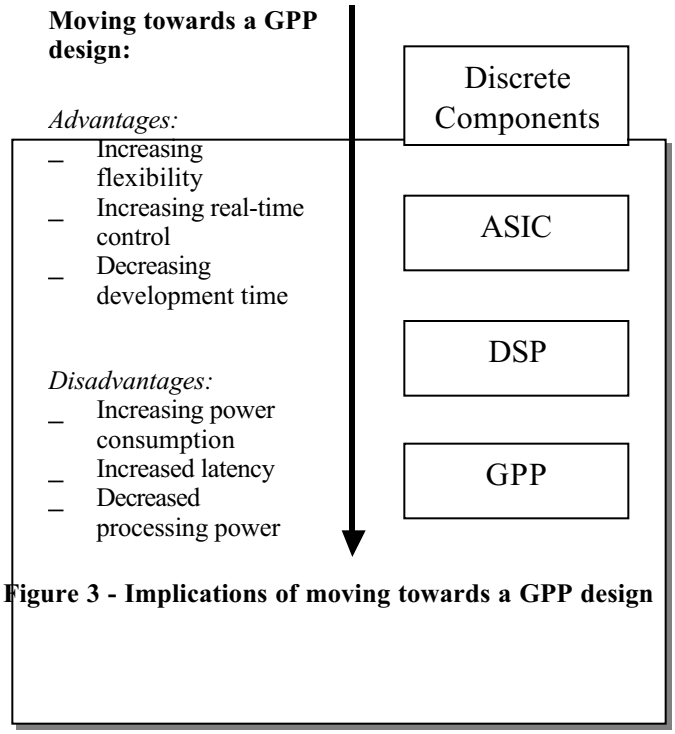

Figure 3 - Implications of moving towards a GPP design

- An efficient method for transferring data between the CPU/memory and radio front-end must exist. Facilities such as DMA should be used to reduce the amount of CPU time used in transferring data to/from the radio front-end as this processing power will be required for digital signal processing.
- The wireless schemes being implemented must be able to cope with glitches occurring due to multithreaded operation. This may not be an issue for analogue schemes such as voice where glitches or delays on the order of milliseconds can possibly go unnoticed.
- A compatible operating system and development tools are available

4. HARDWARE ARCHITECTURES

The key issue that differentiates software radio hardware from conventional designs is that their wireless interface cannot be re-programmed once a device has been manufactured and deployed. To illustrate, figure 4 shows the typical functions provided by a GSM chipset. Chipsets such as these offer programmable logic and some programmable DSP options but these are only useful at the design stage. What is required is a binding between the lower level wireless interface and high-level software abstraction. Figure 5 shows the major hardware components involved in such a system. In most of these cases including the GSM chipset, a custom designed

controller runs the radio interface and has limited configurable abilities.

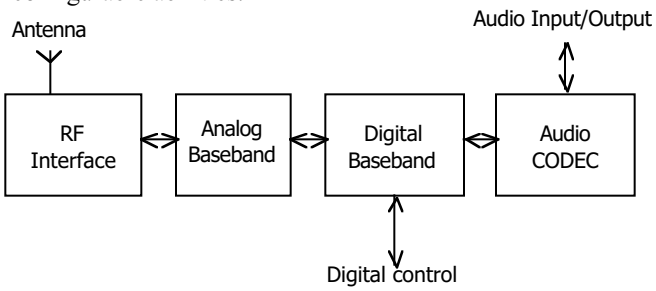

Figure 4 - Functions provided by a typical GSM chipset

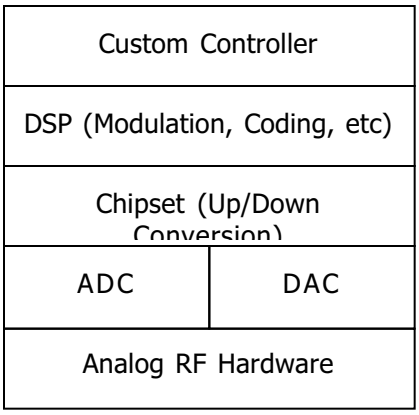

Figure 5 - Typical Implementation

A better approach would be to use a general-purpose processor such as the x86 to perform all digital signal processing tasks (see figure 6). In conjunction with an operating system, the wireless interface would become much more flexible. In such a system new wireless standards are realized by running different software programs on the GPP CPU. Due to the limitations presented in section 3, designs like this are not readily realizable with current GPP technology. A better approach is shown in figure 7 where a balance is struck between using a programmable DSP and a GPP. In this hardware architecture the GPP performs monitoring and control of the interface and also some baseband signal processing at low data rates. Larger signal processing tasks are handled by a programmable DSP, thus reducing the amount of data being transferred in and out of the PCI bus. In this scenario the GPP can reprogram the DSP in real-time thus offloading processing, while still maintaining control over the wireless device. This coupled with a component based engine as shown in section 2 leads to a more future proof design that has all the advantages of code reuse, rapid development, etc.

6. CONCLUSION

In this paper we have shown both the advantages and limitations of using a general-purpose processor as the core of a software radio implementation. We looked at various high-level approaches to building these systems and suggested that with current hardware capabilities a hybrid approach involving a GPP and reprogrammable DSP offers the best compromise between performance and flexibility.

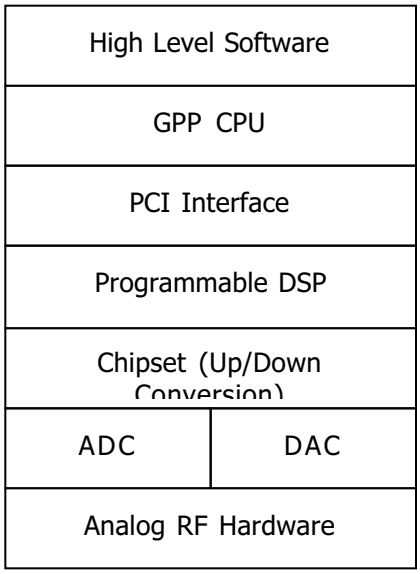

Figure 7 - Compromise

[1] J.H. Reed, *Software Radio, A Modern Approach to Radio Engineering*, Prentice Hall, 2002

[2] L. Doyle, P. Mackenzie, D. O'Mahony, K. Nolan, D. Flood, "A General Purpose Processor Component Based Software Radio Engine", Proceedings of the Second European Colloquium on Reconfigurable Radio, June 2002

[3] Nolan,K.E., Doyle,L., O'Mahony,D. and Mackenzie, P., 'Signal Space based Adaptive Modulation for Software Radio', in Proceedings of the IEEE Wireless Communications and Networking Conference

[4] Mackenzie, P., Doyle, L., O'Mahony, D. & Nolan, K., 'Software Radio on General Purpose Processors", in *Proceedings of the First Joint IEI/IEE Symposium on Telecommunications Systems*, Nov. 01