

The Impact of Software Radio on Wireless Networking

Vanu G. Bose*

*Laboratory for Computer Science
Massachusetts Institute of Technology*

Abstract

A software radio is a wireless communications device in which some or all of the physical layer functions are implemented in software. The flexibility provided by the software implementation enables a single device to interoperate with other devices using different wireless physical layer technologies, by simply invoking the appropriate software. A mobile computing device equipped with a software radio would have access to a wide range of connectivity options including cellular, wireless LAN and satellite systems. This would not only enable seamless anytime, anywhere connectivity, but also provide users the flexibility of choosing from the available connectivity options to best suit their price/performance requirements.

Most software radio research to date has been driven by military and commercial cellular applications. Mobile networking applications require additional functionality present new software radio design constraints. This paper reviews existing software radio research, describes the SpectrumWare software radio system and identifies some important research challenges that must be addressed in order to apply software radio research to mobile networking applications.

1 Introduction

Mobile networking today is hindered by a lack of interoperability at the physical layer. There are a wide range of wireless connectivity options available, including cellular, satellite, wireless MANs and a multitude of wireless LAN systems operating in several different RF bands, but different physical devices are required to interoperate with each system. To truly achieve anytime, anywhere connectivity, it would be necessary to carry a different wireless modem for each system that might be needed. Software radio technology has the potential to solve this problem by providing a single physical device that can be reprogrammed to interoperate with systems utilizing a variety of physical layer technologies.

To date, most software radio research has been driven by commercial cellular and military applications. The multitude of different cellular standards inhibits universal roam-

ing. In addition, new standards are deployed slowly, since it requires installing new basestation hardware and distributing new handsets to users. Similarly, the varying operational requirements of the different branches of the military require different radios, which hinders the coordination of joint operations. While the motivation behind software radio development has been to solve interoperability problems, a fully-programmable software radio can provide better spectrum utilization by dynamically adapting the physical layer to best meet the current environmental conditions, traffic constraints and user requirements. With the appropriate mechanisms for communication between the network and physical layers, wireless networks can take advantage of this flexibility and optimize the physical layer to provide the best possible service.

The SpectrumWare software radio system described in this paper provides more flexibility than any existing system by combining wideband digitization with software processing on a general purpose processors [BIWG98]. The system has provided a flexible, easy to use software radio research platform that has been used to implement multiple simultaneous cellular receivers, a frequency hopping network link and a physical layer patch between two incompatible radio systems.

This paper begins with comprehensive review of software radio research and applications. The SpectrumWare system is then described together with a few representative applications. The paper concludes with a discussion of some potential applications of software radio technology to wireless networking, and identifies some key research challenges that must be addressed in order to apply current software radio research to wireless networking applications.

2 Software Radio Research

Software radio research has been primarily motivated by interoperability problems that result from the implementation of radios in dedicated hardware. The first significant software radio was the SpeakEasy system [LU95] which was designed to emulate more than ten different military radios. Commercial interest was spurred by an RFI from Bell South Cellular on Software Defined Radio [Blu95]. The principle commercial application driver in the U.S. is the multiple in-

*vanu@lcs.mit.edu

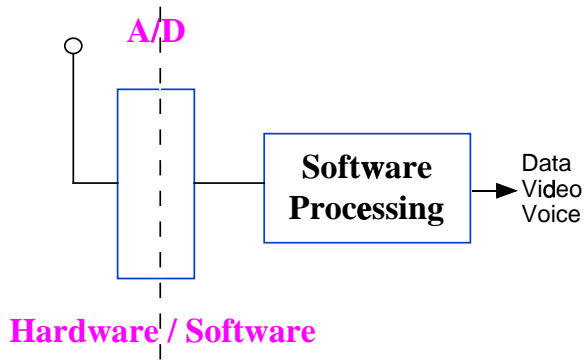


Figure 1: Ideal software radio.

compatible cellular and PCS communications. In Europe the widespread deployment of GSM has mitigated interoperability problems, but there is significant interest in using software radio to enhance services for third generation cellular systems [?].

The ideal software radio, illustrated in Figure 1, would directly digitize the entire band of interest, transport the stream of digital samples to memory where it can be directly accessed by a microprocessor. Current wideband receivers, A/D converters, I/O systems and processors cannot meet the requirements imposed by a direct implementation of this architecture. Current research in the area of software radios covers system design as well as the development of the enabling technologies of tunable wideband front-ends and the A/D converters capable of digitizing wideband signals with a signal-to-noise ratio sufficient to enable digital cellular applications.

Ideally, the front-end could be tuned to any band of interest, and would be capable of capturing the entire band of interest (e.g. 26 MHz for the 900 MHz ISM band). Most practical solutions involve several discrete front ends, each of which is optimized for a certain band. Active research aimed at producing a single chip tuner that is capable of operating in the 2000 MHz range is being performed as part of the DARPA Glomo program¹. A discussion of the requirements and challenges associated with the design of a wideband tuners can be found in: [Bra98a] [Wep95] [Ako97].

The development of A/D converters capable of digitizing the entire cellular band with enough resolution to implement the digital cellular standards in software is a significant research challenge. The key parameter is the spurious free dynamic range, which is a measure of the non-linear sources of error in the converter, and is often the limiting factor in the performance of high-speed converter. The GSM system imposes the most stringent constraints, requiring a 91 dB SFDR with a minimum sampling rate of 24 MHz. An in depth discussion of the research challenges can be found in

¹<http://www.darpa.mil/ito/glomo>

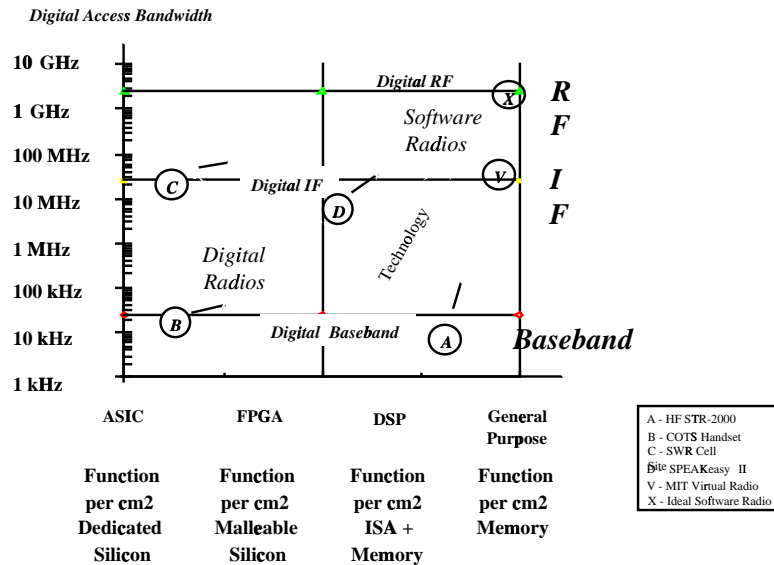


Figure 2: Software Radio Phase Space

[Wal98] [Bra98b] [WHS96].

2.1 Software Radio Systems

Software radio systems are designed for a range of applications ranging from low-power handheld devices to fixed basestation units. As a result, different systems employ a different technologies and architectures. Joe Mitola has proposed a software radio phase space [Mit98] which is a useful metric for comparing software radio systems. The software radio phase space, shown in Figure 2, represents a given radio implementation in terms of the digital access point and the degree of programmability. The vertical axis is not simply the frequency at which the A/Ds and D/A s operate, but the point at which the processing is fully programmable. For example, consider a cellular system that digitizes a 12 MHz band at a rate of 25 MHz and then uses dedicated hardware to select out a single 30 kHz channel, with a sampling rate of 60 kHz, to be processed in software. The digital access point would not be 25 MHz, but the 60 kHz sample rate that is subject to programmable processing. The horizontal access represents the degree of programmability of the underlying technology. Four different base technologies, ASICs, FPGAs, DSPs and general purpose processors are placed on the axis as a guide for mapping different implementations into this space.

The choice of processing technology is a trade-off between flexibility and power consumption that is determined by the application requirements. For example, low power handheld implementations cannot support the power requirements of DSP or general purpose processors, so a less flexible technology, such as re-configurable logic [?] must be used.

The SpeakEasy software radio system was designed to be portable, but not handheld, and has a less stringent power constraint. The system emulates more than 10 existing military radios, operating in frequency bands between 2 and 200 MHz. In order to achieve the necessary processing capability with reasonable power constraints, a multichip module containing four TMS320C40 DSP processors was developed. The modules being developed under phase II of the program run at a clock speed of 50 MHz, which provides 200 million FLOPS, 1,100 MIPS and 300 Mbytes per second I/O, while consuming less than 10 watts of power.

The SpectrumWare software radio [BIWG98] described in the next section demonstrates the feasibility of using a general purpose processor coupled with wideband digitization to implement a software radio. This system is closer to the upper right corner of the phase space than any other system reported in the literature [Mit99]. While current technology does not allow for this approach to match the power consumption of re-configurable logic or low power DSP processors, there are many advantages to the use of a general purpose processor, including: greater flexibility in the range of functionality that can be implemented; easy porting between processors allowing the software radio platform to track the performance of Moores Law; coupling between the application and the radio is much tighter allowing for better system optimization; and improved resource utilization since all processing functions run on the same platform.

3 SpectrumWare

3.1 System Architecture

The SpectrumWare system architecture shown in figure 3 makes only one concession to the ideal architecture depicted in figure 1, which is to first downconvert a wideband of the spectrum to an IF frequency and then digitize it. The center frequency of the RF band is selectable in software and it is important to note that what is downconverted is not just a single channel but a wideband (e.g. 10 - 20 MHz). For most systems this enables dynamic modification of the multiple access protocol, since it is implemented in software. Other than the restriction of looking at one particular band at a time, the system is functionally equivalent to the ideal software radio.

The system has four components:

- Signal acquisition
- I/O sub-system
- Programming Environment
- Radio specification model

Together, the signal acquisition and I/O sub-systems digitize a wideband of the RF spectrum and place it in memory

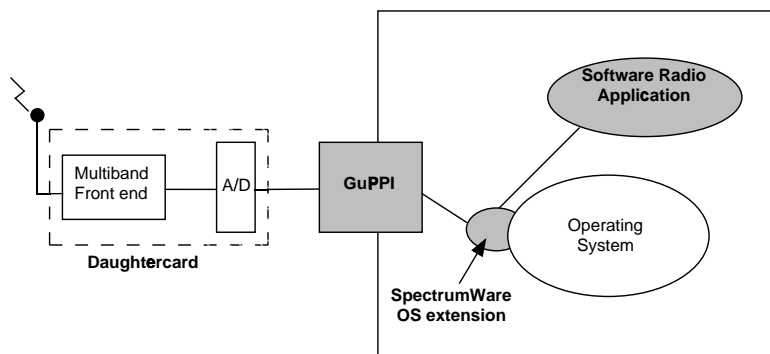


Figure 3: SpectrumWare System Architecture

that can be accessed by the processor. The programming environment is a modular system that allows for data-intensive real-time signal processing applications to be created and dynamically modified. The radio specification model provides a framework for specifying the software implementation of radio, and enables the specification of incremental changes to the processing.

3.1.1 Signal Acquisition

The signal acquisition utilizes commercially available mixers, filters and A/D converters and is capable of digitizing any 10 MHz band located between 100 kHz and 2.6 GHz. The signal acquisition was designed to be a modular component that could be replaced with high performance, low power solutions as they become available.

3.1.2 I/O sub-system

The General Purpose PCI Interface (GuPPI) was used to transport the samples from the D/A converter to memory, and from memory to the A/D converter. The high throughput requirements of software radio applications (e.g. digitizing the 12.5 MHz wide cellular A-side band at 25.6 MHz with 16 bits of resolution produces at data rate of 409.6 Mbits/sec) expose two bottlenecks in the existing workstation architecture. First, workstations lack a high-throughput port into which our front end can be connected, creating the need to develop custom hardware. Second, the path between a device driver and the application is rather inefficient, requiring modifications to the operating system. For comparison, the VuSystem [HAI⁺95] reported sustained throughput of 100 Mbits/sec to the application with an unmodified Digital Unix operating system.

There are two main components in the architecture of the I/O system: the GuPPI (for General Purpose PCI I/O), which physically connects the analog front end to the workstation's I/O bus, and operating system additions to the virtual memory system, which provide the means for the application to access the sample streams.

The GuPPI implements a new variant of scatter/gather DMA that we have named *page-streaming*. This approach takes advantage of the fact that the I/O is a continuous, regularly spaced stream of samples to reduce the overhead associated with the transfer.

The operating system virtual memory additions provide the low-overhead, high-bandwidth transfer of data between the application and the device driver and the external interface to the application. To the application, the GuPPI appears to be a standard Unix device with copy semantics. However, virtual memory manipulations are used to make the read and write system calls to the GuPPI copy-free. If the calls were implemented in the usual manner, there would not be enough CPU cycles available to copy the data stream from kernel memory to applications memory. Using the virtual memory manipulations, this overhead was reduced to less than one half of a cycle per input sample.

The maximum rate at which an application using the GuPPI driver can sustain a continuous flow of input samples GuPPI is 512 Mb/s, and the peak transfer rates are 933 Mb/s for input and 790 Mb/s for output. Additional details on the design and performance of the GuPPI can be found in [Ism98].

3.1.3 Programming Environment

SPECTRA is a Signal Processing Environment for Continuous Real-time Applications. The primary design goals were to support real-time signal processing applications on a general purpose platform and to create a programming environment that provided for a simple and straightforward implementation of signal processing functions in order to reduce code development and maintenance overheads. The system embodies a set of abstractions and design rules that allow for the implementation of modular, dynamically re-configurable real-time signal processing systems while enabling significant software reuse.

The key design aspects are the data-pull model, the stream abstraction, and the partitioning of in-band and out-of-band functions. The design aspects are reflected in the SPECTRA architecture, which consists of three components, processing modules, connectors and an out-of-band script. Together, these components comprise an extremely flexible system that is capable of handling the data intensive signal processing associated with many wireless communications systems.

A visual description of the programming environment is shown in Figure 4. The system is partitioned into in-band and out-of-band axes, in a manner similar to that used in the design of the VuSystem [Lin94]. The in-band axis is where the temporally sensitive, computationally intensive work takes place. The out-of-band axis is used for control, configuration and inter-module communication. All of the

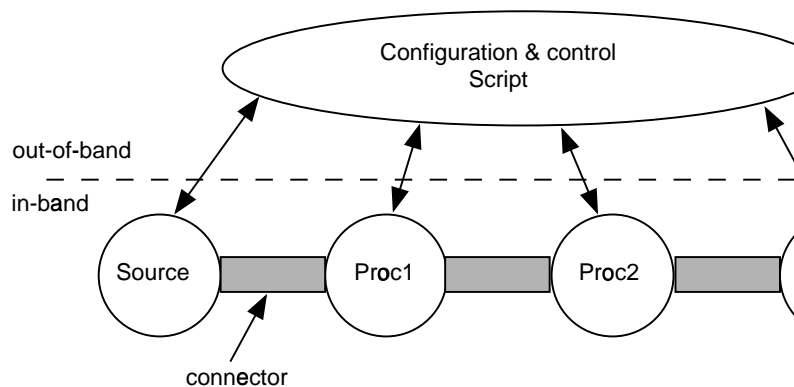


Figure 4: Graphical description of the programming environment.

application specific functionality is contained in the out-of-band section. This partitioning allows for maximal re-use of the computationally intensive in-band signal processing modules, since none of their functionality is specific to a particular application. The design of this programming environment incorporates the data-pull model, and the data is pulled down the pipe by the sink.

The SPECTRA programming environment employs a novel “data-pull” model, which was designed to overcome some of the limitations of a traditional data flow implementation. On the surface, the approach looks very much like data flow, as a graph of interconnected modules is defined to create a signal processing system. In a typical data flow implementation, the data is pushed from the source to the sink, in the data-pull model, however, the execution is driven by the data sink which requests data, as it is needed, by the upstream modules. This data is computed lazily, which can lead to significant computational savings.

The data-pull approach has three significant advantages:

- Improved computational efficiency resulting from the ability to transparently implement lazy evaluation.
- Rapid and efficient response to changes in the processing requirements.
- Improved performance on platforms employing data caching.

Most signal processing and media toolkits utilize some form of a data flow model [LWT94] [MBK+97]. One of the defining characteristics of a data-flow system is that execution is triggered by the availability of data [GBBG86]. In a modular data-flow processing system, this means each individual module can begin execution as soon as data is available at its input port. However, this can lead to extra work being performed and limit the dynamic flexibility of the system, limitations that are overcome by the data-pull approach.

In order to realize software implementation of all of the pro-

cessing required to transform between the samples streams produced by A/D converters and the application data the system must handle a wide variety of data types. The output of an A/D converter (or input to a D/A converter) is most naturally represented as a continuous stream of samples, which is parameterized by a sampling rate and resolution. In digital communications systems these streams are processed to produce a sequence of symbols. These symbols are often aggregated into frames with error detection and correction information. Once the data reaches the application, it may be represented as network packets, video frames or even converted back to a sample stream as in the case of audio transmission over the internet.

Although this may seem like a disparate collection of data types, the SPECtRA stream abstraction unifies them into one object heirarchy. The fundamental observation is that the input or output of any processing function can be represented as a stream of data objects. The A/D samples naturally form a stream, video and network data can be viewed as streams of frames and packets respectively. For purposes of discussion, these objects contain the data as well as a time stamp for each data unit.

For many signal processing applications, a stream is a natural I/O model. Functions such as filters operate on a continuous stream of input samples and produce a continuous stream of output samples. Many systems use finite buffers to pass data between one processing module and the next, but this leads to considerable extra code to take care of end conditions and make the sequence of finite buffers appear to be a seamless infinite stream of data. Furthermore, since the code required to achieve this effect is dependent on the processing function, this extra code must be incorporated into each processing module. The stream abstraction is implemented in the connectors, relieving the signal processing programmer of the burden of managing the data stream. From a processing module's perspective, these streams can be thought of as infinite buffers. This abstraction facilitates the translation from a mathematical representation of a signal processing algorithm to actual signal processing code. This has resulted in very small source code required for applications (e.g. 600 lines of code for an AMPS receiver), and very short development times.

4 Radio Specification Model

Radio systems have traditionally been divided into several separate stages, such as the RF, IF and baseband processing stage. This stages were defined primarily to facilitate hardware implementation. This chapter describes a software framework for specifying the signal processing requirements for a wireless communication system.

The software radio specification model consists of several well-defined processing layers, which can be used to com-

pletely specify a wireless communications system. This is a refinement of the OSI layering model [Tan88], which is a subdivision of the traditional Physical layer. The signal processing performed by this layer can be naturally subdivided into a finer grain model, but has traditionally been lumped into one layer because of its implementation in dedicated hardware. For our purposes, however, this is too coarse. To interoperate with different networks, it may only be necessary to change small parts of the existing layers. For example, two different systems may employ the same modulation and coding but use different multiple access protocols, or a given system may only need to change the type of coding to dynamically adapt to changing channel conditions. To facilitate this flexibility, we would like to create new communication systems by simply combining existing functional modules, rather than by writing a new piece of software for each network interface that encompasses all of the functions in the link and/or physical layers.

The definition of new layers is not something to be done lightly. The new layers were defined according to the design principles of the OSI model [Tan88], in particular the principle that the number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy. The network layering model was revisited in the light of software signal processing that allows modification of functions that were traditionally placed in the same layer since it was necessary to implement them in dedicated hardware.

Figure 5 illustrates the layering model, and its relation to the OSI model. One of the goals of layering is to minimize the information flow across the layer boundaries. This is enforced by the implementation of clean interfaces. There are five different data types that exist at the interfaces of the software portion of the layered structure described above:

- **Continuous signals** represent analog signals, they are parameterized by a bandwidth and dynamic range.
- **Discrete Signals** represent digitized signals, they are parameterized by a sampling rate and resolution (i.e. the number of bits per sample).
- **Bits** simply represent a binary value, there is no inherent timing associated with a bit stream.
- **Bytes** represent 8-bit values, there is no inherent timing associated with a bit stream.
- **Frames** represent any type of framed bytes, such as ATM cells or IP packets. Some frames may contain time stamps, but there is no requirement that they do.

The data type required for each interface is indicated on the right-hand side of figure 5. The interface data structure also contains parameters relevant to the data type, such as sampling frequency and bits per sample in the case of discrete signals.

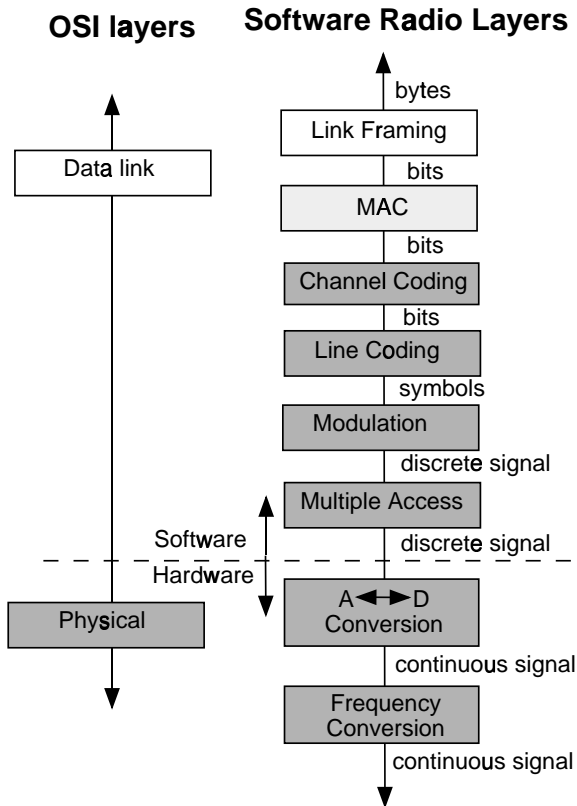


Figure 5: The Software Radio Layering Model shifts many physical layer functions into software.

The function of the link layer is to take the raw transmission facility and transform it into a line that appears to be free of transmission errors to the network layer. Examples of link layer protocols include HDLS and X.25. The traditional link layer is preserved in this model.

The media access control (MAC) functions are often postulated as a layer that sits between the physical and link layers [Tan88]. The primary MAC functions are mediating shared medium access and collision avoidance. There are many systems which use the same link layer protocols with different MAC protocols, thus it is appropriate to break the MAC functions out as a separate layer for the software radio model.

The remaining layers in figure 5 comprise the functions that are traditionally lumped into the physical layer, which is concerned with transmitting bits over a communication channel. The first sublayer is channel coding. A channel code is designed specifically for one of three purposes: error detection, error correction or error prevention [LM94]. They are used to reduce errors at the bit level, but the physical layer does not have to guarantee error free transmission.

Spectrum control over the physical layer can be achieved through the use of line coding, the second sublayer of the physical layer. Unlike channel codes, lines codes are not

concerned with errors, but rather controlling the statistics of the data symbols, such as the removal of baseline drift or undesirable correlations in the symbol stream. The desired parameters are determined by physical characteristics of the transmission medium.

The modulation sublayer is concerned with the transformation between symbols and signals. This not only includes traditional modulation functions such as QAM, but also channel equalization functions. It is tempting to define equalization as its own layer, but this would not be appropriate for two reasons. First, it would violate the layering principle that layer n on one machine *carries on a conversation* with layer n on another machine. In general, equalization is concerned with correcting for effects imposed by the channel, not by a corresponding function on the transmitter, so there is no comparable layer on the transmission side with which to communicate. The second reason is that equalizers are an integral part of the transformation between signals and symbols, and therefore should be part of the same functional layer as modulation techniques.

Finally, we have the multiple-access sublayer, which includes techniques such as TDMA and FDMA. Note that this multiple access layer performs a very different function from the MAC layer, which may also involve a multiple access technique. To illustrate this difference consider the IEEE 802.11 wireless networking standard [IEE97]. The MAC layer in 802.11 is CSMA, which provides shared access among all of the users of the network. The physical layer for 802.11 specifies two different multiple access techniques: frequency hopping and direct sequence spread spectrum. These multiple access techniques do not provide multiple access between users of a particular network, but for the sharing of the spectrum between different networks. Two different 802.11 networks can coexist in the same spectrum by using different spreading codes. The users of each network are not aware of the existence of the other network. In many cases, the other users of the band are not even data networks, but systems such as cordless phones, and wireless microphones. The physical layer multiple access provides isolation from all other systems using the band. When the physical medium is not shared by more than one network, as is the case with ethernet, there is no need for a physical layer multiple access technique. The choice of multiple access technique is usually independent of the modulation technique, and therefore should occupy its own layer.

In general a given system may contain only a subset of the layers. However, one could think of such a system as containing all of the layers, with default functions in some of the layers that do not manipulate the data in any way. This model provides a clean way of thinking about a system design, and describing incremental changes to a system that are required to adapt to changing environmental conditions and traffic loads.

5 Performance

The system performance is analyzed in the context of an AMPS cellular receiver application. The sequence of processing steps for the wideband AMPS cellular receiver is shown in figure 6. Since this wideband receiver demodulates a narrowband signal, the sample rate becomes lower at successive processing stages. It is worth noting that all of the system parameters, including the number and values of the channel filter coefficients and the sample rates, are under software control. This allows many of these parameters to be easily modified, even while the receiver is operating.

The first processing step is the channel selection filter. This module extracts a 30 kHz FM AMPS channel from a 10 MHz input band. This step uses a novel filter that combines the three conventional steps of translating the signal to baseband, low pass filtering and decimating. This step comprises the vast majority of the overall computational load. The channel selection filter consumes the raw samples from the RF front end at $R_S = 33$ MSPS and produces a complex baseband signal with a variable intermediate sample rate, R_D . The channel selection filter, in this implementation, is initially tuned to the nominal carrier frequency of the desired channel. No tracking of the carrier is required and any small frequency offset will create only a small DC offset at the output of the discriminator which is then removed by the audio filter.

The output of the channel filter is demodulated using a simple FM demodulation algorithm that approximates the derivative of the signal phase by the phase difference between successive samples, appropriately scaled.

The two final steps of processing are implemented using finite impulse response (FIR) filters. The first is a low pass decimating filter that removes high frequency components that would cause aliasing when the sample rate is reduced to the audio rate, typically 8 Ksamples/sec. The final step is a bandpass filter which removes out-of-band noise from the voice signal.

The implementation requires less than 40% percent of the CPU. Quantitative measurements of the audio output were not made, but the subjective quality is as good or better than a commercial AMPS cellular handset. Figure 7 quantifies the performance of the receiver in terms of cycles consumed, cycles while a DCU miss is outstanding and cycles during resource stalls. Cycles while a DCU miss is outstand-

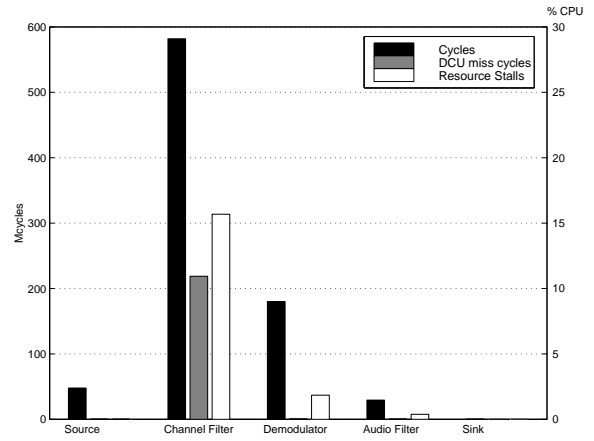


Figure 7: Performance results for AMPS receiver.

ing provides a measure of the cache utilization, this count is weighted by the number of outstanding cache misses. The cycles during resource stalls covers a wide variety of conditions that may cause the processor pipeline to stall, such as an icache miss or a miss-predicted branch. A full description of these counters is given in [Int98]. The channel selection filter is the most computationally intensive, and is also subject to the biggest performance hit due to caching. Slightly more than one third of the cycles consumed by this module are wasted waiting for the cache misses to be serviced. The cache misses are inherent, because this module reads in newly created data from the GuPPI and it must all be brought into the cache for the first time.

Subsequent modules benefit from the cache performance of the data pull model and display excellent computational performance with little variability due to cache performance. When the demodulator runs, it's input data has just been written into the cache by the channel filter, so there are very few cache misses. The same hold true for the input data to the audio filter. The audio filter experiences more cache misses than the demodulator does. This is because the audio filter must also access an array in memory where the filter taps are stored, and this array would have been evicted from the cache due to the data-intensive processing of the channel filter and demodulator. However, there is still a big performance win due the availability of input data in the cache.

A multi-channel receiver was designed to minimize the performance penalty due to caching in the first processing stage. The channel filter is simply four separate channel filters built into the same module, and they run serially, each writing data to the appropriate buffer. Figure 8 shows the block diagram for an application that demodulates four channels simultaneously. The percentages taken by each of the blocks is shown in the figure. Note that this filter requires considerably less than four times the number of cycles of the single channel AMPS filter. This is

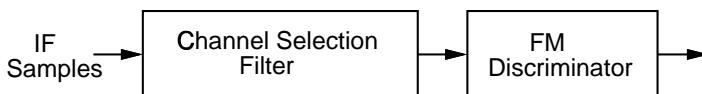


Figure 6: AMPS Receiver Software Architecture

due to a number of factors including: the overhead of only one function call for the four filter; improvement in icache performance for subsequent filters and warming of the data cache for both input data and filter tap arrays.

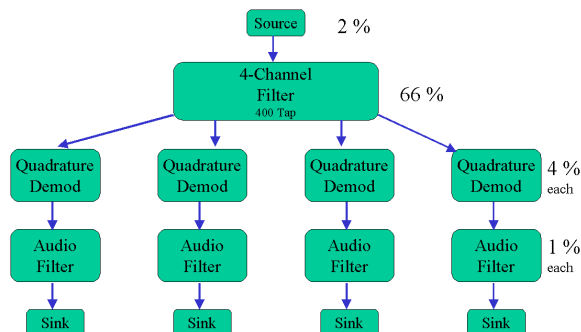


Figure 8: Block diagram of the 4 channel amps receiver. The percentages indicate the percent of the CPU utilized by blocks in that layer.

The multi-channel receiver example illustrates that the processing costs of a software system are very different from analog or digital hardware systems, which requires a different approach to radio architecture and algorithm design.

6 Mobile Networking Challenges

The SpectrumWare system permits any aspect of the communications stack to be modified to best meet the current environmental conditions, traffic constraints and user requirements. This capability opens many opportunities to improve wireless and mobile network systems. The following outlines a few potential applications, and the research issues that must be addressed to allow wireless networking applications to take full advantage of software radio technology.

6.1 Seamless Roaming

There are currently a wide variety of wireless connectivity options available to the mobile user, including satellite, cellular, metropolitan area and local area networks. Taken together, these system have the potential to provide wireless connectivity in wide range of geographical areas. Some of the network layer issues involved with roaming between different systems have been addressed [KB96], but the different physical layers still require the user to carry multiple physical radios, inhibiting seamless roaming. A software radio could detect the new system and execute the appropriate *radio code* to interoperate with that system.

The basic capability to roam across different systems was the motivation behind the development of software radio

for cellular telephony applications. Wireless networking applications present a more challenging problem, owing to the wide variety of standards that are in use and the multitude of bands that wireless networks operate in. In the cellular case, with a few well defined standards, it is quite reasonable to expect that the software for each of the standards could be resident on the phone. However, with wireless networking this is not a reasonable assumption due to the large number of existing standards, and the rapid pace of change in the industry.

To take advantage of the flexibility promised by software radio in the context of wireless networking, two mechanisms must be developed: Efficient detection algorithms that can rapidly identify the wireless system(s) operating in the local area, and a system for dynamically downloading and executing the radio code required to interoperate with that system. The code loading mechanism has the potential to increase that pace of innovation in wireless networks, as new standards can be deployed by downloading them to any radio that desires to connect to the system.

6.2 Customized Services and Channels

The ability to customize channels in wireless systems will result in better application performance and better spectrum utilization. Most systems have fixed channel widths, which only make use of all the available spectrum when the system is fully loaded. A mobile user might be able to get a wider channel during off peak hours, and the provider could generate revenue from more of the spectrum during this period. Application performance suffers because applications such as full- duplex real-time audio and file transfers can be best served by different tradeoffs between the error rate, latency and data rate, but the bandwidth and error correction in the physical layer cannot be adapted to these requirements.

The implementation of these mechanisms requires significant research into algorithms for dynamically determining spectrum usage, interfaces to allow for communication between the physical layer and higher level layers and metrics for quantifying the tradeoffs between bit rate, error rate, latency and robustness to various forms of noise for physical layer functions that can then be used to determine the appropriate physical layer components (e.g. modulation and channel coding) to be used in a given situation.

7 Summary

Software radio technology brings significant flexibility to wireless communications by implementing the physical layer functions in software that can be dynamically modified to best suit the current environmental conditions, traffic loads and user requirements. The SpectrumWare system described in this paper is capable of implementing all of the

physical layer functions in software for a wide range of wireless communications systems, and provides the processing capability to form a software radio testbed for wireless networking. While most software radio research to date has been motivated by commercial cellular and military applications, the capabilities of the SpectrumWare system suggest that software radio technology may be of most benefit to wireless networking systems in the future.

References

- [Ako97] Dennis M. Akos. *A Software Radio Approach to Global Navigation Satellite System Receiver Design*. PhD thesis, Ohio University, May 1997.
- [BIWG98] Vanu G. Bose, Michael Ismert, Matthew Welborn, and John Gutttag. Virtual Radios. *JSAC issue on Software Radios*, October 1998.
- [Blu95] Stephen M. Blust. Software Defined Radio - Industry Request for Information. Technical report, Bell South Cellular, December 1995.
- [Bra98a] Brad Brannon. Digital-radio-receiver design requires re-evaluation of parameters. *EDN*, pages 163–170, November 1998.
- [Bra98b] Brad Brannon. Wideband Radios Need Wide Dynamic Range Converters. *Analogue Dialogue*, 29(2), 1998.
- [GBBG86] Paul Le Guernic, Albert Benveniste, Patricia Bournai, and Thierry Gautier. SIGNAL - A Data Flow-Oriented Language for Singal Processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-34(2):362–374, April 1986.
- [HAI+95] Henry H. Houh, Joel F. Adam, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. *IEEE J-SAC*, 13(4):710–721, May 1995.
- [IEE97] IEEE. IEEE Std 802.11-1997. Technical report, IEEE, 1997. Working Group for Wireless Local Area Networks.
- [Int98] Intel. Intel Architecture Optimiztions Manual. <http://developer.intel.com/design/pro/manuals/>, 1998.
- [Ism98] Michael Ismert. Making Commodity PCs Fit for Signal Processing. In *USENIX*. USENIX, June 1998.
- [KB96] Randy H. Katz and Eric. A. Brewer. The Case for Wireless Overlay Networks. In *Proceedings 1996 SPIE Conference on Multimedia and Networking*, San Jose, CA, January 1996. MMCN '96.
- [Lin94] Christopher J. Lindblad. A Programming System for the Dynamic Manipulation of Temporally Sensitive Data. Technical Report MIT/LCS/TR-637, M.I.T., August 1994.
- [LM94] Edward A. Lee and David G. Messerschmitt. *Digital Communication*. Kluwer Academic Publishers, 2nd edition, 1994.
- [LU95] Raymond J. Lackey and Donal W. Upmal. Speakeasy: The Military Software Radio. *IEEE Communications Magazine*, 33(5):56–61, May 1995.
- [LWT94] Christopher J. Lindblad, David J. Wetherall, and David L. Tennenhouse. The VuSystem: A Programming System for Visual Processing of Digital Video. In *Proceedings of ACM Multimedia*, 1994.
- [MBK+97] Steven McCanne, Eric Brewer, Randy Katz, Lawrence Rowe, Elan Amir, Yatin Chawathe, Alan Coopersmith, Ketan Mayer-Patel, Suchitra Raman, Angela Schuett, David Simpson, Andrew Swan, Teck-Lee Tung, and David Wu. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Proceedings of the 7th InternatProceeding of NOSS-DAV'97*, St. Louis, Missouri, May 1997. Invited Paper.
- [Mit98] Joe Mitola. Software Radio Architecture A Mathematical Perspective. *JSAC issue on Software Radios*, 1998.
- [Mit99] Joe Mitola. Challenges in the Globalization of the Software Radio. *IEEE Communications Magazine*, 1st qtr. 1999. to appear.
- [Tan88] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, Englewood Cliffs, NJ 07632, second edition, 1988.
- [Wal98] R. H. Walden. Analog-to-Digital Converter Survey and Analysis. *JSAC issue on Software Radios*, 1998.
- [Wep95] Jeffery A. Wepman. Analog-to-Digital Convertors and Their Applications in Radio Receivers. *IEEE Communications Magazine*, 33(5):39–45, May 1995.
- [WHS96] J.A. Wepman, J. R. Hoffman, and J. E. Schroeder. An initial study of RF and IF digitization in radio recivers. Technical report, NTIA, 96. in preparation.