

# Accelerating Evolution of the Cellular Infrastructure using Software Radios

Alok B. Shah and Vanu G. Bose\*

*Vanu, Inc.*

## Abstract

*A software radio is a wireless communications device in which all of the physical layer functions are implemented in software. By simply downloading a new program, a software radio is able to interoperate with different wireless protocols, incorporate new services and upgrade to new standards. This paper focuses on the software engineering and system design issues related to software radio, and how careful design can enable the rapid deployment of new services as well as technology tracking of both wireless standards and processing technology. We describe our software radio architecture, which is based on wideband digitization, general purpose processors and application level software, and the applications of this architecture to the cellular infrastructure.*

## 1 Introduction

Innovation and technology tracking in wireless telecommunications infrastructure is slowed by the expensive hardware upgrades required to deploy new standards and services. Infrastructure based on software radio technology can enable new services and even new standards to be deployed via software upgrades to the existing infrastructure. However, in order to realize this benefit, all of the signal processing must be under the control of application level software. If any aspect of the system, such as channel selection, is left to dedicated hardware then the scope of new services and standards that can be fully deployed via software is limited.

This paper presents a software radio architecture that is capable of digitizing the entire band used by a particular cellular system (e.g. 12 MHz for an AMPS cellular system) and performs all of the processing, including channel selection and equalization, in portable application level software. The architecture incorporates three components: an I/O system to acquire and generate wideband IF samples; an operating system extension to enable the high data throughput required by software radio applications; and a programming environment to support real-time software radio applications on a general purpose processor. Our architecture

differs from other approaches to building programmable radio systems in that it utilizes wide band sampling coupled with general purpose processors and object oriented application level software. The system is portable across processing platforms which allows it to track the rapidly advancing processing technology, and the object oriented programming environment permits the rapid development of real-time signal processing applications. Our approach has many benefits, including:

- Deployment of new services via software downloads.
- Software upgrades to new standards.
- Reduction of risk associated with deploying new services.
- Decoupling of hardware and software upgrades.
- Reduced development time and increased software reliability.
- Seamless integration of new services and applications.

This paper begins with a review of software radio research and then describes the design and implementation of our software radio architecture along with a representative application. This is followed by a discussion of how this technology can be used in the cellular infrastructure to reduce cost and mitigate risk in the deployment of new systems and services through software downloads and incremental hardware upgrades that are decoupled from the software upgrades.

## 2 Software Radio Background

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 multiple incompatible cellular and PCS communication standards are the principal commercial application driver in the U.S. In Europe the widespread deployment of GSM has mitigated interoperability problems, but there is significant interest in

---

\*{abshah, vanu}@vanu.com. <http://www.vanu.com>

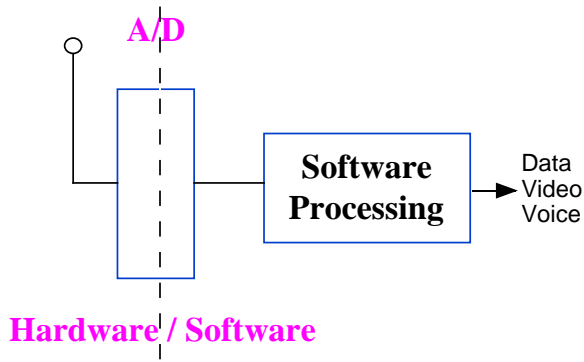


Figure 1: Ideal software radio.

using software radio to enhance services for third generation cellular systems [Tut99].

The ideal software radio, illustrated in Figure 1, would directly digitize the entire band of interest, transporting 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 tuners that are capable of operating in the 2 - 2000 MHz range is being performed as part of the DARPA Glomo program<sup>1</sup>. A discussion of the requirements and challenges associated with the design of 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 (SFDR), 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 converters. 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 [Wal99] [Bra98b] [WHS96].

<sup>1</sup><http://www.darpa.mil/ito/glomo>

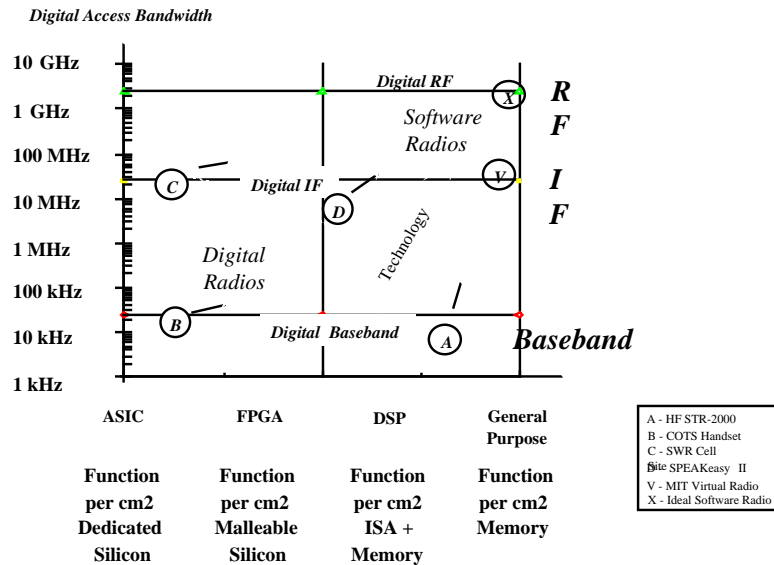


Figure 2: Software Radio Phase Space

## 2.1 Software Radio Systems

Software radio systems are designed for a range of applications ranging from low-power hand-held devices to fixed basestation units. As a result, different systems employ different technologies and architectures. Joe Mitola has proposed a software radio phase space [Mit99b] 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/As 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 hand-held implementations cannot support the power requirements of DSP or general purpose processors, so a less flexible technology such as re-configurable logic [Dic98] must be used.

The SpeakEasy software radio system was designed to be portable, but not hand-held, 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 software radio architecture presented in this paper provides more flexibility than any existing system by combining wideband digitization with software processing on a general purpose processor [Bos99a]. This system is closer to the upper right corner of the phase space than any other system reported in the literature [Mit99a]. 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 Moore's Law; much tighter coupling between the application and the radio allowing for better system optimization; and improved resource utilization since all processing functions run on the same platform. Furthermore, since the system has been shown to be portable, it will track advances in low power processors as they emerge.

## 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 daughter-card shown in Figure 3 is responsible for signal acquisition and digitization. The GuPPI I/O sub-system digitizes a wideband of the RF spectrum and places it in memory 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 entire system, including hardware, operating system

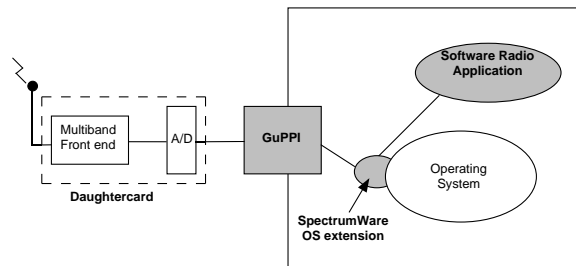


Figure 3: SpectrumWare System Architecture

modifications and application software is portable and currently runs on both Intel and Alpha workstations. We are currently porting the system to the strongARM processor. This portability allows our system to track the rapid advances in processing technology without requiring a significant redesign of the software. The three sub-systems are briefly described below. Further details on the system design and implementation can be found in [Bos99a] [Bos99b] [BIWG99].

#### 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 A/D converter to memory, and from memory to the D/A 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 a 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<sup>+</sup>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 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 [Is98].

### 3.1.3 Programming Environment

The primary design goals of the programming environment 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 reconfigurable 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 environment 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

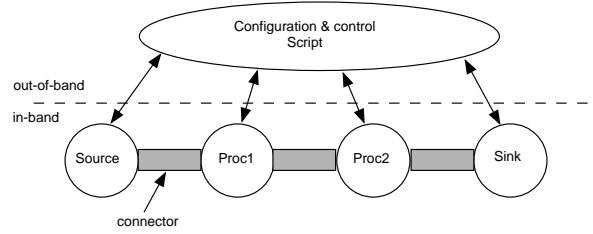


Figure 4: Graphical description of the programming environment.

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 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.

In order to realize software implementation of all of the processing 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 stream abstraction unifies them into one object hierarchy. 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.

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 voice receiver) and very short development times.

## 4 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 5. 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

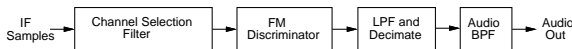


Figure 5: AMPS Receiver Software Architecture

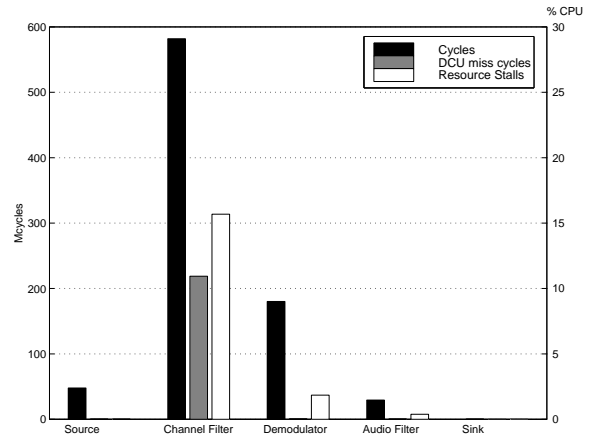


Figure 6: Performance results for AMPS receiver.

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.

Our approach differs from other DSP approaches in that we are using a general purpose processor and operating system to perform the real-time signal processing. Our system was designed to accommodate the uncertainty in execution times due to features such as data caches while taking advantage of the higher clock speeds and multiple forms of parallelism available on a general purpose platform. Figure 6 quantifies the performance of the receiver in terms of cycles consumed, cycles while a Data Cache Unit (DCU) miss is outstanding and cycles during resource stalls. Cycles while a DCU miss is outstanding provides a measure of the cache utilization, since 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, its input data has just been written into the cache by the channel filter, so there are very few cache misses. The same holds true for the input data to the audio filter. The audio filter experi-

ences 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 to 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 7 shows the block diagram for an application that demodulates four AMPS 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 due to a number of factors including: the overhead of only one function call for the four filters, improvement in icache performance for subsequent filters and warming of the data cache for both input data and filter tap arrays.

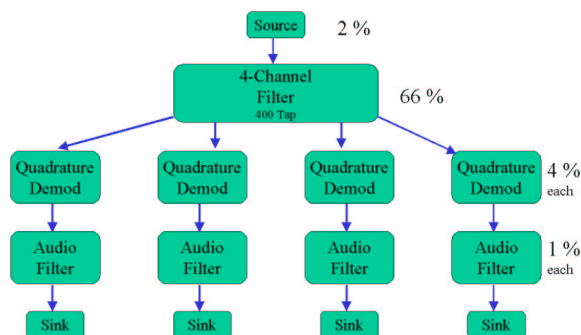


Figure 7: Block diagram of the 4 channel amps receiver. The percentages indicate the percent of the CPU utilized by blocks in that layer.

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

## 5 Application to the Cellular Infrastructure

The flexibility provided by software radio technology provides an opportunity to rapidly evolve the cellular infrastructure at a relatively low cost.

Cellular network operators currently have two major concerns: increasing revenue through new services and gearing up for the next generation of cellular standards. The following discussion describes the benefits of software radio

technology as compared to current systems.

### 5.1 Introduction of New Services

The revenue model for network operators generally calls for subsidizing handset costs in order to profit from monthly charges. Therefore revenue growth comes from two sources: signing up new users and extracting additional incremental revenue from existing users. The former is occurring rapidly, as cellular usage is dramatically increasing on a global scale. The latter is achieved in large part by introducing new services which convince customers to use their phones more often.

Chief among these new services is data. Already, carriers such as AT&T Wireless, Bell Atlantic and Nextel are rolling out systems which allow users to send and receive e-mail, check the latest stock quotes and read news headlines from their phones [Haf99]. International Data Corp. states that 15% of the 64 million current US mobile phone customers use their phones for data, with projections of 70% of 108 million customers doing so by 2002 [Els99]. This staggering number demonstrates the importance of data services to wireless carriers going forward.

There are many new services based on data on the horizon, including web-based organizer functionality, over-the-air downloads of books and music and airline ticket purchasing. The issue for carriers becomes how to best implement and roll out these new features at the lowest cost. The software radio architecture is suited to this situation in a number of respects.

- Lower deployment costs. As long as new services will be operated in the same spectrum as the existing cellular and PCS services, software radio infrastructure would require no change in hardware.
- Faster deployment. Making hardware changes to implement new services is not only expensive, but slow as well. It requires the installation of new hardware at each basestation. Software-only upgrades could be done simply by downloading the code to each base station, thus avoiding the need for visits to each site.
- Risk mitigation. Since the implementation of new services is based entirely in a modular software environment, the cost of deployment is low, thus the financial risk involved with deploying a service which is not popular with users is significantly reduced.
- Software portability. The modular design and platform-independent nature of the software allows for significant software re-use and portability, which leads to lower development costs and better technology tracking.

There are a wide variety of services that could be deployed in the cellular infrastructure. Software radio technology makes it possible to deploy these services rapidly, and experiment with new services with little additional investment. Two examples of potential new services are described below:

### 5.1.1 Voice Recognition

One concern about the introduction of new services is that current customers might have to purchase a new phone in order to use the new features. As stated above, network operators generally subsidize the cost of handsets, and therefore the only reason that they would want current customers to purchase a new handset would be for access to lucrative new features. Software radio infrastructure could allay this concern by implementing some features entirely in the infrastructure and having a sequence of buttons which allows older handsets to access the feature.

An example of such a feature is voice recognition. The latest, most accurate voice recognition algorithms require too much memory and too many MIPS to run on a handset, and thus much of the work is focused on infrastructure-based systems. There are two problems with this. One is that these network-based voice recognition systems require adding new hardware to the infrastructure. The second is that existing phones are unable to access this feature. Since a software radio architecture can allow features to be implemented with no additional hardware, the first problem is solved. As for the second problem, one solution is for users of older phones to enter special sequences of keys in order to run the voice recognition functionality. The software radio base station would interpret the key sequence to be a voice recognition function call, and run the voice recognition software. This function could be used for many purposes. For example, the voice recognition could be used to surf and search the web from you phone, or the voice commands could be re-routed back to the phone to enable voice control over the user device itself.

Our system is compatible with available voice recognition software [SHL<sup>+</sup>98], and permits the functionality to be added as a software only add-on to the device. Integration of the voice recognition with network applications is greatly simplified, since the entire system runs on a general purpose processor and operating system.

### 5.1.2 Push-to-Talk

The general pattern that has been followed in wireless communications is for one operator to roll out a new service on its network and have the other operators deliver the same service after some period of time. For the push-to-talk feature, however, this has not been the case.

Only one wireless operator, Nextel, offers a true push-to-talk function for its mobile phones. The reason that this feature

is not yet being offered by other carriers is mainly due to the infrastructure limitations. Therefore other network operators would have to make costly hardware additions to their infrastructure in order to implement the feature, and the expense of both developing and deploying this upgrade is prohibitive. A software radio infrastructure would permit the implementation of push-to-talk in a local coverage area without any additional hardware.

## 5.2 Software Upgrades to New Standards

Deploying new infrastructure is a significant cost to any service provider. At the same time, it is important for operators to stay on the technology curve and offer advanced features and standards as they are developed. This has caused deployment costs to become an ongoing expense for carriers.

When it comes to wireless data, telecom operators are facing a dilemma. The earliest wireless data systems, which employed the CDPD protocol, required millions of dollars in investment for build out. Now, wireless data can be provided more efficiently by modifying the current digital cellular voice networks to also send and receive data. In a few years, however, third-generation (3G) networks, which promise much higher data rates than today's data networks, will be ready for roll out. So operators have two choices: they can either provide data services over CDPD networks until 3G network deployment, or they can spend millions of dollars to modify the current cellular systems and then spend millions more on 3G [Els99]. Clearly, neither choice is an optimal one. If they wait for the introduction of 3G, then the operators risk losing customers who want the functionality now, and there is certainly no guarantee that those customers will return to an operator when its 3G network is deployed. Operators who decide to upgrade their networks now are spending tremendous amounts of money on systems which will be obsolete in a few years.

A software radio architecture can alleviate this problem. Once a software radio infrastructure is in place, upgrading it for 3G operation would require minimal, if any, hardware changes. Terrestrial 3G service has been allocated 60 MHz for uplink and 60 MHz for downlink. It is likely that each operator will get 15-20 MHz of spectrum in each direction. Therefore our software radio basestation could potentially be upgraded to 3G from a hardware perspective simply by shifting the frequency band which is being digitized, and downloading the 3G air standard in software.

As new services and standards are deployed, it is likely that more computational horsepower will be needed at the basestation. Our software radio architecture is unique in that it allows the hardware and software upgrades to be decoupled. This enables incremental upgrades to parts of the basestation over time, allow the cost to be spread out over a longer period of time. Computational power can be increase in one

of two ways: upgrading the CPU or adding new CPUs to take advantage of the parallelism that is well supported by general purpose processors and operating systems.

This processor change would have no impact on the software. Unlike many DSP software solutions, our software is completely portable across several processing platforms. Furthermore, general purpose processors do a much better job of maintaining backwards compatibility due to the large installed user base. Together these allow for the decoupling of hardware upgrades from software upgrades. Therefore Moore's law of advances in processor technology can be exploited without requiring software changes. Any the underlying processing technology can be upgraded over time.

## 6 Summary

Software radio technology brings significant flexibility to the cellular infrastructure by implementing the physical layer functions in software. This enables new services and standards to be deployed as software upgrades, which significantly reduces the cost and risk associated with upgrading the infrastructure.

Our software radio system is portable across several processing platforms, allowing it to track the rapidly advancing processor technology without requiring significant changes to the software. Furthermore, the use of an object oriented approach to the implementation of real-time signal processing code has resulted in very small source code bases for applications, and code re-use of up to 90% for some applications. This has resulted in reduced development costs and considerably accelerated the development of new systems in software.

The current issues in establishing a worldwide 3G standard point to the need for flexible implementations of wireless infrastructure. Recently, Ericsson and Qualcomm agreed on a CDMA-based standard for 3G implementation. However, this standard is actually made up of three optional modes: direct-sequence frequency-division duplex, multi-carrier frequency-division duplex and time-division duplex. Therefore software radio will enable carriers to deploy infrastructure prior to deciding which flavor of third-generation system to deploy.

Today, software radio based cellular infrastructure will reduce interoperability problems and facilitate the migration to 3G systems. In the future, the flexibility of the software radio architecture will ensure continuing interoperability with new standards as well as enabling the deployment of a wide variety of new services.

## References

- [Ako97] Dennis M. Akos. *A Software Radio Approach to Global Navigation Satellite System Receiver Design*. PhD thesis, Ohio University, May 1997.
- [BIWG99] Vanu G. Bose, Michael Ismert, Matthew Welborn, and John Guttag. Virtual Radios. *JSAC issue on Software Radios*, February 1999.
- [Blu95] Stephen M. Blust. Software Defined Radio - Industry Request for Information. Technical report, Bell South Cellular, December 1995.
- [Bos99a] Vanu G. Bose. *Design and Implementation of Software Radios Using a General Purpose Processor*. PhD thesis, Massachusetts Institute of Technology, April 1999.
- [Bos99b] Vanu G. Bose. The impact of software radio on wireless networking. *Mobile Computing and Communications Review*, January 1999.
- [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.
- [Dic98] Chris Dick. FPGA's for High Performance Communications. In *International Symposium on Advanced Radio Technologies*, 1998.
- [Els99] Peter Elstrom. Hello Internet. *BusinessWeek*, 3 May 1999.
- [Haf99] Katie Hafner. Web Phones: The Next Big Thing? *New York Times*, 15 April 1999.
- [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.
- [Int98] Intel. Intel Architecture Optimizations Manual. <http://developer.intel.com/design/pro/manuals/>, 1998.
- [Ism98] Michael Ismert. Making Commodity PCs Fit for Signal Processing. In *USENIX*. USENIX, June 1998.
- [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.
- [LU95] Raymond J. Lackey and Donal W. Upmal. Speakeasy: The Military Software Radio. *IEEE*

*Communications Magazine*, 33(5):56–61, May 1995.

- [Mit99a] Joe Mitola. Challenges in the Globalization of the Software Radio. *IEEE Communications Magazine*, 1st qtr. 1999. to appear.
- [Mit99b] Joe Mitola. Software Radio Architecture A Mathematical Perspective. *JSAC issue on Software Radios*, 1999.
- [SHL<sup>+</sup>98] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. Galaxy-II: A Reference Architecture for Conversational System Development. In *IC-SLP'98*, Sydney, Australia, November 1998.
- [Tut99] Walter H Tuttlebee. Software Radio Technology: A European Perspective. *IEEE Communications Magazine*, February 1999.
- [Wal99] R. H. Walden. Analog-to-Digital Converter Survey and Analysis. *JSAC issue on Software Radios*, 1999.
- [Wep95] Jeffery A. Wepman. Analog-to-Digital Converters 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 receivers. Technical report, NTIA, 96. in preparation.