Smart Radio Challenge Final Report
Spectrum Access for First Responders

Peiman Amini, Ehsan Azarnasab, Salam Akoum, Xuehong Mao, Harsha Rao

Electrical and Computer Engineering Department
The University of Utah

Team leader: Peiman Amini

Advisor: Dr. Behrouz Farhang-Boroujeny

September 29, 2007

Abstract

The issue of spectrum access is the single greatest obstacle in the development of a wireless network geared towards first responders. First responders need to effectively and reliably communicate with each other to provide help for the people in need. Smart radios or cognitive radios present themselves as strong candidates for the future of emergency communications. From a user's perspective, a cognitive radio network should operate identically to a standard wireless network. However, cognitive radio nodes are designed to be "aware" of the other users of the spectrum, and avoid interfering with these users. A cognitive radio network is built to coexist in a given portion of the spectrum with the legacy devices to which the spectrum is assigned.

We use Filterbanks for spectrum sensing and we show that this method exhibits superior performance in terms of the spectral dynamic range when compared to the conventional Fast Fourier Transform (FFT) techniques. We perform packet detection and channel equalization using a cyclic preamble. A fractionally spaced equalizer is used for both channel and timing recovery. The different processing tasks are divided between a TI c64x+ DSP and a XILINX Virtex IV FPGA. An ARM9 core is used for interfacing and running the Greenhills operating system.

This report covers the cognitive radio modem components that we have implemented on the Small Form Factor (SFF) Software Defined Radio (SDR) platform, and the theory behind their design. It summarizes the work we have achieved throughout the phase II of the 2007 Smart Radio Challenge.

Acknowledgement

We would like to thank the hardware and software support that we have received from SDR Forum, Lyrtech, Texas Instruments, Xilinx, Mathworks, Greenhills, Prismtech, Zeligsoft, and Synplicity. It has been a challenge to learn this extraordinary collection of hardware and software, but indeed, a rewarding one. We are truly grateful to all the supporting companies.
We would also like to sincerely acknowledge Professor Behrouz Farhang-Bouroujeny for his precious guidance and feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In disaster situations, it is absolutely crucial for law enforcement, rescue agencies, and other first responders to have the ability to communicate and exchange information quickly and reliably. Since wired networks cannot survive all types of disasters and are often impractical, communication through wireless networks is the preferred choice. However, spectrum allocation plays a major role in the development of any wireless network. Given the relative scarcity of available spectrum, coupled with the amount of time required to approve new users, the issue of spectrum access is the hardest task in the development of wireless networks geared towards first responders.

Cognitive radio technology is the natural choice to overcome this access problem. First responders, acting as cognitive nodes, can use the idle portions of the spectrum to communicate with each other [1]. From a user's perspective, a cognitive radio network should operate identically to a standard wireless network. However, cognitive radio nodes are designed to be "aware" of the other users of the spectrum, and avoid interfering with these users. A cognitive radio network is built to coexist in a given portion of the spectrum with the legacy devices to which the spectrum is assigned. In this work, the Family Radio Service (FRS) band is chosen to implement a cognitive network of nodes able to transmit voice and data traffics with different Quality of Service (QoS) requirements [2].

Construction of a cognitive network for first responders presents many difficult challenges, the most obvious of which is how to fulfill the "awareness" requirement. Each node must be able to *sense* the channel for primary users, *i.e.* to identify the presence of *licensed* communications over the portions of spectrum, and share this information with the other nodes to allow the cognitive nodes to communicate reliably while avoiding the legacy devices. In our problem setup, 200 carriers (25KHz bandwidth each) are used by the cognitive radio to transmit voice or data, provided that the legacy users are not using the channel. The method used for sensing should feature a high spectral dynamic range to enable the detection of the low power users.

We choose the filterbank sensing method [3, 4]. Filterbanks achieve a high dynamic range and perform better than FFT in terms of detecting low power users when both users with high power and low power are present. The sensing is done by each node in the network and the results are passed to a base station which combines all the sensing information to compile a channel state information (CSI), which is then broadcasted to all the nodes. The CSI is also used by the base station for channel allocation. Control channels are used for exchanging sensing information and control messages such as channel assignment between the leaf nodes and the base station. This distributed sensing method has the advantage of solving the hidden node problem of power sensing methods.

Packet assembly is performed such that the voice and data traffics pass through the same signal processing blocks at the transmitter and at the receiver. Cyclic preamble is appended to the payload to be used in packet detection and carrier recovery. A fractionally spaced equalizer is utilized for channel and timing recovery. The equalizer is trained with the cyclic preamble and tuned with a decision directed algorithm [5].

The cognitive radio modem is implemented on the Lyrtech Software Defined Radio Development Platform (SFFSDR) [6]. Design decisions such as dividing the tasks between the FPGA and the DSP subsystems, and choice of the appropriate methods for implementing each block are made in order to optimize the use of hardware resources. We simulate the functional modem in MATLAB and Simulink, then we use System Generator for DSP for the implementation of the FPGA blocks. Simulink and Realtime Workshop (RTW) are used to develop some individual modules of the DSP subsystem while the majority of the code is developed independently and tested as MATLAB MEX files before optimizing for the target processor. TI Code Composer Studio (CCS) is used to compile the code for the DSP subsystem. The algorithms are tested by running the code on hardware and testing the results inside MATLAB interfaced with the board. Networking simulations are performed in DEVSJava exploiting the Hardware in the Loop (HIL) technique [7], that we have developed to test the networking algorithm when hardware resources are not available.

The report is organized as follows. Chapter 2 presents the cognitive radio modem setup including the transceiver, the filterbank channel sensing method and the MAC layer design. In chapter 3, we present the system design and software and hardware development procedures used to implement the modem on the SFFSDR platform. We also present the software architecture and integrability of various software packages in more detail. At the end of chapter 3, the measurement results of the filterbank channel sensing methods are included. Chapter 4 presents the time line of our project from the beginning of phase II until the end of September 2007. The difficulties and limitations we encountered throughout the project, the companies who helped fund our group's effort and the facililites we used in our project are discussed in chapter 5. In chapter 6, we list the deliverables of our project and present the future plans and potentials of our system.

Chapter 2

System Design

In this section, we present our sensing methodology which is the most important part of the cognitive radio. Then we describe the signal processing algorithms that we have chosen to implement a functional cognitive radio with minimal complexity. The system is broken down into four parts: (i) Channel sensing, (ii) Transmitter, (iii) Receiver, and (iv) MAC layer.

2.1 Channel Sensing

Choosing an appropriate channel sensing method is a key element in developing a functional cognitive radio system. This method should take into consideration the discrepancy in the received power levels from the different users and needs to feature a high dynamic range in order to reliably detect the spectrum holes.

While FFT has been long suggested as one possible sensing method [8], it suffers from spectrum leakage caused by the large side lobes in the frequency response of the filters that characterize each sub-carrier [9]. This results in significant inaccuracy and low dynamic range. On the other hand, the multi-taper method (MTM), suggested by Haykin as the optimal channel sensing method [10], comes at the expense of extra computational complexity. In our solution, we propose using filterbanks to sense the channel. By using filterbanks, the side lobes of the filters associated with each sub carrier can be made arbitrarily small by adjusting the filter length and other design specifications [3, 4]. The signal power of the output of the filterbank is used to estimate the signal spectrum.

In our system, we sense the channel 10 times per second. To sense the channel, each node halts its transmission for a specific amount of time, collects the necessary samples and runs the filterbanks sensing algorithm. The sensing information is next reported to the base station. The base station collects the sensing results from all the cognitive nodes, including itself, and broadcasts the compiled CSI results to all the leaf nodes in the network.

2.2 Transmitter

The software defined modem provides two modes of operation to process two different types of services. One service is a 19.2 kbps computer-to-computer data stream while the other service is a 16 kbps Continuously Variable Slope Delta Modulation (CVSD) vocoded voice. The data stream is encoded using a rate 1/2, constraint length 7, convolutional encoder. The encoder generates two outputs using the two generator polynomials $x^6 + x^5 + x^4 + x^3 + 1$ and $x^6 + x^4 + x^3 + x + 1$. The data stream, constructed from the outputs of the encoder, is interleaved using a simple row-column block interleaver. The output of the interleaver is divided into groups of three coded bits and each group is mapped to an 8-PSK constellation using Gray mapping.

For the voice stream, we use a Reed-Solomon (RS)(63, 51, t=6) encoder having a generator polynomial $p(x) = x^6 + x + 1$. This has the ability to correct up to 6 random byte errors and does not need to be followed by an interleaver. The encoded voice stream is divided into groups of two coded bits and mapped using a Gray code to a QPSK constellation.

To use only one digital upconverter from baseband to Intermediate Frequency (IF) for both services, the packet assembly is done such that the symbol rate at the input of the pulse shaping filter is 20 kbps for both data and voice streams. The transmitted packet consists of two major parts: a 192-sample cyclic preamble, generated using three identical Binary Phase Shift Key (BPSK) modulated pseudo noise (PN) sequences of length 64, and a payload constructed using the data or voice stream output of the symbol mappers.

Upconversion is done using Cascaded-Integrator-Comb (CIC) filters [11] and a novel pulse-shaping filter (PSF) whose coefficients are chosen to achieve the Nyquist-M property while at the same time compensating for the CIC passband drop [12]. This combined CIC and pulse shaper (CPSCIC), when compared with other methods available in the literature, has been proven to achieve less passband ripple, more stopband attenuation and less ISI. Moreover, combining the pulse-shaping filter and the CIC compensator filter allowed us to save a great amount of space on the hardware. The upconverted signal is finally modulated to Intermediate Frequencies (IF) for transmission over the channel.

2.3 Receiver

The received signal is first down-converted by means of a CPSCIC matched to its transmitter counterpart. The baseband signal is then passed to the synchronization and channel equalization modules, both of which implemented in the fractional space. The fractional spacing between the samples is chosen to be $T_s/2$, where T_s is the symbol interval. The portion of the receiver performing the synchronization is shown in Fig. 2.1.

The received signal $y(t)$ can be modeled as:

$$y(t) = x(t - \tau - (nT_s/2))e^{j2\pi((f+\Delta f_c)t+\theta)} \tag{2.1}$$

where *x(t)* is the transmitted signal, τ is the time offset, Δf_c is the carrier offset and θ is the phase offset.

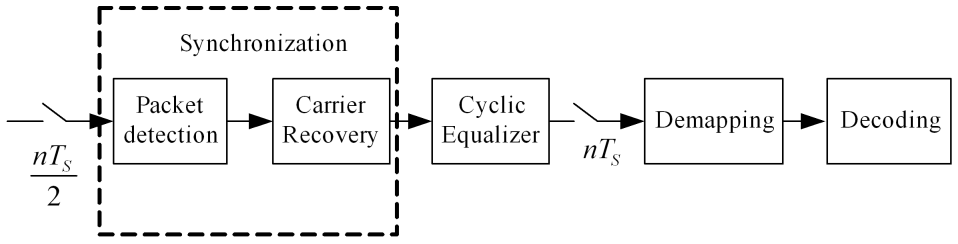

Figure 2.1: Receiver block diagram after digital down conversion.

We have ignored channel noise for brevity.

Synchronization is performed using a cyclic preamble. Cyclic preamble is chosen in our model because it can serve the dual purpose of estimating the timing and carrier offsets while at the same time equalizing the channel effects when coupled with a cyclic equalizer [13]. The repetition structure of the cyclic preamble allows us to detect the start of the packet as well as the carrier offset. This method exhibits good performance and is easy to implement. Packet detection is performed by computing the autocorrelation of the received signal. We correlate the signal with a shifted version of itself and find the position of the preamble by identifying the interval over which the autocorrelation is significantly large [5].

On the other hand, the carrier offset is estimated using the following equation, [5]:

$$\Delta f_c = \frac{\angle\left\{\sum_n x(n')\cdot x^*(n)\right\}}{\pi(N+1)T_s} \tag{2.2}$$

where the summation is over one cycle of the preamble, $\angle(\cdot)$ represents the phase angle in radians, and $*$ denoted the conjugate.

After compensating for the carrier offset, we make use of a fractionally-spaced adaptive equalizer to compensate for the channel distortion, any residual carrier offset and to obtain the correct timing phase [5]. The equalizer coefficients obtained using this algorithm are further fine-tuned using a decision-directed adaptive scheme. The half symbol-spaced cyclic equalizer is shown in Fig. 2.2.

In this figure, $p[n], n = 0, 1, \ldots, N$ represents a cycle of the preamble. The received signal is given by $\mathbf{y}_0 = [y[n], y[n-2], \ldots, y[n-2N]$. $\mathbf{w} = [w[o], w[1], \ldots, w[N]$ denotes the equalizer coefficient vector. The following equations describe the adaptation algorithm at step i.

$$e[i] = p[i \operatorname{mod} \mathrm{N}] - \mathbf{w}^{\mathrm{H}}[\mathrm{i}]\mathbf{y}_{\mathrm{i}} \tag{2.3}$$

$$\mathbf{w}[i+1] = \mathbf{w}[i] + 2\mu e^{\star}[i]\mathbf{y}_i \tag{2.4}$$

where H denotes the Hermitian operators, and $\mathbf{y}_i$ is a cyclically rotated version of $\mathbf{y}_0$, delayed by 2 samples for each shift. The adaptive algorithm minimizes the mean-square error denoted by $\|\mathbf{e}\|^2$. $\mathbf{e} = [e[0], e[1], \ldots, e[N]]^T$ is used to obtain the optimal equalizer coefficients $\mathbf{w}$. We choose N equal to 64.

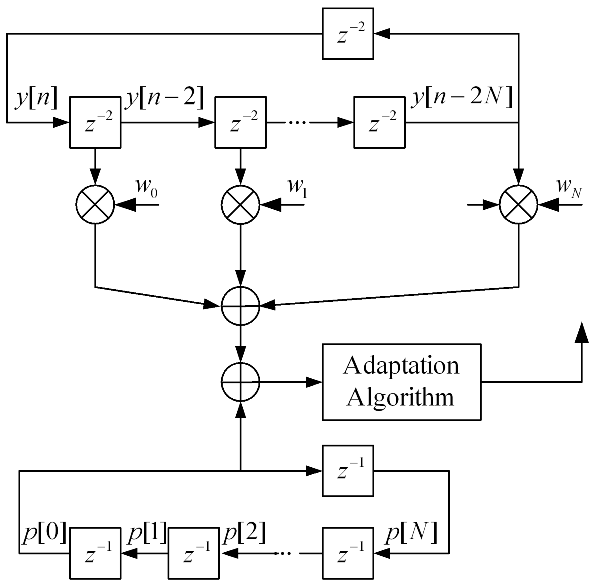

Figure 2.2: Half symbol-spaced cyclic equalizer

Fig. 2.3 depicts the eye diagram of the received signal after packet detection, carrier recovery and cyclic equalization, respectively. Synchronization was simulated in MATLAB and Simulink. The signal to noise (SNR) was set equal to 9 dB. The normalized carrier offset is assumed to be 0.001.

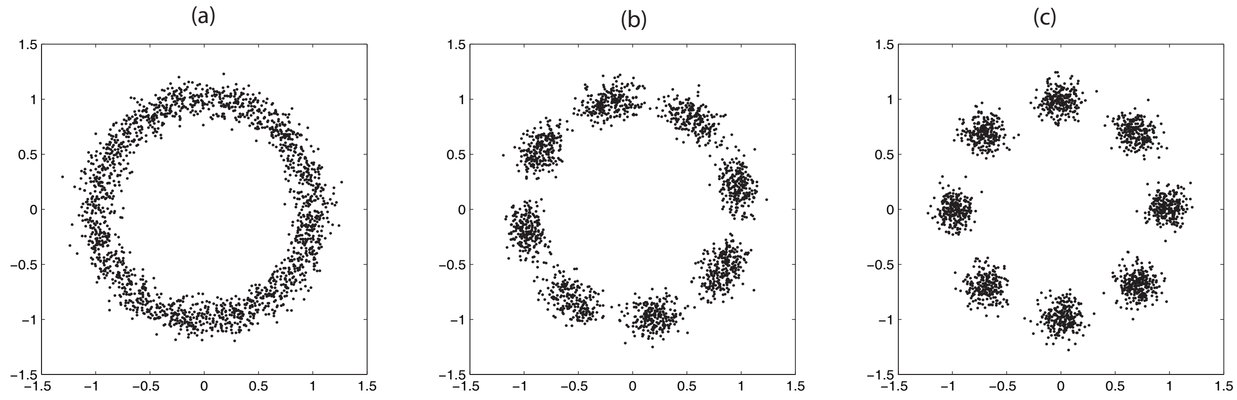

Figure 2.3: Eye diagram of the received signal (a) after packet detection (b) after carrier recovery (c) after equalization

2.4 MAC Layer

The MAC layer is designed to manage the medium access so as to coexist with primary users. Coexistence is the primary goal in this design as well as in other CR systems, such as IEEE802.22 [14]. Researchers implement the spectrum agility and required channel sensing paradigm to realize the cognition [15]. The medium access method, as defined in the smart radio challenge problem statement[2], is frequency-division multiple access (FDMA). Secondary users (SUs) use frequency-division duplexing (FDD) to communicate with each other. SUs resort to single carrier communication when data is available and conform to *quiet period* concept of IEEE802.22 standard.

Two control channels are used for coordinating sensing information, controlling leaf node communications and other management tasks. Our setup features uplink and downlink control channels. The uplink is based on Aloha while the downlink is a base station controlled broadcast channel used to broadcast sensing information and other management packets. We choose Aloha over carrier sense multiple access collision avoidance (CSMA/CA) because in the latter, channel sensing is not always possible; it is governed by the channel environment and the propagation delay. The base station receives and compiles the channel state data from all of the leaf nodes, including itself. The base station then broadcasts the compiled channel allocation table to all the leaf nodes. This allocation table is retained by the base station and the leaf nodes until the conclusion of the next sensing interval. Each cognitive node stops transmitting and then senses the entire channel periodically. If the current sensing information differs from the last compiled channel state information, following a random delay, the channel state data is transmitted by each leaf node to the base station. More detailed description of our MAC Layer design and implementation can be found in [1] and [7].

Chapter 3

Implementation and Results

3.1 System Layout

The cognitive radio modem was implemented on a Small Form Factor (SFF) Software Defined Radio (SDR) development platform provided by Lyrtech in collaboration with Texas Instruments (TI) and Xilinx. The SFF SDR is a self-contained platform consisting of three separate modules: the digital processing module, the data conversion module and the RF module.

The base of the platform is the digital processing module. It is designed around the TMS320DM6446 (also called DM6446) Digital Media Processor (DMP) System on Chip (SoC) [16] from TI and Virtex-IV XC4VX35 FPGA from Xilinx. DM6446 combines an Advanced Very Long Instruction Word (VLIW) 64x+ DSP and Reduced Instruction Set Computer (RISC) ARM926J-S cores, where the ARM micro-controller is mainly set to run the INTEGRITY Real-Time Operating System (RTOS) while DSP performs complex data processing. The data conversion module is equipped with a 125 MSPS, 14-bit dual channel ADC and a 500 MSPS 16-bit dual channel interpolating DAC provided by TI. The RF module is configured to have either 5 or 20 MHz bandwidth with working frequencies of 200-930 MHz for the transmitter and 30-928 MHz for the receiver. The SDR modem implementation is divided into different tasks each consisting of several modules. The SFF SDR platform gives the designer the option to choose the silicon device that is most suitable to the task being developed. We use the INTEGRITY and SMSHELL API provided by Lyrtech to target the board while we develop our signal processing tasks on the DSP core and the FPGA. The division of tasks between the DSP and the FPGA was made based on the availability of resources, the inherent characteristics of these cores, and the extra functionalities offered by TI and Xilinx. We make use of the already available Xilinx Logicore Blocksets for FPGA and the optimized DSP libraries written for vectors of complex numbers for C64x+ core. For instance, a Viterbi Algorithm was first written and optimized for the DSP, but considering the realtime required rate in the problem definition, the algorithm alone took 40% of the available time for processing. On the other hand, using the Viterbi decoder in the Xilinx Logicore performs the same task much faster. Traditionally, the algorithms that require fast and complex calculations but can be parallelized are best fit for the FPGA, while algorithms that require sequential analysis and decision making such as cognition and networking are usually implemented on the DSP. We note that it is hard to draw such a distinction between the functionalities of FPGA and DSP since DSPs are nowadays offering more pipelining and FPGAs are able to run sequential processing.

Interfacing between the DSP and FPGA is done using the Video Processing Sub-system (VPSS) data port. The DSP VPSS is a DM6446 DSP 16-bit synchronous video transfer port modified to support transfer of non-video data to and from the DSP. The VPSS consists of a Video Processing Front End (VPFE) and a Video Processing Back End (VPBE). The VPFE is used as an input interface to the DSP and the VPBE as an output interface from the DSP to FPGA. In the FPGA, a VPSS data port module, also consisting of a VPBE and a VPFE, is implemented to interface with the DSP VPSS. The data bus inside the FPGA is 32 bits and the VPSS of DM6446 DSP bus is 16 bits [6]. On the other hand, custom registers, a shared memory of eight 32-bit words between the DSP and the FPGA On-Chip Peripheral Bus (OPB), are used as configuration registers. As a result, the fast VPSS 32-bit bus is our gateway between DSP and FPGA.

The tasks developed for the FPGA are implemented using the System Generator for DSP. System Generator, an add-on to Simulink provided by Xilinx, produces a highly optimized FPGA realization, since each module used in the architecture maps to an FPGA library component that has been carefully constructed and optimized for the FPGA target device. Moreover, the System Generator provides us with a visual representation of the system that not only serves as the design specification, but as the behavioral simulation model and the source definition for the hardware. The System Generator implementation also facilitates the rapid investigation of various design options in the system [17, 18].

Development of the DSP side of the system involves generating C code. One way to do this is by using code generation capability for the already developed Simulink blocks. To elaborate, the algorithms targeting the DSP processor are first implemented in Simulink blocks. The Realtime Workshop (RTW) is next used to produce the first version of the code for the individual Simulink blocks. Each block is then individually tested in Simulink external simulation. Although RTW is able to generate stand-alone C code for the Simulink blocks, it can only be used for rapid prototyping and testing since the code it generates is not optimized for a specific DSP or GPP target. The RTW generated code often needs extra memory and processing power and the optimization burden is put on the compiler only. In addition keeping the design inline with RTW and simulink requirements requires extra effort and often results in redundant and complicated code. Furthermore RTW cannot be used to implement the complete DSP subsystem whose requirements include realtime performance in terms of memory, speed and special data alignment. To overcome these problems, a Target Language Compiler (TLC) file is developed to customize the code generation. In writing the TLC, it is feasible to use optimized TI DSP libraries DSPLib [19] and compiler optimization techniques such as giving feedback to the compiler. Furthermore, the wrapper TLC (unlike inline TLC) saves a single version of each algorithm and by reducing the redundancy it can simplify the code maintenance. The wrapper TLC code (written for the individual blocks) can also be reused in the independent compilation of the complete DSP subsystem project, without involving the RTW. Finally, the C code of the complete DSP subsystem (either generated by RTW or written as wrapper TLC or even generated elsewhere) is compiled by TI Code Composer Studio (CCS). CCS makes use of the high performance VelociTI architecture of DM6446 to optimize the code down to the programming level optimization (using the *-pm* switch). During the development progress, we decided not to use RTW because of the extra effort it requires and resort to the conventional IDEs such as CCS.

3.2 Subsystem integration

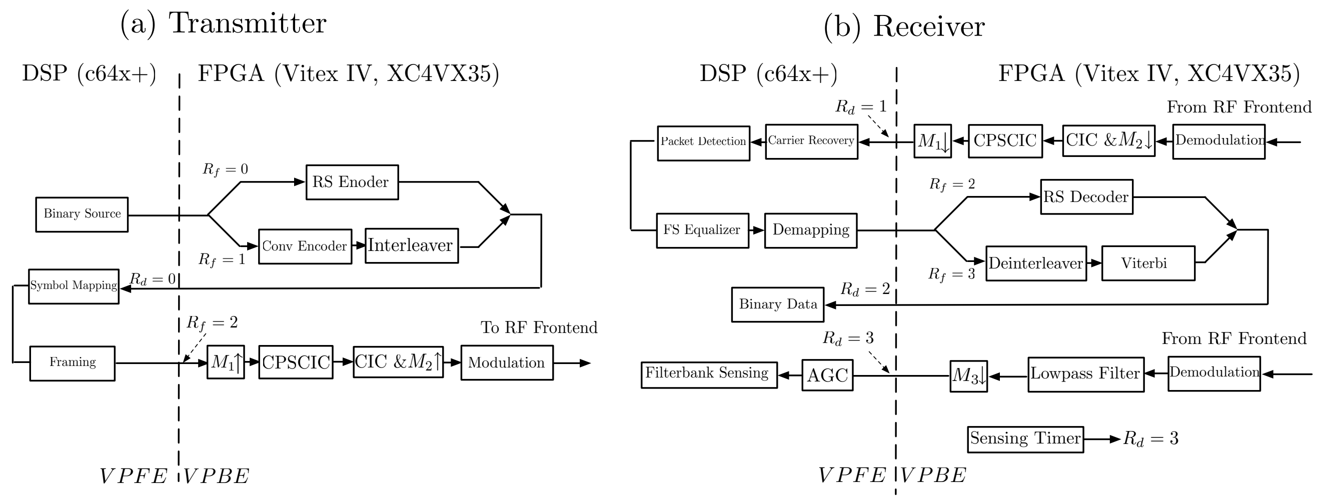

Figure 3.1: System data flow

The distribution of the SDR modem components between the DSP core and the FPGA is shown in Figures 3.1. The VPBE and VPFE are used to transfer the data streams back and forth between the two modules while the custom registers are used for handshaking.

Since FPGA is logic based, any changes we make to the values saved in the custom registers are monitored and responded inside the FPGA. We use the custom register R_f to indicate the services required by DSP to be done by FPGA. Depending on the type of data processing required from the FPGA, the DSP specifies a command number inside the custom register R_f and then transfers the data through the VPSS. Similarly, the custom register R_d is used by the FPGA to inform the DSP about the characteristics of the bit-stream arriving at the VPFE in the DSP subsystem. The DSP, being sequence based and often running an endless loop, uses the content of the R_d register to select the appropriate DSP function to be applied on the incoming data. The transmission is initiated, as shown in Figure 3.1, in the DSP core. The binary source is an arbitrary bitstream of voice or data incoming to the DSP. The voice data is input from the pcm3008 stereo audio codec at 48 KHz and encoded by the Continuously Variable Slope Delta Modulation (CVSD) having a data rate of 16kbps. The binary voice stream is zero padded to achieve a data rate of 20kbps. The data traffic, on the other hand, is a fixed computer-to-computer data stream. This data stream has a lower bit error rate than the voice stream and is retransmitted in the MAC layer (if missed or corrupted) to ensure data integrity of critical information.

The transmit packet is sent to the FPGA through the VPSS. The custom register R_f is set to zero if the source is audio and one if non-audio. In the FPGA, depending on the data content, either RS coding or convolutional coding combined with interleaving is performed. Following the source coding in the FPGA, the binary vectors are retransferred to the DSP ($R_d = 0$) to perform modulation and framing. 8-PSK modulation is used for the data stream while QPSK is used for voice as per the problem statement. The binary vector is finally sent back to the FPGA ($R_f = 2$) to upconvert the signal to intermediate frequency (IF) and eventually transmit it over the air. Digital up conversion in the FPGA consists of three blocks. A novel combined CIC and pulse shaper (CPSCIC) that follow the Nyquist-M criterion, a CIC integrator, and a Direct Digital Synthesizer (DDS) to modulate the signal to the IF frequency. The CPSCIC filter we used is a 80-tap FIR filter generated using the Xilinx FIRCOMPILER provided by the System Generator for DSP. The baseband signal was modulated to an IF frequency of 30 MHz at a sampling rate of 80 MHz. Note that in the SFF SDR platform, the FPGA has access to IO, Data Conversion (DConv) and RF modules.

At the receiver side, two separate functionalities are first performed in the FPGA, digital down conversion (DDC) and sensing. To downconvert the received signal to baseband, a CIC differentiator and a CPSCIC filter are used. The resulting signal is then passed to the DSP ($R_d = 1$) where synchronization tasks and symbol demapping are performed. After demapping, the signal is passed back to the FPGA for decoding ($R_f = 2$ for voice and $R_f = 3$ for data). The decoded signal is finally passed to the DSP ($R_d = 2$). Sensing is activated in the FPGA every 100 ms by means of a timer. This timer activates the sensing module 10 times a second for almost 4 μs. The timer control circuit disables the transmitter functionality while the sensing is performed. The sensing data is first demodulated to baseband by means of a DDS whose frequency is centered at the IF frequency. A lowpass polyphase decimator is used to filter the required signal band and bring the sampling rate down to 5 MHz. Note that in order to make use of the maximum dynamic range of the ADC, An automatic gain amplifier (AGC) is developed in the DSP to control the gain of the analogue amplifiers available on the data conversion module before the signal is digitized. The resulting signal is then sent to the DSP ($R_d = 3$) for further processing.

In the DSP, the DSPLib library for C64x+ is used for efficient implementation of the filterbank sensing. A filterbank is implemented in polyphase structure using 256 8-tap polyphase elements which are the decimated coefficients of the described prolate filter. The output of the polyphase elements are then passed through FFT. The output energy of the filterbanks is then averaged over three decimated samples. The sensing information is compared with a tunable threshold to locate possible active primary users and create a 32-byte channel state information. This information is then transmitted to the basestation. The basestation compiles the sensing information from all of the users and creates a common CSI to be used for channel assignment.

3.3 Software Development Progress

Using the progressive simulation based design and development, we start from simulation (in MATLAB) and substitute the models with real modules step by step to build a fully functional modem. The transition involves code generation for individual algorithms, debugging the algorithms individually and integrating them to build the DSP subsystem. The algorithms to be run on DSP subsystem (including coding, decoding, protocol stack, sensing and cognition) are developed in standard C, first using Microsoft Visual C (MSVC) and tested inside MATLAB (as MATLAB executable MEX files) against other MATLAB simulation of the modem. Basic algorithm optimization for speed and integrity tests are performed in this step using debugging features of MSVC integrated with MATLAB engine. In the next step to migrate the code to the board, the code is compiled in Code Composer Studio (CCS) to build the output file for DSP. In this step we made use of the compiler specific switches for optimization and also compiler directives to save a single version of the developed algorithm among CCS and MSVC (and possibly Realtime Workshop TLC) for easier maintenance and code reuse. Finally the individual modules of the code (running on the SFF SDR) are tested inside MATLAB. Basically the debugging and code evaluation phase takes place in MATLAB environment and starts from uploading the CCS generated output file to the SFF SDR board. The functionality of each algorithm and the amount of time it takes is measured to be consistent with the realtime requirements of the system.

The interface between the platform and MATLAB is developed using MEX files and SMSHELL API [20] provided by Lyrtech for the SFF SDR platforms. Modules of our system are migrated to the board step by step. For each block of the system, the input is generated using MATLAB and passed to the board using the developed MEX function which writes into a predefined memory addresses in the DSP using SMSHELL and the network socket. The results of the algorithm and the values of variables at the intermediate stages of the algorithm are monitored in MATLAB by reading the predefined memory locations of the end result inside DSP. A protocol is implemented to synchronize the transaction between the host (running MATLAB) and SFF SDR by continually polling a flag in DSP memory.

After the first modem is implemented, to develop the network of the cognitive radio modems (while only one SFF SDR board was present at the time), we deploy the available real modem along with the simulated primary and secondary users in a central simulation [7] in DEVSJava [21] and MATLAB. To simulate the effect of the primary users on real modems, a vector signal generator is used to emulate the transmitted signal of licensed users on the channel. The signal generator is capable of transmitting an arbitrary sequence of length 10000 samples and is easily interfaced with MATLAB, which gives us the ability to program it in a way to simulate the behavior of primary network considering the probability distribution of channel access of PUs. In a scenario, two secondary users are using a channel when a PU enters the spectrum and SUs change the frequency band accordingly to avoid interference. This is the feature which will be demonstrated in the upcoming 2007 SDR technical conference at Denver. The bit error rate of the primary user before channel switching of SUs should be minimized, while the average discovery of spectral holes should be maximized applying opportunistic channel access.

In the hardware in the loop (HIL) simulation, we try different schemes for channel sensing, cognition and transceiver functionality of the modem before the second and third modems join the cognitive network by replacing the simulated modems. When using HIL for network simulation, it is easier for the host - the machine that runs the simulator- to simulate a complete transceiver instead of individual internal modules inside the transceiver. This simulated modem takes parameters such as packet generation distribution and channel characteristics as inputs and reuses the MATLAB simulation code to test the real modem in a semi-realistic environment. During this step we shifted from Realtime Workshop development tool to a combination of CCS, MATLAB and DEVSJava (for HIL network simulation).

The event based nature of the radio network makes Discrete Event System Specification (DEVS) a suitable environment with enough power and more efficiency than time based modeling. Availability of the DEVSJava [21] package (written in Java as a portable language with many features including easy concurrency) made DEVSJava the natural choice for use as a starting point. DEVS is a theoretical modeling which can describe modular systems in hierarchical manner. Based on the formal definition of DEVS, it is able to model both time based and event based systems. One characteristic of practical real-time systems is the ability to work with concurrent real objects, having the outputs ready by a deadline and the ability of decision making based on computational processing units. Having that in mind, the real-time simulator objects in DEVSJava are implemented as concurrent *threads.* In addition, while in MATLAB programming we can generate arbitrary waveforms and pass it to the board, we do not have the capabilities of DEVSJava for event based simulation. The MATLAB functionalities in communications and signal processing are brought inside Java environment using MATLAB Builder for Java. MATLAB Builder for Java, can build Java classes from the MATLAB functions and have them ready to use in DEVSJava. The MATLAB m-files that were already developed for MATLAB simulation of the channel, the MATLAB visualization methods, and also the hardware-interfacing MEX files are compiled to Java classes that are called later inside our DEVSJava simulation. The integration of DEVS with MATLAB gives us the opportunity of decoupling the model and underlying (often very complicated) math. While it is possible to reuse the code available in MATLAB (including the MATLAB simulation codes), the event based nature of DEVSJava (adding the concept of time to the objects) adds to the power of normal object oriented Java language. In addition, the message passing structure of DEVS models fits naturally into the category of networking. As a result we can benefit from a well-defined simulation engine which is an alternate to some other very expensive solutions.

We recently received two other functioning SFF SDR boards and are integrating them into the system. The new design thus puts less burden on the vector signal generator (for substituting missing radios) and rather connects the SFF SDR boards by streaming data between them while the Signal Generator emulates only the Primary Network. To move forward to build a larger network we are now working on MAC layer implementation considering the requirements of IEEE802.22 such as quiet period (during which al SUs should stop transmission to sense the SU-free channel) and Listen-Before-Talk policy [22] (band-of-interest channel sensing).

3.4 Measurement Results

We have used a vector signal generator to generate a signal that emulates the effect of the primary users on the channel. 10000 samples are generated in MATLAB and are used as input to the signal generator. The signal generator modulates the signal and transmits over the frequency range 462 MHz to 467 MHz. Six sinewaves are added and then passed to the function generator. The function generator modulates these sinewaves to 463.278MHz, 463.367MHz, 463.456MHz, 463.545MHz, 463.624MHz, and 463.985MHz. The sinwaves at 463.278MHz, 463.456MHz, and 463.624MHz are transmitted at 25dbm power. The ones at 463.367MHz, and 463.545MHz are transmitted at -15dbm. 463.985MHz sinewave is transmitted at -9dbm.

The Power Spectral Density (PSD) of the received signal is presented in Fig. 3.2. 2048 samples are used for the filterbank with prototype filter of length 2048. The results of FFT, FFT with a Hanning window and filterbank are averaged over three decimated sample. The calculated PSD from 462MHz to 467MHz for these three methods are depicted in Fig. 3.2. As we can see filterbank sensing is able to show all the transmitted sinewaves clearly. FFT, on the other hand, has a considerable spectrum leakage which results in missing the three sinewaves at 463.367MHz, 463.545, and 463.985MHz. FFT with hanning while having better dynamic range than FFT, also misses two of the sinewaves.

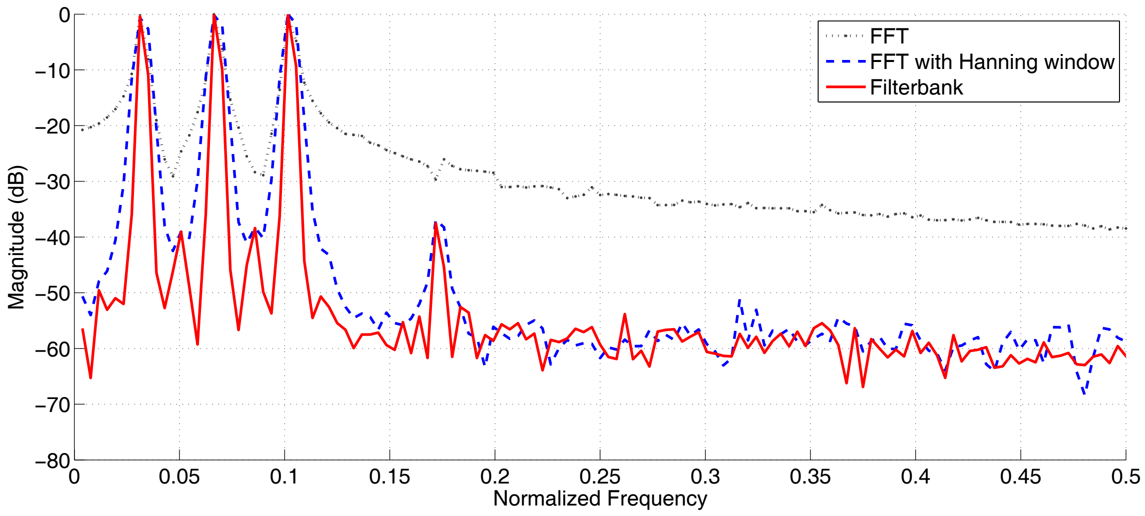

Figure 3.2: Power Spectral Density (PSD) normalized measurements by FFT, FFT with hanning window, and filterbank

These measurement results show that filterbank sensing method has less spectrum leakage than FFT, as predicted in [3, 4]. Therefore, unlike the FFT method, we are able to detect the presence of a low power PU when they are located at adjacent low and high power frequency bands.

Chapter 4

Progress Time Line

During the 10 months of the second phase of the Smart Radio Challenge, the system defined in our proposal was developed through software simulation and hardware implementation. Our timeline for the implementation is summarized in table 4.1. As can be seen, the timeline is divided into three stages: From December to February, March to June, and July to September.
In the first stage, we worked on the physical layer simulation in MATLAB and the MAC layer simulation in DEVSJava. We also tried to familiarize ourselves with DSP core and FPGA by reading the catalogues and different tutorials. We then started to move some of our physical layer code from MATLAB to C. We used MATLAB MEX files to develope different modules in C and then tested them in MATLAB.

In the second stage, upon receiving some of the development software, we started working on developing some code to target the DSP core and FPGA. Different modules such as channel coder/decoder, voice codec, and polyphase filtering were developed in this stage. We received one functional SFF SDR EVM on May 8th. This module was a baseband module with the first version of Lyrtech software. We started to familiarize ourselves with the hardware and the Lyrtech libraries at this stage.

We received the first complete module with RF frontend on July 2nd which signaled the beginning of stage three of the implementation. We downloaded the developed code to the hardware and started testing our system. We implemented the sensing and cognition parts of our system on DSP and FPGA by using a vector signal generator to emulate traffic of primary users. However, we were not able to test our complete system because only one complete board was available to us. We received two SFF SDR platforms without RF frontend at the beginning of September. We received the RF Frontends for those modules on September 25th. Now, we are working on putting the system together on the three boards.

Table 4.1: Timeline of the progress made during the phase 2 of the project

Dates	**Tasks**	**Participants**
Stage 1 -	**Software simulation** **Hardware preparation**	
12/02/2006 to 02/31/2007	Basic physical layer simulation in Matlab - Channel Coding/Decoding - Baseband Modulation/Demodulation - Framing/Packet Detection - Filterbank based Spectrum Sensing MAC and cross layer simulation of a cognitive radio network - Simulation using DEVSJava Documentation of software simulation Hardware preparation - Familiarizing with the different software packages - Developing C code in Matlab MEX files	Peiman Amini Ehsan Azarnasab Salam Akoum Xuehong Mao Shafagh Abbasi
Stage 2 -	**Developing code for the hardware** **Software simulation (con't)**	
03/01/2007 to 07/1/2007	Advanced physical layer simulation in Matlab - Synchronization (timing/carrier/phase recovery) - Preamble investigation Developing spectrum sensing algorithm in VHDL Physical layer simulation and implementation in C MAC layer simulation - Distributed Channel Sensing in DEVSJava **One Baseband module arrived on May8th** - Familiarization with the hardware - Implementation of some DSP and FPGA Code	Peiman Amini Ehsan Azarnasab Salam Akoum Xuehong Mao Harsha Rao
Stage 3-	**Hardware implementation** **Research paper**	
07/02/2007 - 09/25/2007	**Arrival of one complete SFF SDR on July2nd** Software Simulation - RF/IF Modulation - Fractionally Spaced Cyclic Equalization - Physical layer Simulation in Simulink Hardware Implementation - On-board tesing of developed C code - DUC/DDC implementation on the FPGA - Spectrum Sensing implementation on FPGA and DSP - Interfacing Matlab and Java to SFF SDR Documentation - Two submitted papers to the SDR Conference - Final technical report **Two complete SFF SDR is available to us on September 25**	Peiman Amini Ehsan Azarnasab Salam Akoum Xuehong Mao Harsha Rao

Chapter 5

Issues and Considerations

5.1 Limitations

Various difficulties were encountered as we progressed through the project. These include:

- **Delivery of hardware:**

  The main limitation that we faced in our project was the delay in the delivery of the hardware necessary to conduct our experiments. Three functional SFF SDR boards were supposed to arrive in January, but due to unforeseeable reasons we received the first functional baseband evaluation module on May 8th. The first complete SFF SDR platform arrived at the beginning of July. Two more boards arrived in September. We received the RF front-ends of the latter two on September 25th. As a result of this delay, our initial workplan had to be changed and intensive simulation work was done to model different parts of the physical layer of the modem. Furthermore, to be able to test each part of the system separately, we resorted to the Hardware in the Loop technique with one SFF SDR platform.

- **Changes in the team members**:

  Throughout the project duration, the team has witnessed changes in its members. David Palchak, one of the team members with hardware experience, graduated and left the team in February. He was replaced by Ehsan Azarnasab who started to work on the challenge in March. Scott Talbot and Shafagh Abbasi left the team in November and May respectively. Harsha Rao joined our team in March, he left for an internship in June and rejoined the team in August. This variation in the team composition was one of the major limitations we had to deal with. We had to constantly reassign tasks to accommodate for these modifications.

- **Revision of the Design**:

  We expected to be able to use Real Time Workshop (RTW) for our development on the DSP c64x+. However, we realized that, while RTW is a powerful tool for testing single modules and generating their codes, it was not able to provide the real-time requirement of most of the modules in our system. Therefore, we had to use some of the code that was developed using RTW and combine it with some other optimized code written in Code Composer Studio. The final code was then compiled in Code Composer Studio.

5.2 Use of sponsor materials

In this section, we provide a description of the software packages and hardware platform we used for our implementation. These tools were made available by several sponsoring companies including Lyrtech, Texas Instruments (TI), Xilinx, Mathworks, Prismtech, Zeligsoft, Greenhills, and Synplicity.

While we familiarized ourselves with most of the tools we have received, we were not able to use some of them in our phase II implementation because of the delay in delivering the hardware and some of the software packages.

Our system design is now mainly implemented in TI's CCS and Xilinx's ISE and System Generator for DSP. Mathworks' MATLAB and Simulink were used in the simulation part of the design. We further employed Real Time Workshop to generate the code for individual modules in the system. Lyrtech provided us with DSP API, FPGA components, Board Support Package (BSP), SMSHELL Library to interface the SFFSDR platform to the host and the Model Based Design Kit (MBDK).

GreenHills Multi and Integrity are being used to develop some applications over the ARM9 core. We have familiarized ourselves with Prismtech's SPECTRA and Zeligsoft's CE and are going to use them in phase III of the project to develop SCA compliant APIs. A new addition to our lab is currently working on developing code for the FPGA using the Synplicity evaluation module.

5.3 Testing equipment

The equipment used for development and testing include:

1. An Agilent ESG Vector Signal Generator.
2. An Agilent E4407B spectrum analyzer.
3. A Tektronix TDS 224 4-channel digital real-time oscilloscope.
4. Two BK precision 4040A 20MHz sweep/function generators.
5. An Agilent 2 G samples/sec oscilloscope.

Chapter 6

Deliverables and Future Work

6.1 Deliverables

Completion of the project will include delivery of the following items prior to the established deadline.

6.1.1 System

1. An over-the-air demonstration of a functional cognitive radio node which can sense the channel, find the spectrum holes, and transmit on them. Furthermore, the cognitive radio will be able to move to a new frequency band when a primary user is using the band.

2. A simulated demonstration of a functional system with our maximum system size of 15 leaf nodes, one base station, and 4 Primary Users. Both demonstrations will show the following minimum capabilities:

   - Channel sensing
   - MAC handshaking
   - Leaf node communication via both data and voice
   - SU channel switching following appearance of a Primary User

Software

1. A detailed functional simulation of the entire system, including RF transmission effects such as fading and Doppler shift.

2. A functional cognitive radio modem simulated in MATLAB as well as Simulink.

3. Code generated using system generator for DSP to implement the FPGA blocks.

4. Code generated using Simulink along with Real Time Workshop to develop the individual modules for the DSP.

5. VHDL source code for all FPGA components, unless prohibited by third-party licenses.

6. DSP source code for all components, unless prohibited by third-party licenses.
7. All unit tests and test data sets.
8. All integration test routines and data sets.
9. Complete data sets used in system demonstrations.

Documentation

1. One page monthly status reports have been submitted to the SDR Forum. These reports have included the following details:
    - Progress to date
    - Development issues that have arisen, including design changes
    - Proposed resolution to issues, including justification for design changes
2. A test plan to demonstrate the capabilities of our system design was submitted to the SDR Forum in May 2007. At the time of writing the test plan, no hardware in functional condition had been delivered to us. Hence the test plan was on a purely system design level and assumed the hypothetical availability of two SDR SFFs.
3. The work has culminated in two papers [23, 7] titled "Implementation of a Cognitive Radio Modem" and "Hardware in the loop: a development strategy for software defined radios". We will be presenting these papers at the 2007 SDR Technical Conference and Product Exposition.
4. Finally, our work has also been documented in this report. It includes the description of the final design and implementation to the component level. For each component, the report includes:
    - A description of the final design.
    - A description of the implementation, including hardware dependent optimizations.
    - An evaluation of the optimality of the component in perspective to its relative significance or complexity.

6.2 Status of the Project and Future Work

For the past two months, we have been testing our design on one complete SFF SDR board which along with the simulated cognitive nodes act as a network. We received two more complete boards in late September and are currently working to develop all the functionalities required for our cognitive network. After implementing our network, we plan to use the available wireless testbed of the Flux Group [24] as a primary user network and study the effect of different types of primary user traffics on our cognitive radio as well as the effect of possible interference from the cognitive network on the legacy networks. Our proximity with the Flux group provides us with this exciting opportunity.

After finishing the implementation of the base-station centric cognitive radio network [1], in the phase III of the challenge we will investigate the possibility of implementing an ad hoc network for the 2008 Smart Radio Challenge. To this end, we will employ ad hoc on-demand routing algorithms and distributed sensing to implement an ad hoc cognitive network for first responders in disaster scenario. Thanks to the availability of hardware and the expertise in the team, we will be able to move forward to implement a complete functional ad hoc network.

One new master student, Pooyan Amini, has recently joined our team with expertise in algorithm and hardware development. Furthermore, we have involved two undergraduate students namely Daryl L. Wasden and Arash Farhang. These students are learning the development tools and will be able to contribute to the 2008 Smart Radio Challenge.

Bibliography

[1] P. Amini, D. Palchak, X. Mao, S. Talbot, S. Akoum, and S. Abbasi, "Smart radio challenge proposal: Spectrum access for first responders," Smart Radio Challenge, Available at `http://www.ece.utah.edu/~pamini/proposal.pdf`, Tech. Rep., September 2006.

[2] "Smart radio challenge," *http://www.radiochallenge.org/*.

[3] P. Amini, R. Kempter, R. R. Chen, L. Lin, and B. Farhang-Boroujeny, "Filter bank multitone: A physical layer candidate for cognitive radios," *2005 Software Defined Radio Technical Conference*, November 2005.

[4] P. Amini, R. Kempter, and B. Farhang-Boroujeny, "A comparison of alternative filterbank multicarrier methods in cognitive radio systems," *2006 Software Defined Radio Technical Conference*, November 2006.

[5] B. Farhang-Boroujeny, *Signal Processing Techniques for Software Radio.* available at `http://www.ece.utah.edu/~farhang`, 2007.

[6] "Lyrtech SFF SDR development platform technical specs," Lyrtech Inc., Available at `http://www.lyrtech.com/ publications/sff_sdr_dev_platform_en.pdf`, Tech. Rep., February 2007.

[7] E. Azarnasab, P. Amini, and B. Farhang-Boroujeny, "Hardware in the loop, a developement strategy for software defined radios," *Software Defined Radio Conference*, 2007.

[8] T. Weiss and F. Jondral, "Spectrum pooling: and innovative strategy for the enhancement of spectrum efficiency," *IEEE Communications Magazine*, vol. 42, no. 3, pp. S8–S14, March 2004.

[9] T. Weiss, J. Hillenbrand, A. Krohn, and F. Jondral, "Mutual interference in ofdm-based spectrum pooling systems," *IEEE 59th Vehicular Technology Conference, VTC 2004-Spring*, vol. 4, pp. 1873–1877, May 17-19 2004.

[10] S. Haykin, "Cognitive radio: Brain-empowered wireless communications," *IEEE Journal on Selected Areas in Communications*, February 2005.

[11] E. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Transactions on Acoustic, Speach, and Signal Processing*, vol. 29, April 1981.

[12] S. Talbot and B. Farhang-Bouroujney, "Pulse shape filter design in digital modems employing CIC filters," *SDR Forum Technical Conference*, November 2006.

[13] B. Farhang-Boroujeny, "Cyclic equalization options in software-based radios," *SDR Technical Conference*, November 2007.

[14] The IEEE LAN/MAN Standards Committee, "802.22 wg on wireless regional area networks (WRANs)," *http://www.ieee802.org/22*.

[15] H. Kim and K. G. Shin, "Efficient Discovery of Spectrum Opportunities with MAC-Layer Sensing in Cognitive Radio Networks," *accepted to IEEE TRANSACTIONS ON MOBILE COMPUTING.*

[16] "TMS320DM6446 digital media system-on-chip," Lyrtech Inc., Available at `http://focus.ti.com/docs/prod /folders/print/tms320dm6446.html`, Tech. Rep., March 2007.

[17] C. Dick, F. Harris, and M. Rice, "Synchronization in software radios - carrier and timing recovery using fpgas," *In Proceedings of the IEEE symposium on Field-Programmable Custom Computing Machines*, 2000.

[18] "System generator for dsp reference guide," Xilinx Inc., Tech. Rep., August 2007.

[19] "TMS320C64x+ DSP Big-Endian Library Programmer's Reference," Texas Instrument, Tech. Rep., March 2006.

[20] "Lyrtech SMSHELL technical reference guide," Lyrtech Inc., Tech. Rep., February 2007.

[21] B. P. Zeigler and H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models*, jan 2005.

[22] C.-T. Chou, "Adaptive quality-of-service provisioning in wireless/mobile networks," Ph.D. dissertation, The University of Michigan, 2004.

[23] P. Amini, E. Azarnasab, S. Akoum, X. Mao, H. I. Rao, and B. Farhang-Boroujeny, "Implementation of a cognitive radio modem," *2007 Software Defined Radio Technical Conference*, November 2007.

[24] "Emulab network simulation testbed," *http://www.emulab.net/*.