THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

Interference reduction with relay beacon in cognitive radio networks

ARNAB KUMAR BANIK
Fall 2010

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Dr. Sven G. Bilén
Associate Professor of Engineering Design,
Electrical Engineering, and Aerospace Engineering
Thesis Supervisor

Dr. Jeffrey Louis Schiano
Associate Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The existing static frequency allocation policy, controlled by the Federal Communications Commission (FCC) in the United States, is fast approaching an apparent spectral crisis, owing to consistent and rapid increase in spectrum demand by wireless users. In such an inevitable spectrum scarcity scenario, opportunistic use of the idle spectrum bands that are otherwise licensed to primary users is one key solution approach. Cognitive radio (CR) arises to be a tempting solution to this spectral congestion. The purpose of a cognitive radio network is to efficiently detect spectrum holes so that secondary transmissions can optimally use the spectrum band without interfering with primary users.

Misdetection of spectrum holes, owing to signal fading and path loss, however, will cause in concurrent transmission with primary users resulting in undesirable interference that violates the very basic philosophy of cognitive network transmission. Therefore, it is important to study how this interference level relates to design parameters such as the beacon detection threshold, and how it affects the primary users' performance. Also, to improve spectrum sensing and to reduce interference, it is of key interest to formulate how a cooperative scheme of relay transmission of beacon signals suitably addresses the effect of signal fading and path loss, which often results in misdetection of spectrum holes.

This work deals with the effect of interference owing to concurrent transmission by both primary and secondary user for the case of failed detection of a beacon transmitted by primary user. In such a scenario, beside theoretical study of the interference phenomena, we experiment with it by transmitting data from both primary and secondary user at the same frequency to allow channel interference. To avoid this, we introduce a beacon signal by the primary user to notify the secondary users to transmit data at a different frequency in the available frequency band. Further, that signal might still be lost due to fading, so to avoid this case we introduce a relaying method in which the secondary user retransmits the beacon to the second secondary user, hence minimizing interference even more. All the results were carried out with software-defined radio using the Universal Software Radio Peripheral and GNU radio platform in LINUX. It uses a number of GNU Radio modules along with other Python and C-Shell scripts giving a working baseline structure for a cognitive radio.

Contents

List of Figures

List of Tables

Acknowledgments

This thesis was made possible by the great support from those around me. I would like to thank my thesis advisor, Dr. Sven G. Bilén, for the inspiration to take on this project and for the continued support to see it through. Your critical guidance all along the project work and your thorough review of the manuscript made this work possible to his level. I also thank Dr. Jeffrey Louis Schiano, my honors advisor, who has been instrumental in my success at Penn State offering guidance at every step during my undergraduate career. My special thanks to Dr. Gary Weisel, Professor of Physics and Dr. Pinaki Das, Professor of Mathematics both at Penn State, Altoona for their encouragement, support and guidance throughout my undergraduate study and particularly during my study at Altoona during the first two years of my undergraduate study. I owe many thanks to Matt Sunderland, who helped me in defining codes for the software models of receive and transmission techniques at various points during this project. I could not have accomplished this without your support.

Most importantly, I would like to thank my parents, whose constant encouragement, support and guidance helped me achieve my education to this level.

Chapter 1

INTRODUCTION

1.1 Motivation

Access to the wireless spectrum in the United States is controlled by the Federal Communications Commission (FCC) and most of the frequency bands useful to wireless communication ("RF bands") have already been licensed by the FCC. The other few available unlicensed bands, such as ISM (Industrial, scientific and medical) band, for example, are also fast filling up. But because of the continuous increase in demand for the radio spectrum, this static allocation policy is fast facing a spectral crisis.

To address this spectral crisis, the FCC published a report prepared by its Spectrum Policy Task Force (SPTF) [1]. The report recommends certain rules and regulations for the efficient use of radio spectrum and methods for improving the usage of existing spectrum. With respect to spectrum utilization, this report illustrates that there is significantly more inefficient utilization of spectrum rather than actual spectrum scarcity due to legacy systems and the rules imposed by FCC. Most of the allotted radio frequency (RF) and are not in use most of the time; some are partially occupied, while others are heavily used. Recent measurements have shown that for as much as 90% of the time, large portions of the licensed bands remain idle (Figure 1).

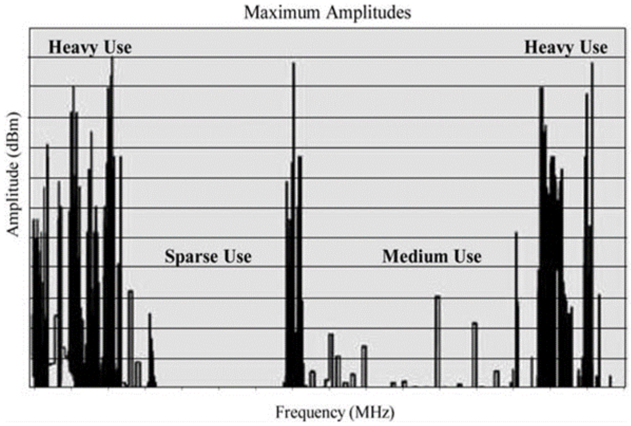


Figure 1: Spectrum usage showing highly uneven nature [1]

A new communication philosophy is necessary in order to utilize the existing wireless spectrum optimally and in an intelligent manner. Newly emerging cognitive radio (CR) technology [2] is envisaged to solve problems in wireless networks resulting from the limited available spectrum and the inefficiency in the spectrum usage.

One aspect of CR is for a radio transceiver intelligently detect which communication channels are in use and which are not, and to instantly move into vacant channels while avoiding occupied ones. This optimizes the use of available RF spectrum while minimizing interference to the users.

1.2 Overview

In the inevitable scenario of spectrum scarcity, opportunistic use of the idle spectral bands that are otherwise licensed to primary users is one key solution. Cognitive radio, as introduced by Mitola [2] is a promising solution to this spectral congestion. A CR system efficiently detects spectrum holes so that secondary transmissions can optimally use the spectrum band without interfering with primary users. Thus, a CR can be described as an intelligent radio that is capable of determining its frequencies of action and adapting to the changing frequency usage within that frequency band. In other words, if a radio is capable of setting and configuring its own parameters including "waveform, protocol, transmitting and receiving carrier frequency and networking" autonomously, it can be called a cognitive radio.

The innovation that makes CR possible is the technology of software defined radio (SDR) in which the "software meets the antenna" [3]. In this thesis project, the property of CR in which it is able to sense the frequencies of transmissions in its surrounding environment, has been implemented using GNU Radio and he USRP platform. GNU Radio is an open-source SDR programming platform that works closely with the Universal Software Radio Peripheral (USRP, pronounced "U-Surp"), a hardware platform designed for GNU Radio [4].

1.3 Objective

The purpose of this work is to test the hypothesis given below and to design a system in which secondary users can detect a beacon signal from primary user and then scan for an alternative frequency (i.e., to locate spectrum hole) in the given FM band for transmission. In addition, we include signal relaying by the secondary user (one that is closer to the primary user) to effect retransmission of the beacon signal to another secondary user.

The hypothesis follows like this. We first transmit an mp3 file from one computer to another acting as primary user. After that, we introduce two more users as secondary users both transmitting at the same frequency as that of primary user and, since all the three users will be transmitting at the same time and frequency, we expect that there will be some interference in the receiving end. To detect the interference we would encounter some data overlap in the primary receiver from the secondary users.

To resolve this interference issue, we will introduce a beacon transmitted from the primary user to notify the secondary users. Therefore, when the secondary user detects the beacon signals, they are informed as to if the primary user is in an active or in an idle state. Once the secondary user detects the beacon, it will then look for another frequency by scanning in the given band so that it can identify and transmit at another available frequency rather than the specified frequency already in use by primary user. However, when due to signal loss, the secondary users fail to detect the beacon from primary user; it may assume the signal to be idle at that time and will transmit the same signal concurrently causing the interference. Such situation can be verified when one the secondary user, which we expect to be the one closer to primary user, will detect the beacon and transmit at another frequency, and whereas the other secondary user, which is further away, will fail to receive the beacon and therefore transmit in the same frequency. To resolve this issue, we will try to re-transmit the beacon signal from the nearest secondary user to the other (relay beacon), and if this relay process works out then the second secondary user will be able to receive the beacon from primary user. This process can be tested and verified in the experiment and we can test if under this case both the secondary user scans for an alternative frequency to transmit than the original one used by primary user.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 discusses the principles and architecture of software-defined radio, the impact and extent of interference from a cognitive user on a primary user, and the operation of GNU Radio along with its hardware (USRP) and software (Python language) elements that represent the working blocks of research experiments. In Chapter 3, the experimental setup of the hardware component explained. In Chapter 4, the results from experimental tests using GNU Radio and USRP are demonstrated and analyzed. The thesis is then follows with conclusions and future work in Chapter 5. Appendices A and B provide the MATLAB files and Python scripts used in the work.

Chapter 2

BACKGROUND

2.1 Software-Defined Radio

2.1.1 Definition

A software-defined radio (SDR), as the name implies, is a radio system that makes use of software in the processing of communication signals. The process has the distinct advantage of reducing the number of hardware components in a radio system. Additionally the SDR is more flexible and changeable in contrast to a hardware radio which has fixed characteristics. Thus an SDR is a radio system in which some or all of the physical layer functions are software defined. The software in an SDR determines the specifications of the radio and what it does. If the software within the radio is changed, its performance and function may change.

In general, an SDR, therefore, can be implemented on a generic hardware platform consisting of digital signal processing (DSP) processors as well as general purpose processors and/or field programmable gate arrays (FPGAs). These processing that occurs in these elements may be modulation and demodulation, filtering (including bandwidth changes), and other functions such as frequency selection and, if required, frequency hopping.

Based on the level of radio re-configurability, an SDR is categorized by one of the tiers given below: [5]

- Tier-0: A non-configurable hardware radio, i.e. one that cannot be changed by software.
- Tier-1: An SDR where limited functions are controllable. These may be power levels, interconnections, etc. but not mode or frequency.
- Tier-2: In this tier, a significant proportion of the radio (frequency, modulation and waveform generation/ detection, wide/narrow band operation, security, etc.) is software configurable. The RF front end still remains hardware based and non-reconfigurable.
- Tier-3: The ideal software radio or ISR in which the boundary between configurable and non-configurable elements exists very close to the antenna and the “front end” is configurable. It can be said to have full programmability.
- Tier-4: The ultimate software radio or USR is a stage beyond that of an ISR. Not only does this form of SDR have full programmability, but it is also able to support a broad range of functions and frequencies at the same time.

Software radios have significant utility for the military and cell phone services, both of which must serve a wide variety of changing radio protocols in real time.

In the long term, SDRs are expected by proponents like the Wireless Innovation Forum (formerly the SDR Forum) to become the dominant technology in radio communications. SDRs, along with software-defined antennas are the enablers of the cognitive radio.

2.1.2 General Architecture of SDR

As defined in Tier-3 [5], the idea behind ideal ISR lies in the fact that the boundary between configurable and non-configurable elements exists very close to the antenna, and the "front end" is configurable Therefore, the ISR would consist of an antenna with its signal being sampled by an ADC and the rest done in software (Figure 2). Significant amounts of signal processing are channeled to the general-purpose processor (GPP), rather than being done in special-purpose hardware. Such a design produces a radio that can receive and transmit widely different radio protocols (sometimes referred to as waveforms) based solely on the software used.

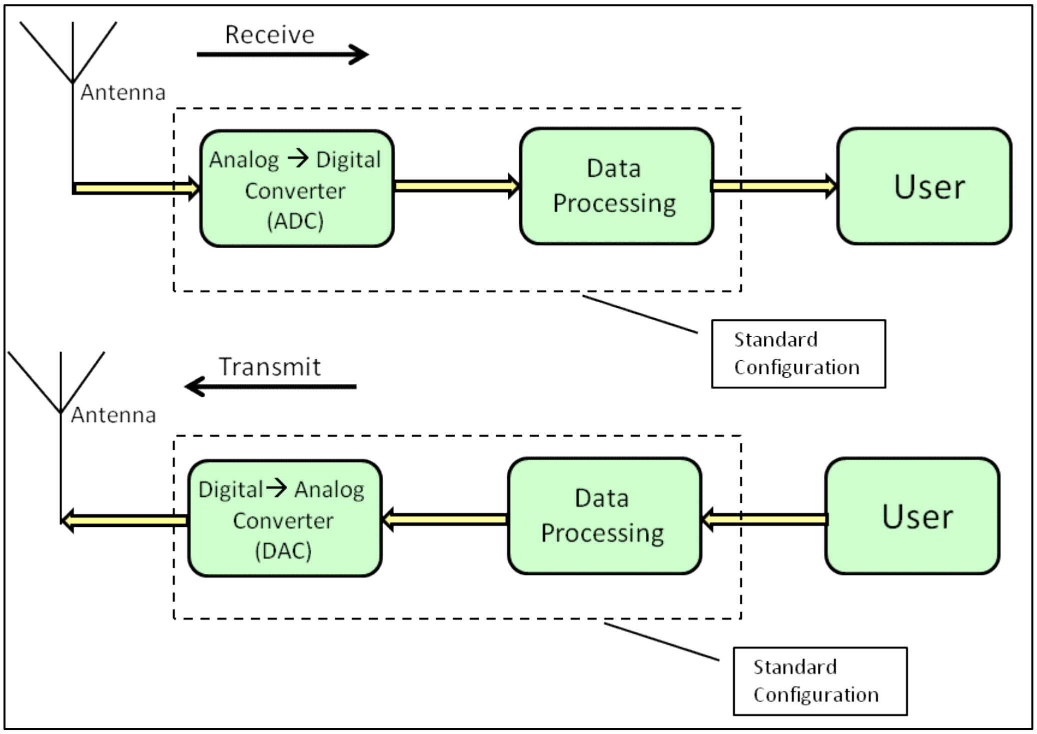


Figure 2: Block diagram of an Ideal SDR

In reality, however, there is no hardware currently available that can implement transmission or reception without some RF front-end hardware. Additionally, a single antenna cannot receive the entire radio spectrum; therefore, to demodulate signals at higher frequencies we need extra hardware to properly compute and sample the signal [6]. Based on these constraints, a practical SDR is more like that shown in the block diagram of Figure 3. This SDR system is divided into four main sections, i.e., RF section, IF section, baseband section and data section [7]

RF Section: This block consists of antenna and RF front-end. The antenna covers the spectrum linked to entire range of operation for the purpose of both transmission and reception of the signal whereas RF front-end is responsible for tuning, detection, signal transmission and analog up conversion (for transmission from IF) or analog down conversion (to IF after reception).

The major challenge to the RF front end comes from the increased bandwidth of new signals. With increased bandwidth the radio becomes more vulnerable to interference.

IF Section: The section plays most important role in digital radio because it is where analog-to-digital conversion (ADC) of the down-converted IF signal for reception and digital to analog conversion (DAC) for transmission takes place. This block also carries out digital up-conversion (DUC) or digital down-conversion (DDC) to make high frequency data compatible with available computer resources. It also performs the most important function of modulation and demodulation.

Baseband section: The baseband section deals with operations linked to security protocols, correlation etc.

Data section: This section provides instructions that the baseband section and DUC/DDC used to carry. This section acts as an interface (such as to a PC) to program and develop intelligent controls and different routines for the SDR system.

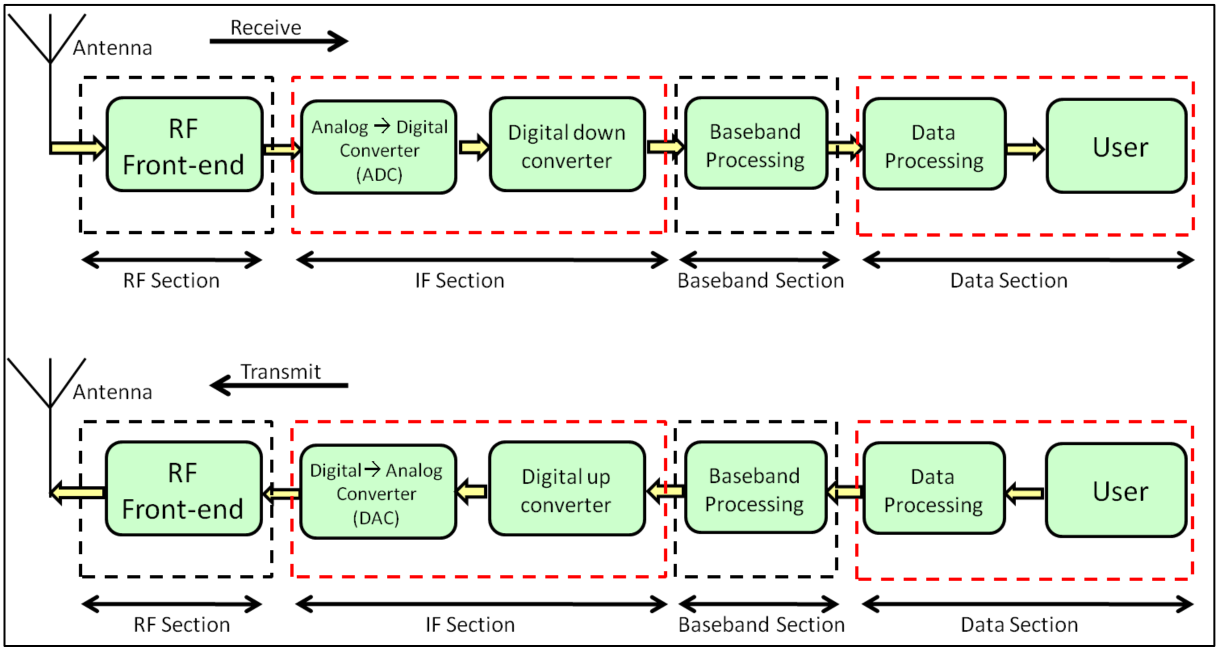


Figure 3: Block diagram of a real or practical SDR

2.1.3 Open Source SDR Projects

GNU Radio is an open source [7] SDR project that was started about ten years ago by Eric Blossom, an electrical engineer. The main idea behind this project, as its founder says, was to turn all the hardware problems into software problems, that is to move the complexity of a radio equipment from the hardware to the software level, and get the software as close to the antenna as possible.

GNU Radio is a free software development toolkit and its complete source code is available online. The source code allows development of a custom non-commercial radio receiver by combining and interconnecting appropriate software blocks, as if they were functional blocks. Each module is able to perform a specific signal processing function to support implementation of oscilloscope, concurrent multichannel receiver and an ever-growing collection of modulators and demodulators.

Ettus Research, LLC (recently purchased by National Instruments) is the key provider of hardware components for GNU Radio. The minimal hardware components required to work with GNU Radio is offered by the USRP platform. This USRP performs almost everything that a GNU Radio is capable of doing. GNU Radio and its hardware and software functionalities are discussed in greater detail in Chapter-3.

2.2 Cognitive Networks: Models and Design

Interference analysis in cognitive radio system has been studied by a number of authors [8–10]. Taking into consideration a simple model of a realistic network, the impact of interference by varying certain key parameters is worth analyzing in a set up containing cognitive users. In one such cognitive network model with a single primary user and multiple CR users, as modeled by Vu *et al.* [11], the impact of interference and its upper bound are mathematically derived. By way of background, we present study here those mathematical relationships as analyzed by Vu *et al.* [11].

2.2.1 Model Principle

In a network with beacon, the primary users transmit a beacon before each transmission [11]. This beacon is received by all users in the network. The cognitive users, upon detecting this beacon, will abstain from transmitting for some duration. The mechanism is designed to avoid interference from the cognitive users to the primary users. In practice, however, because of channel fading, the cognitive users may sometimes miss-detect the beacon. They could then transmit concurrently with the primary users, creating interference. Therefore, it is of interest to know how this interference level relates to design parameters, such as the beacon detection threshold, and how it affects the primary users' performance.

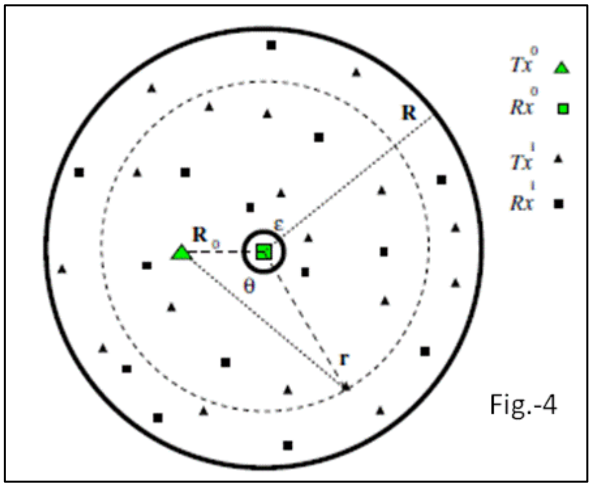


Figure 4: Network model [11]

2.2.2 Model

For interference analysis, the model used a constant density of cognitive users. The interference power can be derived as a function of the beacon detection threshold, the cognitive user density and transmit power. Subject to random fading and random cognitive user locations, this interference is random. The mathematical derivation thus provides closed-form upper bounds on the mean and variance of this interference power.

Network model:

The following planar network is used. (Figure 4)

1. Single primary users (PU) and *'n'* number of cognitive users (CU)
2. PU (receiver) is at the center and transmitting PU is at distance R_0 from the PU (receiver).
3. The following notations are used.
    - T_x^0 and R_x^0 ≡ distance of PU transmitter and receiver
    - T_x^i and R_x^i ≡ distance of CU transmitter and receiver
    - Subscript *'0'* for primary and *'i'* for CU where i= 1, 2, 3 … n for n CUs
    - h_0, h_i → channel received by R_x^0 (primary receiver)
    - g_i → channel received by CU (i) from T_x^0 (primary transmitter)
4. *'n'* number of CU have surrounded PU with a uniform density λ.
5. Thus, network area with radius R increases with n, hence $\lambda = n/\pi R^2$
6. To limit interference all CT (cognitive transmitters) are ε distance away from primary receivers

Channel Model:

The following are mathematical relations of wireless channel with path loss and fading.

From path loss and fading in wireless channel, the composite channel is given by

$$h = \frac{A}{d^{\alpha/2}}\tilde{h}, \tag{1}$$

where d is the distance between transmitter and receiver;

α is power path loss and $\tilde{h}$ is the small scale fading factor (it is a complex circular Gaussian (0,1) RV); and

'A' is a frequency dependent constant which is taken as '1' here for simplicity.

Signal Model:

Based on model philosophy, each cognitive user has an active factor of β, which is the probability that the user is actively transmitting. To avoid interference to the primary receiver, the cognitive users listen to a beacon, which the primary transmitter sends before its own transmission. Upon receiving this beacon, the cognitive users will remain silent for the duration of the primary transmission. The following are the computations:

Let β = active factor of CU = probability of CU actively transmitting = *k*/*n*, where *k* is no of active CU, *n* = total CU

If x_b is beacon signal and $y_{i,b}$ is the beacon received by CU at R_x^i, then

$$y_{i,b} = g_i x_b + z_i \tag{2}$$

where z_i is white Gaussian noise with power σ^2.

So the received power at CU (*i*) is given by

$$P_{r,b}^i = \frac{P_b \mid g_i \mid^2}{\sigma^2} \tag{3}$$

If CU needs to receive a minimum threshold power P_{th} to detect beacon,

then the Probability that CU(*i*) will miss a beacon is

$$q_i = \Pr[P_{r,b}^i < P_{th}] = \Pr[\mid g_i \mid^2 < \frac{P_{th}\sigma^2}{P_b}$$

Now putting,

$$\gamma = \frac{P_{th}\sigma^2}{P_b} \tag{4}$$

The equation becomes

$$q_i = \Pr[\mid g_i \mid^2 < \gamma]) \tag{5}$$

Here, γ is dimensionless as noise power is normalized with path loss constant *A*.

Now if CU(*i*) misses beacon, it will transmit concurrently with PU

Taking x_i as the signal from CU(*i*) and x_0 from primary transmitter, we have the received signal y_0 at Rx_0 given by

$$y_0 = h_0 x_0 + \sum_{i=1}^{n} F_i h_i x_i + z_0 \tag{6}$$

Where F_i is indicator function defined as

$$F_i = \begin{cases} 1 & \text{with probability} \quad \beta q_i \\ 0 & \text{with probability} \quad 1-\beta q_i \end{cases} \tag{7}$$

and F_i is independent and $z_0 \sim N(0,\sigma^2)$.

Also x_i s are independent with mean = 0 and power P as CU do not cooperate.

So, total interference power from CUs is given by

$$I_0 = F_i \,|\, h_i \,|^2\, P \tag{8}$$

where I_0 is also RV and the rate achieved by PU is

$$C_0 = \log\left(1 + \frac{|\,h_0\,|^2\, P_0}{I_0 + \sigma^2}\right) \tag{9}$$

Since the interference power I_0 is random, this capacity is also random. So the outage probability for given rate threshold T can be defined as:

$$P_e = \Pr[C_0 \le T] \tag{10}$$

Based on this, the interference power I_0 and its effect on the primary outage probability, or in other words, the mean and variance of interference power I_0 as $n \rightarrow \infty$ can be studied.

Interference from the Cognitive Users:

The step now is to establish the interference to the receiver from a cognitive transmitter at a radius r ($\varepsilon \le r \le R$) and at an angle θ to the line connecting the primary transmitter and receiver, as in Figure 4.

Now, at a distance r, the density function given by $f_r(r) = \dfrac{2\pi r\lambda}{\pi(R^2 - \varepsilon^2)\lambda}$

$$\rightarrow f_{r_i}(r) = \frac{2r}{(R^2 - \varepsilon^2)}, \qquad \text{with } \varepsilon \le r \le R \tag{11}$$

and the distribution of θ is uniform between 0 and 2π.

Assuming the probability that this cognitive user misses the beacon is q_i, then from (1), the channel between the primary transmitter and this cognitive user is

$$g_i = \frac{\tilde{g}_i}{d(r,\theta)^{\alpha/2}} \tag{12}$$

where

$$d(r,\theta) = (r^2 + R_0{}^2 - 2rR_0 \cos\theta)^{1/2} \tag{13}$$

So the probability of missing the beacon (5) becomes

$$q_i = \Pr[|g_i|^2 < \gamma] = \Pr[|\tilde{g}_i|^2 < \gamma d^{\alpha}(r,\theta)] \tag{14}$$

With $\hat{g}_i$ being zero-mean circularly complex Gaussian, $2|\hat{g}_i|^2$ is a chi-square random variable with two degrees of freedom with the pdf e^{-z}, hence $\Pr[|\tilde{g}_i|^2 \le \gamma_0] = 1 - e^{-y_0}$.

Analogically, the missing beacon probability can be explicitly calculated as

$$q_i = 1 - e^{\gamma d^{\alpha}(r,\theta)} \tag{15}$$

When missing the beacon, the cognitive transmitter may transmit with probability *β*. The channel from this cognitive transmitter to the primary receiver is

$$|h_i|^2 = r^{-\alpha} |\tilde{h}_i|^2 \tag{16}$$

Since the cognitive transmitters are independent, from (8), (15) and (16), the total interference power from cognitive users can be written as

$$I_0 = P\beta \sum_{i=1}^{n} q_i r_i^{-\alpha} |\tilde{h}_i|^2 \tag{17}$$

where

$$n = \lambda\pi(R^2 - \varepsilon^2)$$

$$q_i = 1 - e^{\gamma d^{\alpha}(r,\theta)}$$

$$d(r_i,\theta_i) = (r_i^2 + R_0{}^2 - 2r_i R_0 \cos\theta_i)^{1/2}$$

$$r_i \sim f_{r_i}(r) = \frac{2r}{(R^2 - \varepsilon^2)}, \varepsilon \le r \le R \qquad \text{i.i.d.}$$

$$\theta_i \sim U[0,2\pi] \quad \text{i.i.d.}$$

$$h_i \sim N(0,1) \quad \text{i.i.d.}$$

and $\alpha, \beta, \lambda, R$ are constants

(i.i.d. → independent identically distributed RV)
By putting

$$I_i = q_i r_i^{-\alpha} \, |\, \tilde{h}_i \,|^2 \tag{18}$$

then I_i are i.i.d. and

$$I_0 = P\beta \sum_{i=1}^{n} I_i \tag{19}$$

2.2.3 An Upper Bound on the Interference

Now just ignoring subscript '*i*' for simplicity and denoting

$$d = (r_i^2 + R_0{}^2 - 2 r_i R_0 \cos\theta_i)^{1/2}$$

$$I = (1 - e^{\gamma d^{\alpha}}) r^{-\alpha} \, |\, \tilde{h} \,|^2$$

Now an upper bound may correspond to increasing r to $r + R_0$. With this distance, it is more likely that CU will be beaconless. This will increase interference with PU.

So

$$I \leq (1 - e^{\gamma (r + R_0)^{\alpha}}) r^{-\alpha} \, |\, \tilde{h} \,|^2 \tag{20}$$

2.2.4 Mean and Variance of Interference Power (I)

With

$$I_0 = P\beta \sum I_i$$

$$\rightarrow E[I_0] = P\beta * E[\sum I_i] \rightarrow E[I_0] = P\beta * nE[I]$$

$$\rightarrow E[I_0] = P\beta\pi(R^2 - \varepsilon^2) E[I] \tag{21}$$

Similarly,

$$E[I_0{}^2] = P^2\beta^2 * E[\sum I_i{}^2] \rightarrow E[I_0{}^2] = P^2\beta^2 * nE[I^2]$$

$$\rightarrow E[I_0{}^2] = P^2\beta^2\pi(R^2 - \varepsilon^2) E[I^2]$$

So

$$\operatorname{var}[I_0] = E[I_0{}^2] - E[I_0]^2 = P^2\beta^2\pi(R^2 - \varepsilon^2)[E[I^2] - E[I]^2] \tag{22}$$

Now $E[I]$ and $E[I^2]$ can be computed:

With

$$E[I] = \int_{\varepsilon}^{R} | \, f(r)dr. f(\theta)d\theta \, , \; E[\tilde{h}^2] = 1 , \; E[\tilde{h}^4] = 3 \text{ and } \int_{0}^{2\pi} f(\theta)d\theta = 1$$

$$E[I] \le 2\int_{\varepsilon}^{R} (1 - e^{\gamma(r+R_0)^{\alpha}}) \frac{r^{1-\alpha}}{R^2 - \varepsilon^2} dr \tag{23}$$

$$E[I^2] \le 6\int_{\varepsilon}^{R} (1 - e^{\gamma(r+R_0)^{\alpha}}) \frac{r^{1-2\alpha}}{R^2 - \varepsilon^2} dr \tag{24}$$

2.2.5 Upper Bound on the Mean Interference Power

As

$$1 - \alpha < 0 \Rightarrow r^{1-\alpha} \ge (r + R_0)^{1-\alpha}$$

Hence:

$$E[I] \le \int_{\varepsilon}^{R} \frac{2(r^{1-\alpha} - e^{-\gamma(r+R_0)^{\alpha}} (r + R_0)^{1-\alpha})}{R^2 - \varepsilon^2} dr$$

The above relation follows from this inequality conditions:

$$1 - \alpha < 0 \rightarrow \alpha - 1 > 0$$

$$\Rightarrow r \le r + R_0 \Rightarrow r^{\alpha-1} \le (r + R_0)^{\alpha-1} \Rightarrow \frac{1}{r^{\alpha-1}} \ge \frac{1}{(r+R_0)^{\alpha-1}}$$

$$\Rightarrow r^{1-\alpha} \ge (r + R_0)^{1-\alpha}$$

$$\Rightarrow -r^{1-\alpha} \le -(r + R_0)^{1-\alpha} \Rightarrow -e^{-\gamma(r+R_0)^{\alpha}} r^{1-\alpha} \le -e^{-\gamma(r+R_0)^{\alpha}} (r + R_0)^{1-\alpha}$$

$$\Rightarrow r^{1-\alpha} - e^{(-\gamma(r+R_0)^{\alpha}} r^{1-\alpha} \le r^{1-\alpha} - e^{-\gamma(r+R_0)^{\alpha}} (r + R_0)^{1-\alpha}$$

$$\Rightarrow E(I) \le \int_{\varepsilon}^{R} \frac{2(r^{1-\alpha} - e^{-\gamma(r+R_0)^{\alpha}} r^{1-\alpha})}{R^2 - \varepsilon^2} dr \le \int_{\varepsilon}^{R} \frac{2(r^{1-\alpha} - e^{-\gamma(r+R_0)^{\alpha}} (r + R_0)^{1-\alpha})}{R^2 - \varepsilon^2} dr$$

This leads to the equation:

$$E[I] \le \int_{\varepsilon}^{R} \frac{2(r^{1-\alpha} - e^{-\gamma(r+R_0)^{\alpha}} (r + R_0)^{1-\alpha})}{R^2 - \varepsilon^2} dr$$

This bound can be interpreted as

Interference = when CU always transmits – when CU received beacon

The above bound corresponds to slightly reducing the latter portion by increasing the distance to the primary Tx from the cognitive user when that user *receives* the beacon. Hence, the interference from this user when missing the beacon will be slightly increased.

This bound can be evaluated in closed forms using the incomplete Gamma function. Specifically, we have an explicit upper bound for the average interference power $E[I_0]$ in (25). With an infinite number of cognitive users, $R \to \infty$, this bound approaches a limit as in (26).

$$E[I_0] \le \frac{2\pi\beta P}{\alpha-2}\Big[\frac{1}{\varepsilon^{\alpha-2}} - \frac{1}{R^{\alpha-2}} - e^{-\gamma(\varepsilon+R_0)^{\alpha}}(\varepsilon+R_0)^{2-\alpha} + e^{-\gamma(R+R_0)^{\alpha}}(R+R_0)^{2-\alpha} + \gamma^{\frac{\alpha-2}{\alpha}}\{\Gamma(\frac{2}{\alpha},\gamma(\varepsilon+R_0)^{\alpha}) - \Gamma(\frac{2}{\alpha},\gamma(R+R_0)^{\alpha})\}\Big] \tag{25}$$

$$E[I_0] \le \frac{2\pi\beta P}{\alpha-2}\Big[\frac{1}{\varepsilon^{\alpha-2}} - e^{-\gamma(\varepsilon+R_0)^{\alpha}}(\varepsilon+R_0)^{2-\alpha} + \gamma^{\frac{\alpha-2}{\alpha}}\{\Gamma(\frac{2}{\alpha},\gamma(\varepsilon+R_0)^{\alpha})\}\Big] \tag{26}$$

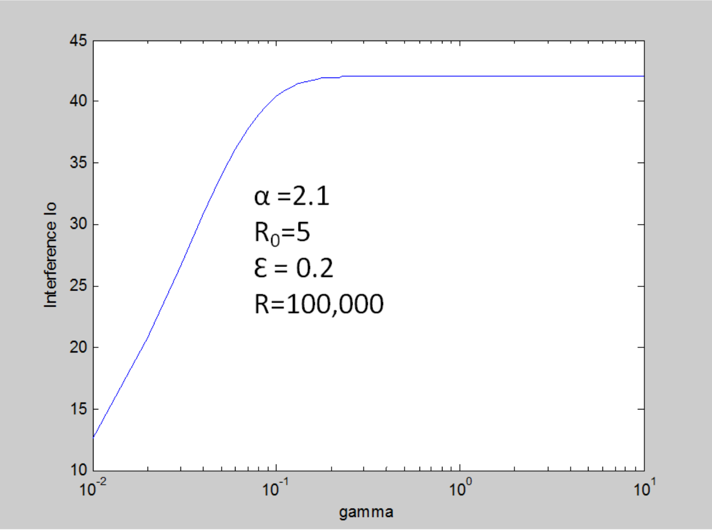


Figure 5: Beacon detection threshold vs. upper bound of mean interference

The graphs of $E(I_0)$ are plotted with the following parameters: (Figure 5)

$\alpha = 2.1$, $R_0 = 5$; $\varepsilon = 0.2$; R = Very Large (~100,000) with all other parameters are normalized to unity. The plot (Figure 6) below results. The response of the plot is explained below.

If γ increases (more beacon misses)

→ Increase in average I

→ The finite U. Bound value at $\gamma \to \infty$ reaches early at a finite value of γ

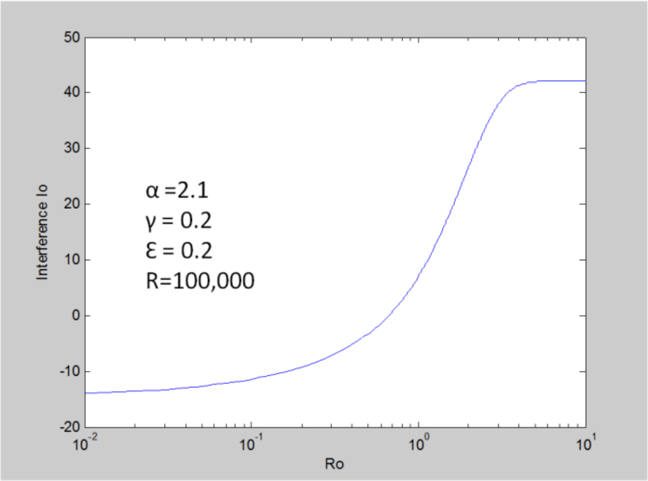


Figure 6: Tx-Rx distance (R_0) vs. upper bound of mean interference

Reference Figure 6 illustrates the following:

- For smaller R_0, most CU will receive beacon and interference will be less.
- If R_0 increases (more beacon misses)

  →An increase in average *I* will result.
- The finite U. Bound value reaches when R_0 is large.

  This maximum upper bound reaches when CR misses all beacons and with further increase of R_0 the upper bound will remain constant as all CR still remains beaconless

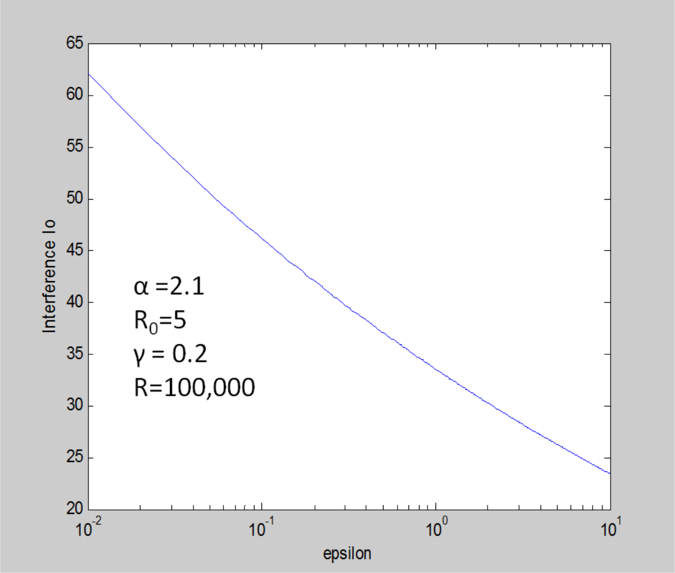


Figure 7: Receiver guard radius (*ε*) vs. upper bound of mean interference

Reference Figure 7 for ε vs. upper bound of $E[I_0]$

- For smaller ε, more CU will involve and Interference will be more.
- If ε increases (less CU in operation- asymptotically decrease)

→Decrease in average I → to value zero

2.2.6 Upper Bound on the Variance of Interference Power

As

$$(R+R_0)^\alpha \geq r^\alpha + R_0^{\ \alpha}$$

→ $E[I^2]$ in previous equation can be further bounded as per equation below

$$E[I^2] \leq \frac{6}{R^2-\varepsilon^2}\int_\varepsilon^R (r^{1-2\alpha} - 2e^{-\gamma(\varepsilon+R_0)^\alpha}(r+R_0)^{1-2\alpha} + e^{-2\gamma R_0^{\ \alpha}} e^{-2\gamma r^\alpha} r^{1-2\alpha})dr \tag{27}$$

$$E[I^2] \leq \frac{6}{R^2-\varepsilon^2}[\frac{\varepsilon^{-2(\alpha-1)} - R^{-2(\alpha-1)}}{2(\alpha-1)} - 2F(\alpha, 1-2\alpha, \gamma, \varepsilon+R_0, R+R_0) - e^{-2\alpha R_0^{\ \alpha}} F(\alpha, 1-2\alpha, 2\gamma, \varepsilon, R)]$$

Where function 'F' is defined as follows:

If $\beta+1 \geq 0$, then

$$F(\alpha,\beta,\gamma,u,v) = \frac{\gamma^{-(\beta+1)/\alpha}}{\alpha}[\Gamma(\frac{\beta+1}{\alpha}, \gamma u^\alpha) - \Gamma(\frac{\beta+1}{\alpha}, \gamma v^\alpha)]$$

And if $\beta+1 \leq 0$, then

$$F(\alpha,\beta,\gamma,u,v) = \frac{1}{\beta+1}(e^{-\gamma v^\alpha} v^{\beta+1} - e^{-\gamma u^\alpha} u^{\beta+1}) + \tfrac{\alpha\gamma}{\beta+1} F(\alpha, \beta+\alpha, \gamma, u, v)$$

As $R_0 \rightarrow \infty$, $\mathrm{var}(I_0)$ depends only on $E[I^2]$ and not $E[I]$.

→ The upper bound of $\mathrm{var}(I)$ is given by:

$$\mathrm{var}(I_0) \leq 6P^2\beta^2\lambda\pi[\frac{1}{2(\alpha-1)}\frac{1}{\varepsilon^{2(\alpha-1)}} - G(\alpha,\gamma,\varepsilon+R_0) + \frac{1}{2}e^{-2\gamma R_0^{\ \alpha}} G(\alpha, 2\gamma, \varepsilon)] \tag{28}$$

where 'G' function is defined as below:

$$G(\alpha,\gamma,x) = \frac{1}{\alpha-1}e^{-\gamma x^\alpha} x^{-2(\alpha-1)} - \frac{a\gamma}{(\alpha-1)(\alpha-2)} e^{-\gamma x^\alpha} x^{-2-\alpha} + \frac{\alpha\gamma^{2-2/\alpha}}{(\alpha-1)(\alpha-2)}\Gamma(\frac{2}{\alpha}, \gamma x^\alpha) \tag{29}$$

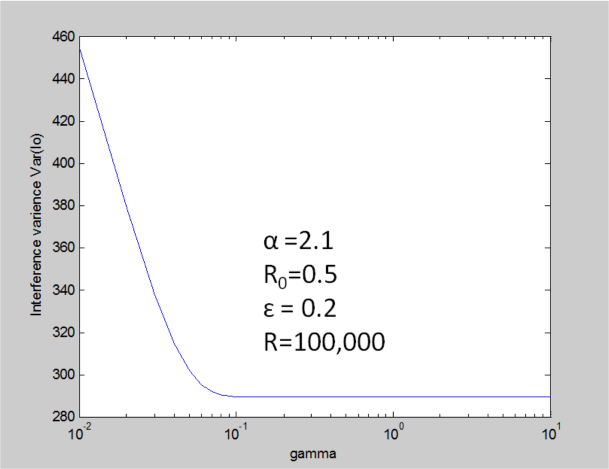


Figure 8: Beacon detection threshold (*γ*) vs. upper bound of variance of interference

Inference variance from reference Figure 8 (above):

Again similar plot for variance are analyzed with the following parameters:

α = 2.1, R_0 = 5; ε = 0.2; and other parameters normalized to unity. The plots and their response are explained below.

If γ increases (more beacon misses)

→ Decrease in variance [*I*] and to a min value → [opposite response of average *I*, Figure 8]

→ Higher the sensitive of beacon (smaller γ) → smaller will be *I* and higher the variance of *I*.

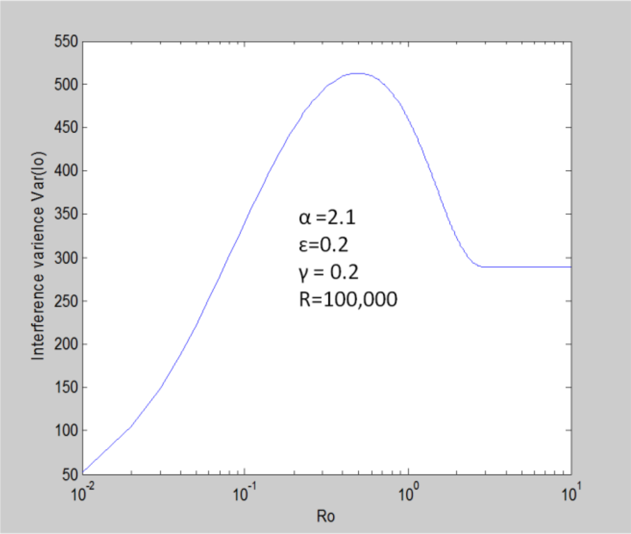


Figure 9: Primary Tx-Rx distance vs. upper bound of variance of interference

Reference Figure 9 (above):

- If R_0 increases → more beacon misses

→Results in increase in variance of *I* to a local max at a critical R_0

- After certain threshold R_0 → Variance of *I* remains constant.

Reference Figure 10 below:

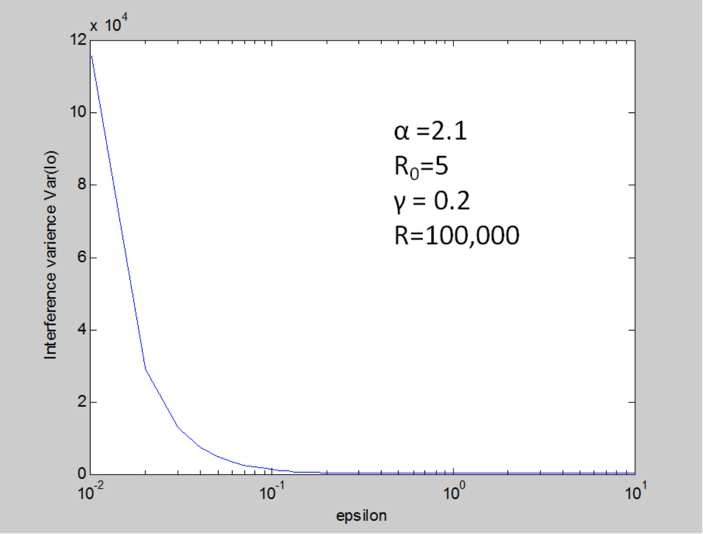


Figure 10: Receiver guard radius (ε) vs. upper bound of variance in interference

- For smaller ε, more CU will involve more and variance in interference will be greater.

Increase in receiver guard radius ε

→Results in fast decrease in variance of *I*

→ Fast approach to zero

These analyses resulted in formulating the interference power as a function of the beacon threshold, the number of cognitive users, the primary and cognitive transmit powers, the distance between the primary transmitter and receiver, and the receiver protected radius. The relation gives closed-form upper bounds on the mean and the variance of this interference power, both with a finite number of cognitive users and in the limit as this number goes to infinity. The mathematical derivation of upper bounds as derived in the original paper [11] offer an analytical understanding of how the interference behaves according to various network parameters. Interference analysis in such networks will be important in understanding the interaction among the users and in designing their algorithms.

2.3 GNU Radio:

GNU Radio is an open-source software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using readily-available, low-cost RF hardware and processors. It is widely used in hobbyist, academic and commercial environments to support wireless communications research as well as to implement real-world radio systems.

Eric Blossom, together with his development colleague Matt Ettus, have realized this project which can turn an ordinary PC into a good quality radio receiver; the only additional hardware required are a “low-cost” RF tuner and an analog-to-digital converter to convert the received signal into digital samples.

The open-source software development toolkit in GNU Radio allows us to develop a custom non commercial radio receiver by combining and interconnecting appropriate software modules, which are independent functional blocks [12]. The package currently includes about 100 modules, but others can be added to the initial library. Each module is able to perform a specific signal processing function (for example mixer, phase locked loop, a filter), with a real-time behavior and with high-throughput. AS most processing is done on the GPP (General purpose processor), a late model PC with enough processing capability and memory were used.

With the GNU Radio approach, the designer is a software developer who builds the radio by creating a graph (in a similar way to what happens in the graph theory) where the vertices are signal processing blocks and the edges represent the data flow between them. The signal processing blocks are normally implemented in C++, whereas the graph structure is defined in Python.

GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ using processor floating point extensions where available. Thus, the developer is able to implement real-time, high-throughput radio systems in a simple-to-use, rapid-application-development environment. GNU Radio is used either to implement real and working radio equipments, or just for research in the area of wireless communication and transmission. GNU Radio software modules support various modulations (e.g., GMSK, PSK, QAM, OFDM), error corrections codes (e.g., Reed–Solomon, Viterbi, Turbo codes), and signal processing capabilities (e.g., filters, FFTs, equalizers, timing recovery).

GNU Radio applications are mainly written in Python; however, the critical low-level algorithms and signal processing modules are written using the C/C++ programming language, with wide usage of floating-point specific instructions for the relevant processor. Python is primarily used to setup the flow graph, after that most of the work is done in C/C++. GNU Radio is simple to use and a radio receiver can be created in a quick and straightforward manner; moreover, the development of a signal processing algorithm can be carried out using a pre-recorded or generated data set, thus allowing development to occur without the need for a real RF hardware. An example of minimal hardware required to work with GNU Radio is offered by the USRP, developed by Ettus Research, LLC. [13]

GNU Radio applications: The GNU Radio package is provided with a complete HDTV transmitter and receiver, a spectrum analyzer, an oscilloscope, a multichannel receiver and a wide collection of modulators and demodulators. Other advanced projects are still in the feasibility phase or in progress, and includes:

- A system able to recording multiple stations simultaneously
- Time Division Multiple Access (TDMA) waveforms
- A passive radar system that takes advantage of broadcast TV for its signal source
- Radio astronomy
- Digital Radio Mundial (DRM)
- Software GPS

- Amateur radio transceivers

2.3.1 USRP- the Hardware in GNU Radio

USRP, which stands for Universal Software Radio Peripheral,[13,14] is a general purpose motherboard that can host a wide selection of daughter boards, each of which implements a signal processing block found in the GNU Radio software package. The USRP platform hosts the following major modules: [14]

- Four 64-MS/s 12-bit, 85-dB SFDR analog-to-digital (ADC) converters (AD9862),
- Four 128-MS/s 14-bit, 83-dB SFDR digital-to-analog (DAC) converters (AD9862),
- An FPGA that can be reprogrammed (Altera Cyclone EP1C12Q240C8 FPGA),
- A high-speed USB 2.0 interface (Cypress EZ-USB FX2) that can send up to 16 MHz of RF bandwidth in both directions,
- 4 extension sockets (2 TX, 2 RX) in order to connect 2–4 daughterboards,
- 64 GPIO pins available through 4 BasicTX/BasicRX daughterboards (16 pins each), and
- Some glue logic.

Aliasing is, however, an important issue with the ADCs and DACs usage in USRP. When the sample rates after the ADC are down-converted to IF, it is important to note that the maximum sampling rate of ADC is, at most 64 MS/ sec. Therefore, the highest sampling frequency allowable to avoid aliasing is 32 MHz. Similarly the DAC limits the transmitted frequency to a maximum of 64 MHz because of the 128 MS/s sampling rate of DAC.

Figure 11 displays pictures of the USRP both externally and internally. Figure 12 below represents the block diagram of the USRP.

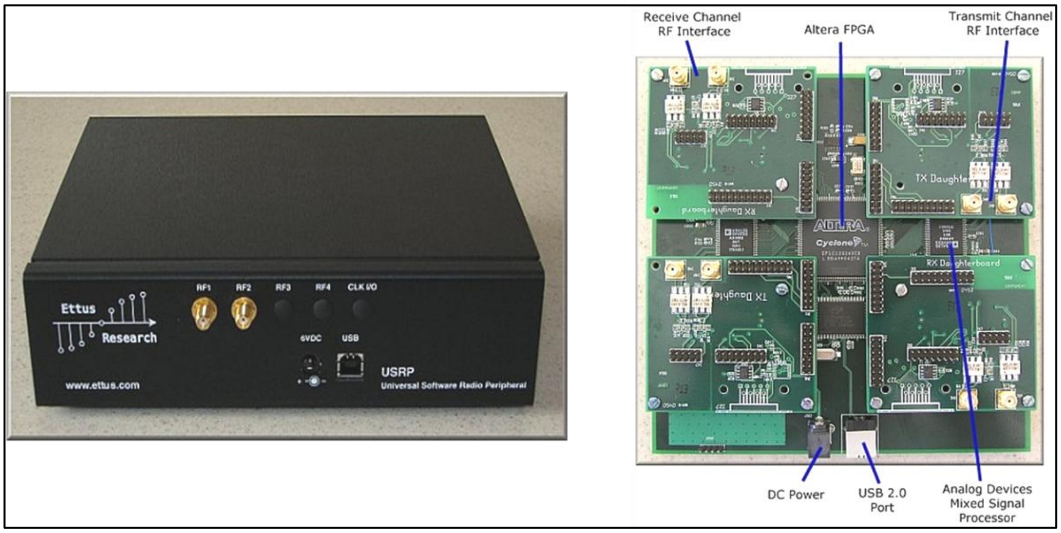

Ettus Research
RF1
RF2
RF3
RF4
CLK I/O
6VDC
USB
USRP
Universal Software Radio Peripheral
www.ettus.com
Receive Channel RF Interface
Altera FPGA
Transmit Channel RF Interface
DC Power
USB 2.0 Port
Analog Devices Mixed Signal Processor

Figure 11: Pictures of USRP both externally (left) and internally (right) [12, 13]

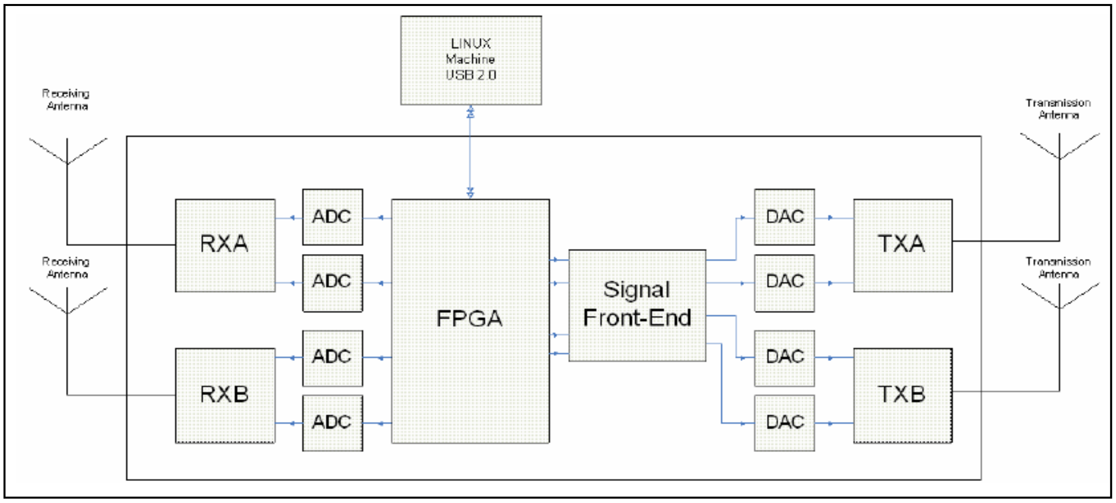


Figure 12: Block diagram of USRP [12, 13]

2.3.2 Functionalities of Different USRP Components:

The USRP provides several functions; digitization of the input signal, digital tuning within the IF band, and sample rate reduction before sending the digitized baseband data to the computing platform via the USB interface. It provides the opposite processing functions for the transmit path. Most the processing performed by the USRP is done in an Altera Cyclone FPGA. An Analog Devices MxFE processor (AD9862) provides some signal processing in the transmit path, and conversion between analog and digital signals for both the transmission and receive paths.

2.3.2.1 Daughterboards

As described earlier, the general purpose motherboard hosts a wide selection of daughterboards to constitute its various signal processing blocks in the USRP (Figures 11 and 12) as the USRP can support up to four daughterboards based on their requirements, generally two for receive and two for transmit. RF front ends are implemented on the daughterboards. The USRP family includes the following based on their varying specifications suited for different jobs such as amplification, filtration, tuning capabilities, etc., in varying frequency ranges. A few of them are listed below [13,14]:

Receivers only support RX (receiving) and consume only one RX port:

- BasicRX, 1–250 MHz Receiver, for use with external RF hardware.
- TVRX, 50–870 MHz Receiver

Transmitters only support TX and consume one TX port:

- BasicTX, 1–250 MHz Transmitter, for use with external RF hardware.
- LFTX, DC to 30 MHz Transmitter.

Transceivers [15] are both TX and RX and consume 2 ports (all come with 70-dB AGC unless specified otherwise):

- WBX, 50 MHz to 2.2 GHz Transceiver, 100-mW output.

- RFX900, 800–1000 MHz Transceiver, 200+ mW output (can be changed into a RFX1800 with basic soldering and flash update).

2.3.2.2 FPGA (Field Programmable Gate Array)

The FPGA on the USRP converts the data rate to match the data rates supported by a USB 2.0 high speed controller. Since the USB bus operates at a maximum rate of 480 million bits per second Mbps, the FPGA must reduce the sample rate in the receive path and increase the sample rate in the transmit path to match the sample rates between the high speed data converter and the lower speeds supported by the USB connection. This entire process is done by several working elements within the FPGA which includes DDCs with cascade-integrator-comp (CIC) filters and MUX. In addition the USB interface combined with the FPGA provide digital IO and low speed analog IO signals for use by the daughterboards. The low speed analog IO points are useful for controlling variable gain amplifiers.

Receive Path: As shown in block diagram (Figure 13), the ADCs are connected to the MUX of FPGA in the receive path and each IQ of the MUX is connected to a DDC which is connected to a data interleave that puts the data in the receive path queue.

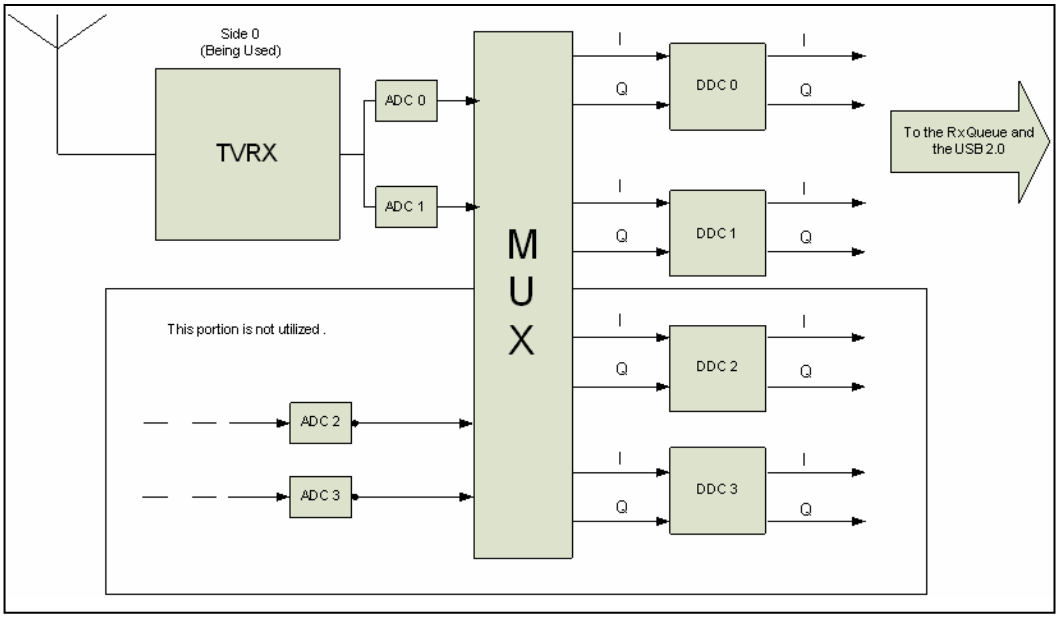


Figure 13: FPGA MUX implementation in receive path [3, 6, 15]

Transmit Path: The CIS filtered I-Q data is demultiplexed in the FPGA, which contains the DEMUX before the digital-up-conversion (DUC) and digital-to-analog conversion (DAC) in its transmission path. Here DUC and DAC are part of Analog Device AD9862 [17] (and not the FPGA) which are connected to BasicTX (TXA/TXB) as demonstrated in block diagram shown in Figure 14)

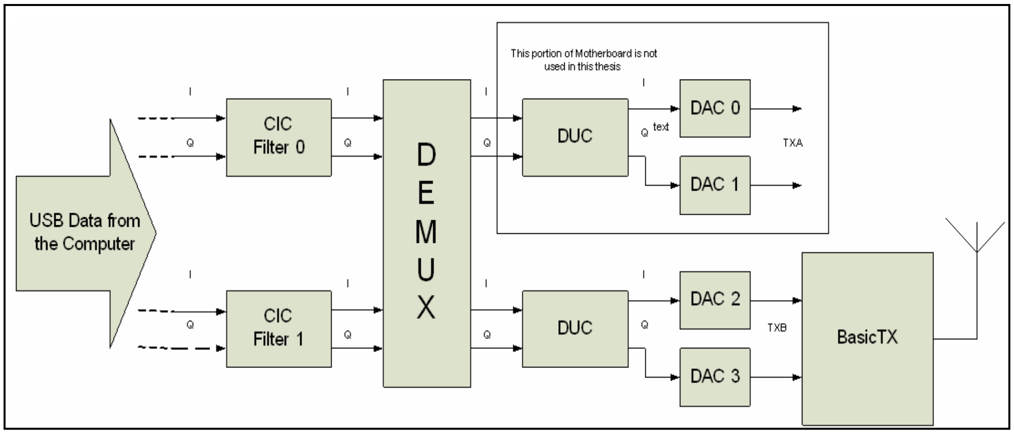

This portion of Motherboard is not used in this thesis
USB Data from the Computer
I
Q
CIC Filter 0
CIC Filter 1
D E M U X
DUC
DUC
text
DAC 0
DAC 1
DAC 2
DAC 3
TXA
TXB
BasicTX

Figure 14 FPGA DEMUX and AD9862 Chip implementation on USRP [3, 6]

2.3.3 Software in GNU Radio

2.3.3.1 Python Programming Language

Python is a dynamic, object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. [18] Many Python programmers report substantial productivity gains and feel the language encourages the development of better code. Python scripts can be written in text files using suffix '*xxx.py*' (e.g., python script.py) which will simply execute the script and return to the terminal afterwards.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, http://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

GNU Radio software architecture is organized using a two-tier structure of Python and C++ programming language. Python script is used for high level organizing, connecting and gluing whereas C++ provides signal processing functionalities. The scheduler in it is using Python's built-in module threading, to control the 'starting', 'stopping' or 'waiting' operations of the signal flow graph.

This thesis is carried out with the GNU Radio software radio and the Universal Software Radio Peripheral.

Chapter 3

System Development

In this chapter we

- Describe the experimental setup of the cognitive radio relay network,
- Describe types of sources used and basic workflow of transmitter and receiver, and
- Introduce mathematical relationships for spectrum detection.

3.1 Experiment Strategy

It was essential to implement the designed (Intelligent Spectrum Sensor Radio) ISSR and therefore a strategy was outlined in order to test the working of system. The objective on this thesis was to have a system that first will receive and transmit an FM signal, as the primary user. Then the signal is analyzed via an FFT plot, and interference with another FM signal is intentionally created at the same frequency, which acts as the first secondary user (Figure 15). Next, the secondary user scans the FM band and determines frequencies in use and holes in the band. Once the holes are identified, the system should be able to transmit a `.wav` file containing music over the air using an FM modulated signal; else the secondary user will keep scanning the FM band indefinitely until holes are detected (Figure 16).

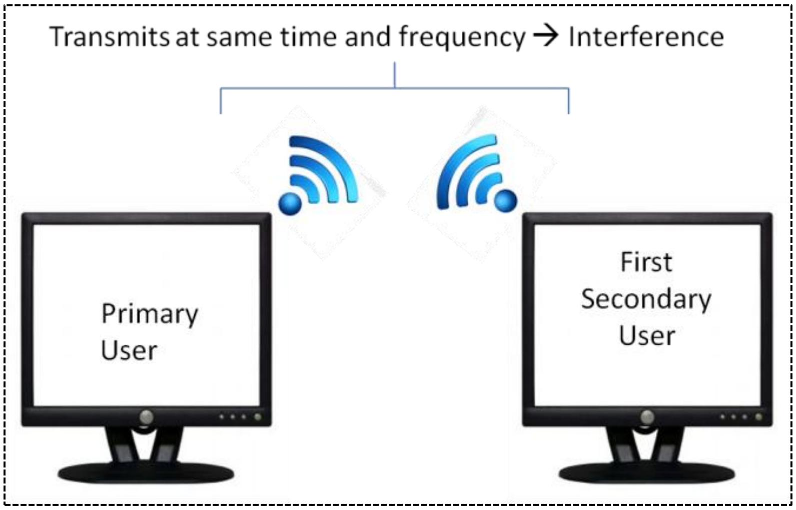


Figure 15: Interference due to simultaneous transmission

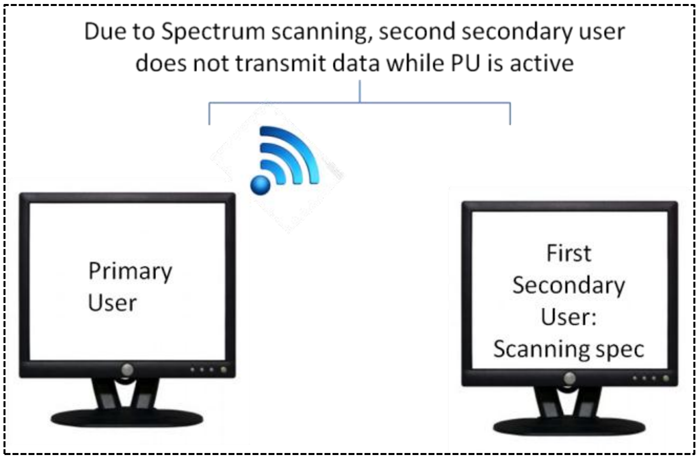

Due to Spectrum scanning, second secondary user does not transmit data while PU is active
Primary User
First Secondary User: Scanning spec

Figure 16: No transmission until PU is active

Once the preceding steps are achieved, a second secondary user will be introduced and intentionally will be made to interfere with the primary user regardless of spectrum scanning. This interference will happen as if the second SU cannot detect the signal because of path loss and/or shadowing effect (Figure 17). This interference can be contained by providing signal-relaying capability of the first secondary user, which receives the beacon signal. The first secondary user will accordingly retransmit the signal it receives from the primary user to the second secondary user which will then be able to detect the spectrum more efficiently. This will allow the second secondary user not to transmit any data until it finds a spectrum hole in the desired FM band (Figure 18). A more concrete way to prove the signal relaying functionality will be shown by stopping the beacon relay intermediately when the primary user is transmitting the data, and the second secondary user continues to scan. By stopping the relaying intermediately, we would expect the second secondary user to again start mis-detecting the spectrum band due to loss of the signal beacon, which will cause that user to interfere with the primary user. This `.wav` file will be stored in the computer and will be received at the same computer with a different file name for simplicity of the test. This `.wav` file should be able to be picked up at that frequency using a FM radio or another USRP. This chapter talks about the experimental setup that was used to perform the given tasks.

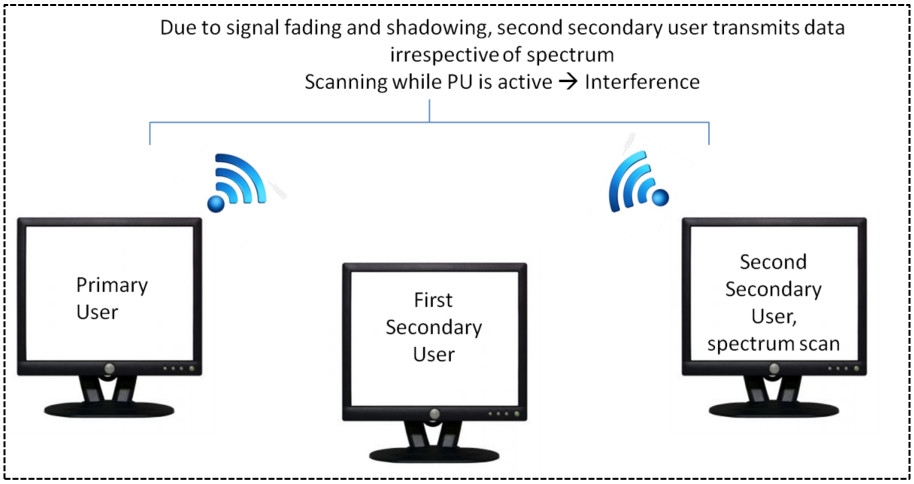

Due to signal fading and shadowing, second secondary user transmits data irrespective of spectrum
Scanning while PU is active → Interference
Primary User
First Secondary User
Second Secondary User, spectrum scan

Figure 17: Interference due to mis-detection of signal by 2nd secondary user

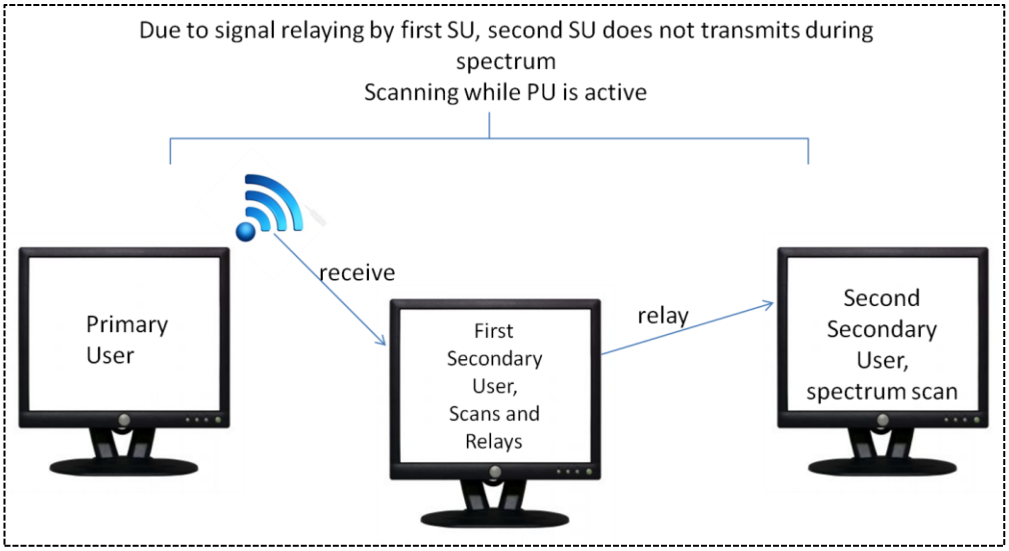

Due to signal relaying by first SU, second SU does not transmits during spectrum
Scanning while PU is active
Primary User
receive
First Secondary User, Scans and Relays
relay
Second Secondary User, spectrum scan

Figure 18: Efficient transmission with relay beacon

3.1.1 USRP and Antennas

The USRP motherboard and the daughterboards have already been discussed in detail in section 2.3. In order to fit the needs of this experiment, three USRPs were used in conjunction with six daughterboards; having two daughterboards for each USRP, which were the BasicRx and the BasicTx. The BasicRx was the main receiving front-end for the system and the BasicTx was used as the main transmitting end. Because the band being used was FM the band, the receiver card was chosen to be the BasicRx which does not include any built-in amplification or filtering and is mainly controlled by the software. This daughterboard was connected directly to the USRP's, on the RX B socket. To receive FM signals a loop antenna was attached directly to the BasicRx board. The gain for this antenna was unknown and was entirely controlled using the Python program. The BasicTx was used to transmit the signals over the air in the FM band. The gain and amplification were controlled entirely via software. The daughterboard was connected directly to the Tx A socket, the choice of which did not affect the experiment in any way.

3.1.2 The Audio Source:

Three audio sources were tested for the entire experiment in this thesis. First was for basic transceiver, second for testing signal detection and third for signal relaying. The first was an mp3 music file stored on the computer. The software tools, discussed in the Section 2.3.3, were capable to transforming the music file very easily to a sample rate and bit rate compatible with the USRP. In the receiver terminal, the USRP performs the DSP and uses the sampling of 32 kS/s after decimation from the ADC rate of 64 MS/s and the same sampling rate was used for the music file. Using the USRP's *file_source()* block, the file was set to be the data source. In the transmitter terminal, the audio source was a digital audio source converted to analog signals by DAC. Therefore the USRP interpolation rate was set to 400 and software interpolation rate was set to 10, setting the sampling rate to 128 MS/s/400/10 = 32 kS/s.

3.2 Experimental Setup

While working with GNU Radio and the USRP, a number of elements were taken into consideration, including the operating system of the computer, soundcard on the computer, the USB interface version etc. It is very important that all components selected for the experimental set up are compatible with each other. The ISSR was set up as shown in Figure 19. It can be seen that the setup is fairly simple and easy to hook up.

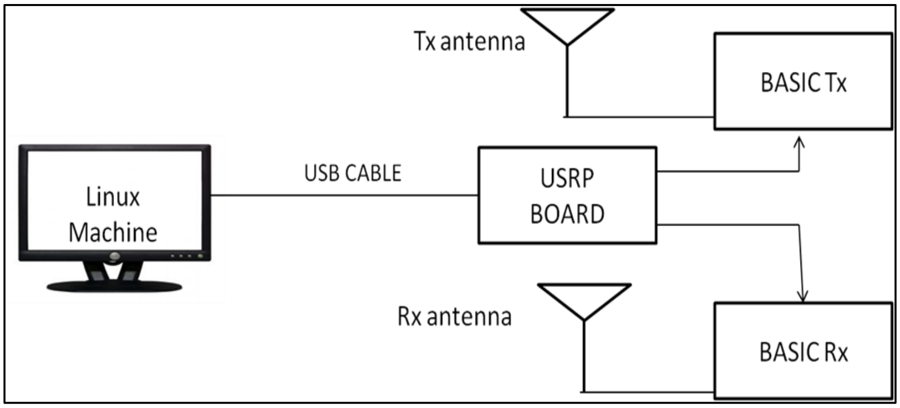


Figure 19: Hardware setup in the experiment

Computer:

GNU Radio works best on a Linux machine and a lot of support is available online. It is also established that the GNU Radio works flawlessly with Fedora as well with Ubuntu 7.04 Feisty Fawn. The compatibility of GNU Radio and the USRP is now well known. The important step was choosing the right Linux machine and we have used the Compaq laptop with Linux machine using Ubuntu 7.04. The detail specifications of the computer used are as follows:

Make and Model:	COMPAQ
System OS:	Ubuntu 7.04
Processor:	Pentium 4 3.0 GHz with HT Technology
System Memory:	1024 MB
Sound Card:	Sound Blaster Audigy 2HX
Graphics Card:	ATI 9600XT 128-MB DDR SDRAM
USB Interface:	USB 2.0
System Storage:	400 GB

3.3 Work Processes

CR networks are designed to facilitate higher bandwidth to mobile users via dynamic spectrum access in a heterogeneous wireless architecture. This is achieved by tapping the underutilized spectrum of certain long term licensed spectrum band in different geological areas. The CR network emphasizes sharing of wireless channels in an opportunistic manner so that it can meet a higher bandwidth without affecting the priority right of access to the licensed band by primary users.

In our work process, the first and immediate task is to continuously monitor and detect those idle portions of spectrum at a particular instant and geographical position which will also be constantly fluctuating or changing.

This thesis implements the above aspect of cognitive radio using the GNU Radio and the USRP combination in the FM band (88 to 108 MHz). The end product is able to sense the entire FM band and perform analysis and estimation routines to identify the ongoing transmission frequencies and spectrum holes (i.e., frequencies where no transmission is occurring) for its intended transmission. This promises a higher spectral efficiency and data rates facilitating an efficient communication protocol. The concepts and methods used to implement such techniques are discussed next.

3.4 Reception Work-flow

3.4.1 USRP Front-End

For detection, reception and analysis of the signal, the amplitude information of the signal is used. The USRP provides information about the signal with the help of daughterboards. In this project, the daughterboard used was BasicRx which has built-in filters and amplifiers and operates within the UHF band which includes the FM band.

The functioning of USRP front-end hardware is discussed in Chapter 2. Here we discuss in more detail the receiver's data pipeline. On the receiving end, the USRP is used in combination with one or two daughterboards although we used only one daughterboard (i.e., the BasicRx) was used. In this board, the system first received and down-converts a real-world analog signal from the FM Band to an IF frequency. This signal is next converted to a digital signal using a Mixed Signal Front End Chipset AD9862 [17] and then decimated down to 4MS/s in this particular system, so that it can be transferred through a USB 2.0 port to be processed by the computer. The stream of data is hen converted to vector form which, in this case, is accompanied using the GNU Radio's built-in Stream-to-Vector block. The values in the vector are complex in I and Q format which are converted to magnitudes before analysis.

The GNU Radio example file *usrp_spectrum_sense.py* was used as the starting point for code development [23]. This script uses the following functions to perform decimation, signal detection, FFT and windowing the details of which are explained in the rest of this chapter:

Table 1: Table of functions

Function	**Operation**
source_c	Put incoming data into variable
set_decim_rate	Set decimation rate
gr.probe_avg_mag_sqrd_c	Spectrum detection
complex_to_mag_squared	Convert complex vector into magnitude squared

3.4.2 Signal Detection and PSD

GNU Radio has built-in signal detection and signal source blocks which most of the time can be used directly to facilitate signal detection. The signal detection block is an energy detector that averages the signal over time and estimates the power spectral density (PSD) of the received signal [20].

The energy spectral density has infinite energy and the Fourier transform for such signal does not exist. Instead, we use power spectral density which can be brought of as the average power over a time period of the signal and is given by (for power, $0 < P < \infty$):

$$P = \lim_{T \to \infty} \int_{-T}^{T} | x(t) |^2 dt$$

As it can be seen, that the average power is basically a windowed version of the average energy of the signal and hence a PSD is a normalized limit of ESD. If the ESD of signal $xT(t)$, then the PSD of the signal is

$$S_x(f) = \lim_{T \to \infty} \frac{1}{2T} | X_T(f) |^2$$

$$P = \int S_x(f) df$$

$$P = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} | x(t) |^2 dt$$

In our system, the PSD is performed as in the example ***usrp_spectrum_sense.py*** and to get the final estimated PSD the vector coming in from the stream is taken by the computer in sections of fixed lengths, L. Digital signal processing including FFT and noise minimizing algorithms, are performed on these data the details for which are explained in subsequent section.

Fourier analysis is performed using the Fourier transform of the signal which is a “generalization of the complex Fourier series”. Since everything is performed in the digital environment, let us look at how the Discrete Fourier Transform (DFT) is defined. The Fourier transform is defined as,

$$F[f(t)] = \int_{-\alpha}^{\alpha} f(t) e^{-2\pi\theta t} dt$$

To make it discrete, the argument t can be set to t_n where $n = 0,1,\ldots,N-1$, which means the DFT is given as,

$$F_n = \sum_{n=0}^{N-1} f_k e^{-2\pi ink/N}$$

A relevant application of Fourier analysis is the frequency spectrum, which plots signal harmonic magnitudes against frequency.

3.5 Transmission Work-flow

The USRP and GNU Radio provide a powerful digital signal processing platform which makes it possible to transmit a data in various different formats and air interfaces. Understanding the transmission protocols used by the USRP and the GNU Radio is very important in order to transmit a signal with the correct properties. Therefore, we will focus on the transmission flowpath that the data take to get from the source to the antenna.

3.5.1 Data Source

For transmission of data, the initial step is to generate a digital signal from a data source. GNU Radio and USRP have provisions for various fast programmable and digital techniques for digital signal generation. Using this digital data, the subsequent radio system processes such as modulation, filtration and other digital signal processing techniques, are easily achieved by generating “synthesis waveforms” [19].

The data source was chosen mostly to be the computer’s sound card during the experiment. The generated digital IQ values coming from the data stream via USB 2.0. This goes through the CIC filters before getting fed to the multiplexer. An important step is to sample the signal properly and set the DAC rates so that the data can be transmitted without any over/under run. Before that is performed, the data are up-converted by passing through the digital up-converters (DUC) and being fed to the DACs and then are ready to be transmitted using the transmitter daughterboard, in our case, the BasicTx.

3.5.2 Modulation Techniques

The signal has to be modulated using a technique that is known to the receiver and is able to be demodulated at the receiver side. Since the band being used is the FM Band, the frequency modulation technique was implemented. Using the BasicTx, an FM signal was intended to be transmitted at a particular free frequency determined by the system.

FM band:

The Intelligent Spectrum-Sensor designed in this thesis project was designed keeping in mind that the FM stations are approximately 200 kHz apart (in the U.S.) with up to 15 kHz wide audio bandwidth, with maximum deviation of ±75 kHz [20] and, therefore, the resolution is set to 100 kHz for receiving and transmission.

3.5.3 Data Sink

In the process of transmission, the blocks of the GNU Radio need to be able to complete the entire path from the source to the sink. The data sink usually the USRP unless it is not being used to transmit. In the transmission line, the following flow graph describes a full working Tx chain (Figure 20):

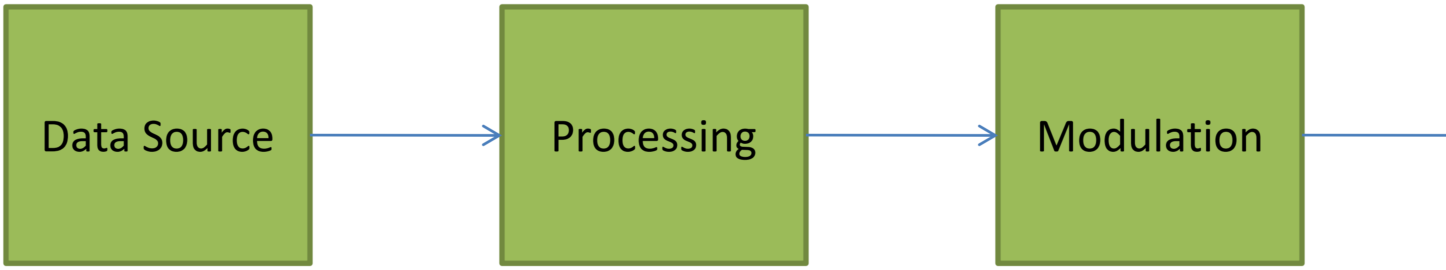


Figure 20: Data transmission pipeline

The following are the salient features of transmission in GNU Radio and USRP:

- Data type changes during processing,
- Data types must match for each connection, and
- Connecting to more than one block is possible (with some limitations).

Chapter 4

Test Results and Analysis

In this chapter we

- Investigate the interference pattern caused by secondary user and how it affects the performance of primary user (Figure 21A);
- Understand how spectrum sensing can reduce the effects of interference to primary user (Figure-21B);
- Discuss extreme environments, i.e., signal fading and shadowing where spectrum sensing fails to minimize the effects of interference (Figure 21C); and
- Introduce signal relaying via which a system can efficiently detect the spectrum (Figure 21D).

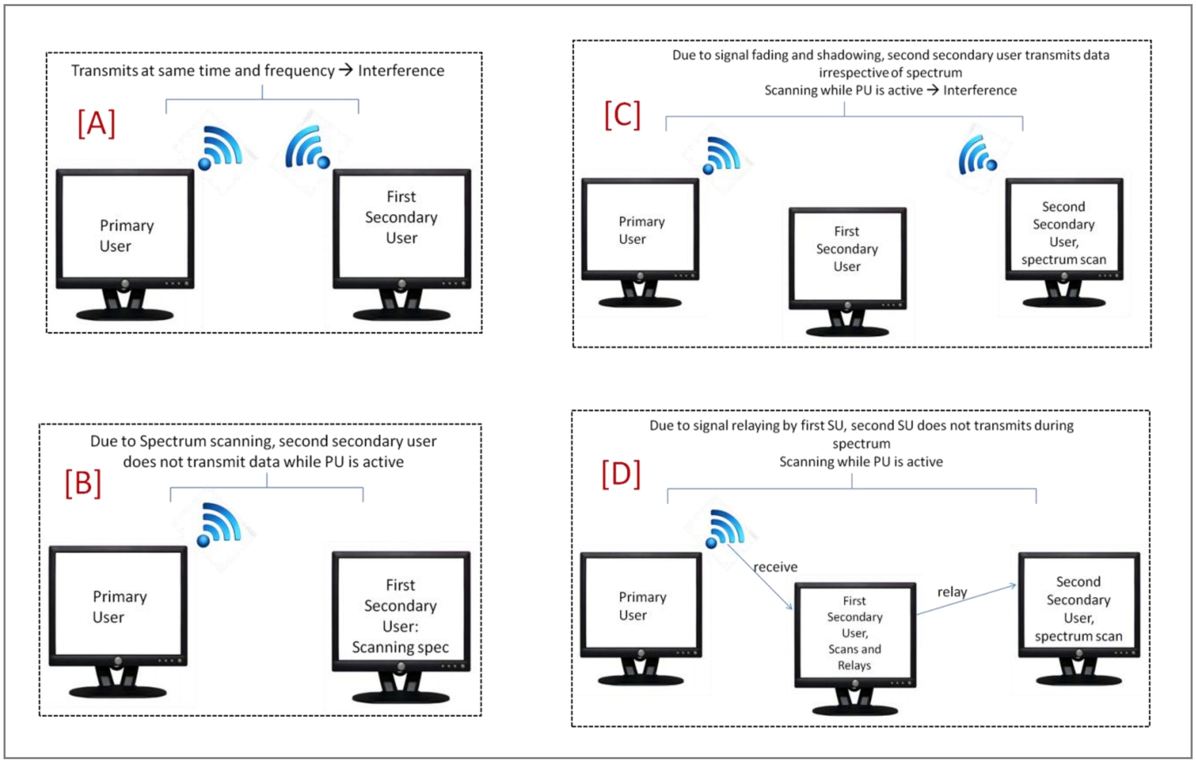

Transmits at same time and frequency → Interference
[A]
Primary User
First Secondary User
Due to signal fading and shadowing, second secondary user transmits data irrespective of spectrum
Scanning while PU is active → Interference
[C]
Primary User
First Secondary User
Second Secondary User, spectrum scan
Due to Spectrum scanning, second secondary user does not transmit data while PU is active
[B]
Primary User
First Secondary User: Scanning spec
Due to signal relaying by first SU, second SU does not transmits during spectrum
Scanning while PU is active
[D]
Primary User
receive
relay
First Secondary User, Scans and Relays
Second Secondary User, spectrum scan

Figure 21: Four different scenarios of spectrum scanning and subsequent transmission scenario by secondary user (SU)

4.1 Testing

After having all the software tools and the hardware configuration, the experiment was carried out. It is easier to look at a state diagram, given in Figure 22, to understand how all the codes are being executed and to see how all the python and C-Shell scripts were called, before looking at the actual screenshots of the experiment's proceedings.

Case I: From the state diagram below, it can be seen that after the execution of C-shell Python script for files *fm_tx.py* and *fm_rcv.py* (Figure 22), which transmit and receive data, respectively. Since there is no obstacle for data transmission, i.e., spectrum detection for holes, any data can be transmitted at the same instant of time and frequency, which will lead to channel interference between the primary and secondary users. FFT plots have been provided below the state diagram, which display the increase in receiver amplitude due to the inability of secondary users to stop its transmission when the primary user is active.

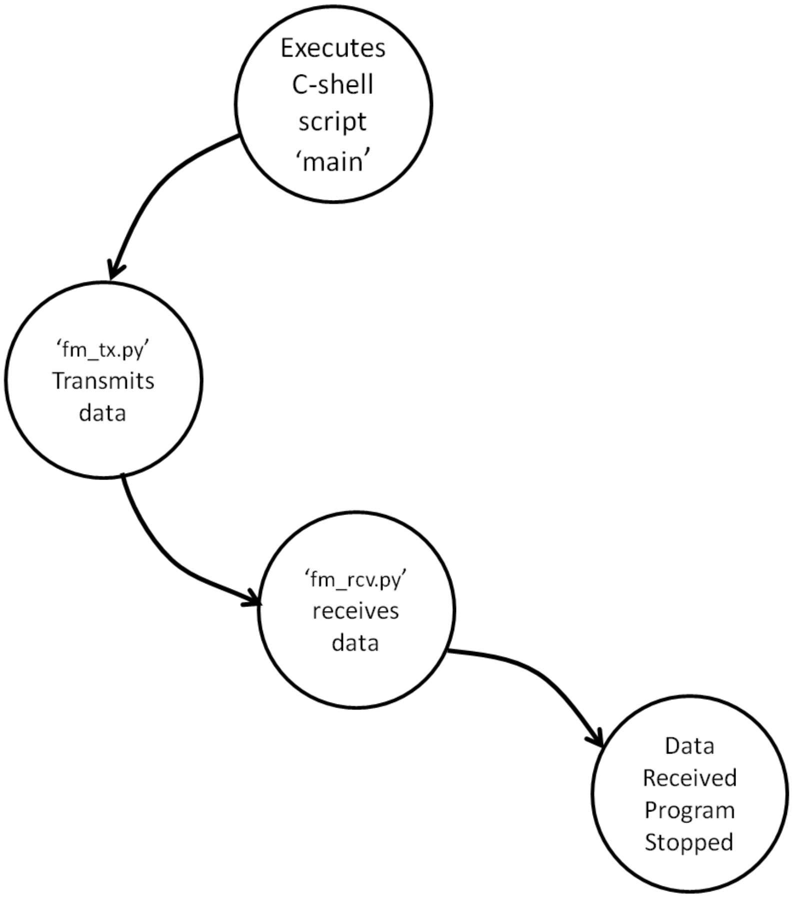
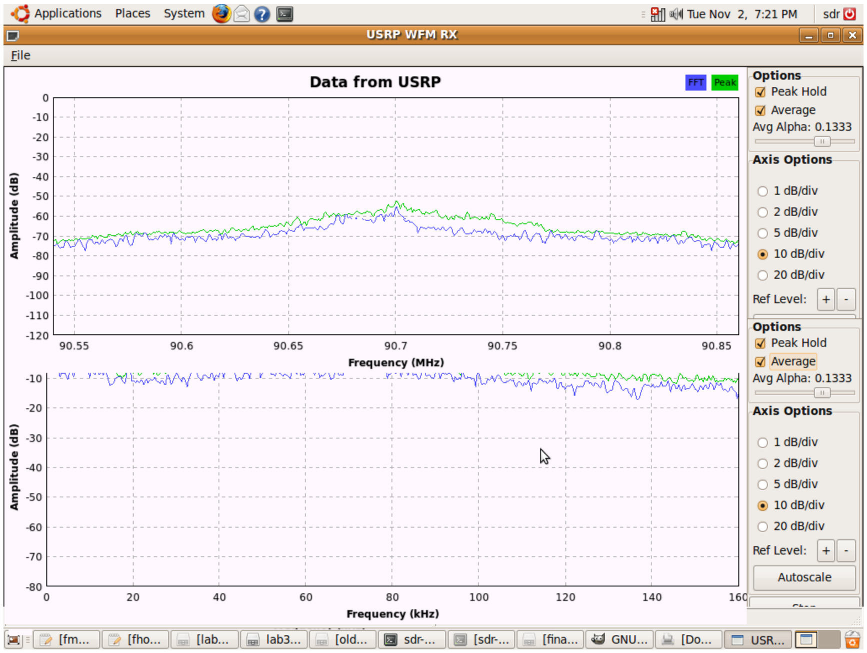

Executes C-shell script 'main'
'fm_tx.py' Transmits data
'fm_rcv.py' receives data
Data Received Program Stopped

Figure 22: State diagram of transmission and its reception

To avoid such situations, a new scheme has been added to the secondary user that will be able to detect when the spectrum is idle or active. The explanation of such development has been described in Cases II and III.

Figure 23 below shows the FFT plot of a FM receiver at 90.7 MHz. As it can be seen, the received signal is extremely weak due to poor reception

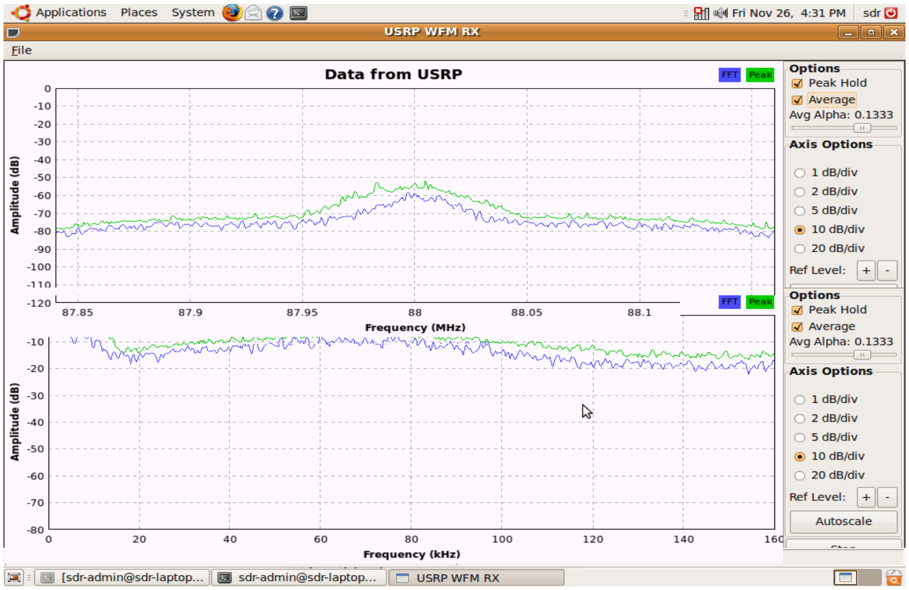


Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div
Ref Level: + -

Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div
Ref Level: + -

Figure 23: FFT plot of a FM receiver at 90.7 MHz

Figure 24 below displays the FFT plot of a FM receiver at 88 MHz where the transmission was done from the base computer.

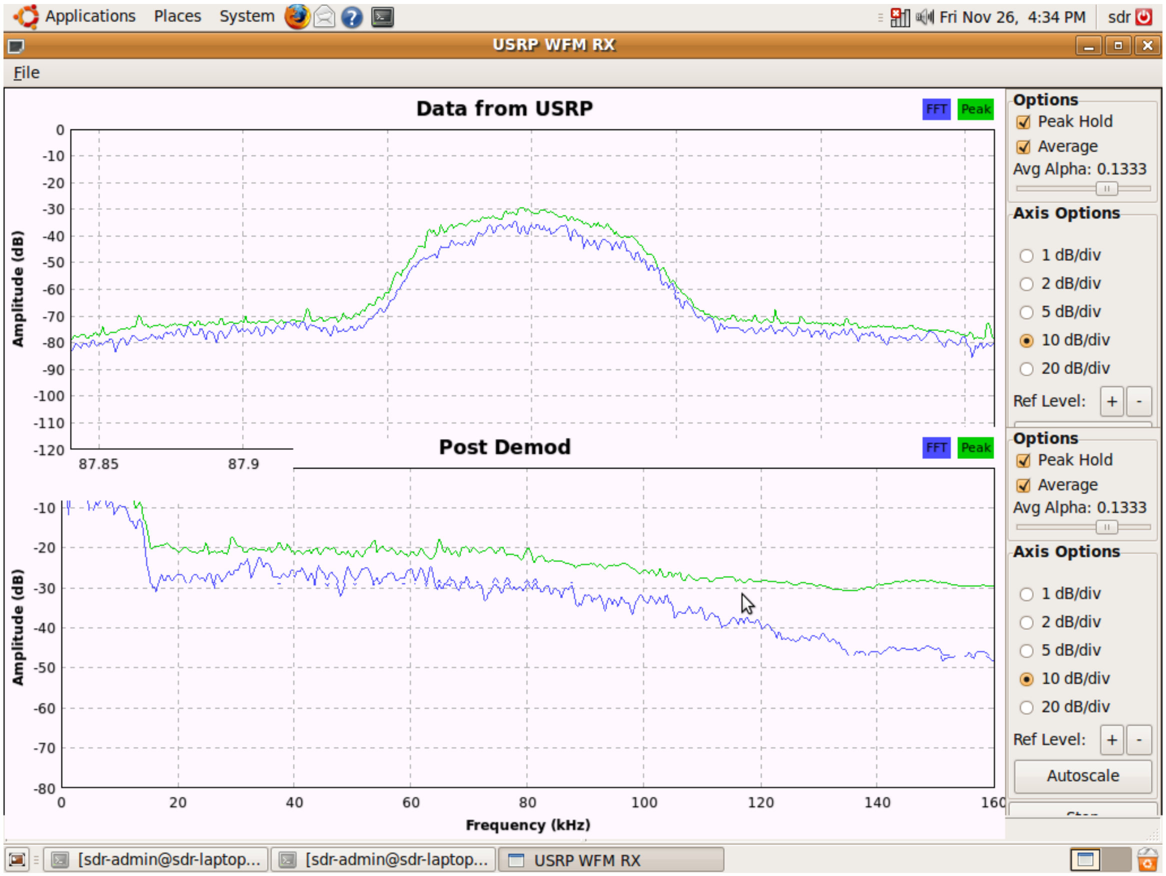


Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div
Ref Level: + -

Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div
Ref Level: + -

Figure 24: FFT plot of a FM receiver at 88 MHz where the transmission was from the base computer

Figure 26 below displays the interference between the primary and secondary user, in which it can be seen that the amplitude level has also increased due to addition of other signal sources.



Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
● 10 dB/div

Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
● 10 dB/div

Figure 25: FFT plot showing reception with interference

To eliminate this interference issue, we introduce the spectrum scanning method for detection of occupied frequency bands, which is discussed in detail in Case II.

Case II: From the diagram below, it can be seen that when the C-shell script ‘main’ executes the file *fhop_trx.py* (Figure 26) the system scans the environment in a loop continuously until it finds a spectrum hole where it will lock to that specific frequency (a bandwidth of 30 Hz is given, ranging from 80.000000 MHz to 80.000003 MHz, a small bandwidth range was provided to show the capability of the spectrum scanning). This will enforce the execution and start transmitting and receiving data at the locked frequency. Due to misdetection at times, a better way to improve efficiency of the specific scheme was by allowing the frequency to be locked and re-check for locking at the same frequency. This is done so as to verify that indeed there is a spectrum hole, and not a misdetection of holes.

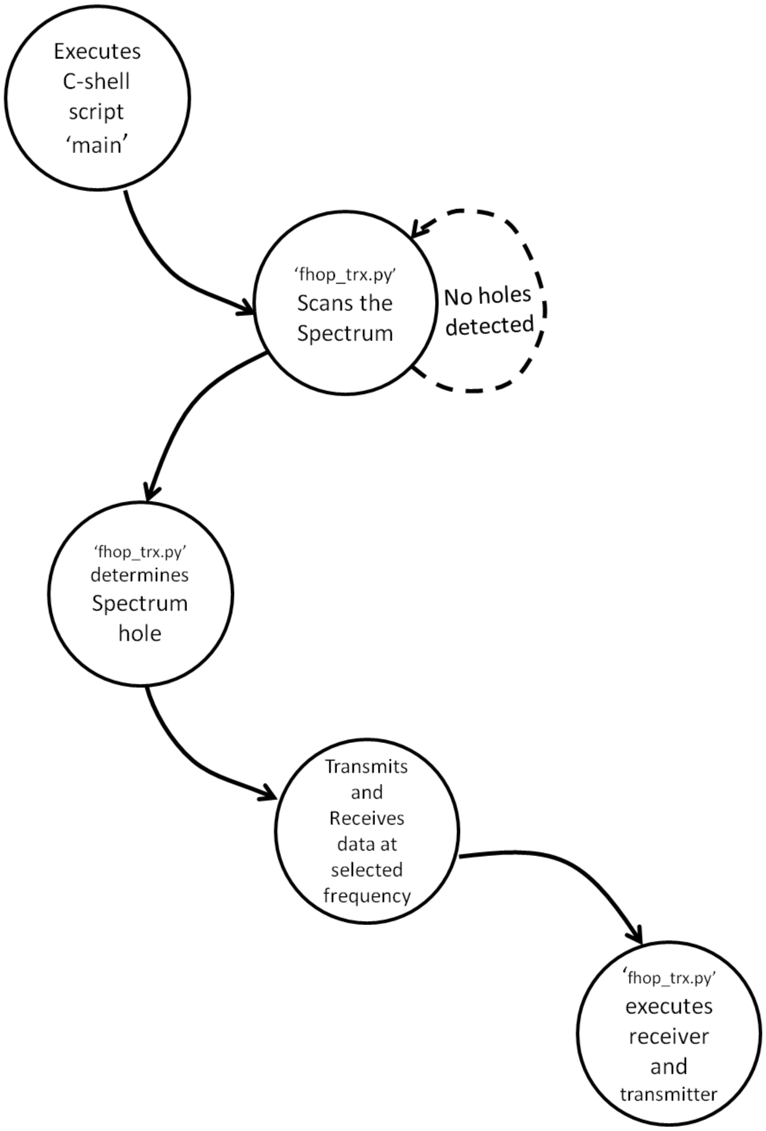
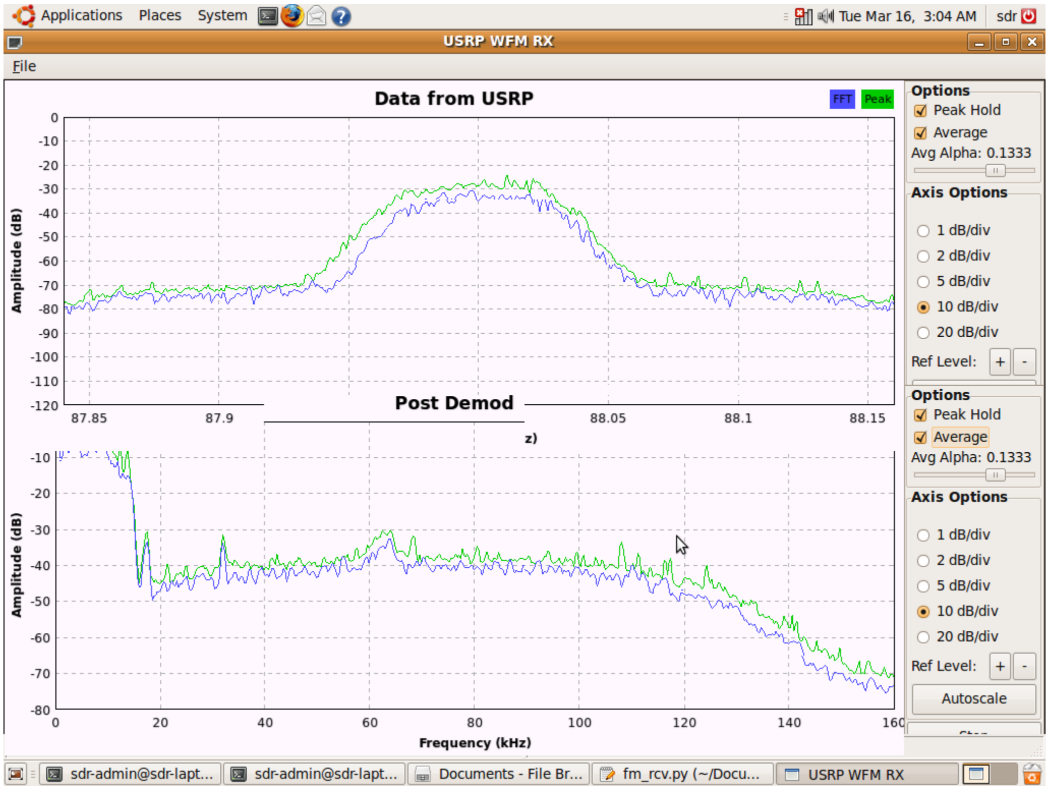


Figure 26: State diagram of transmission and reception with signal detection by secondary user

Figure 27 gives the interference pattern with primary and second secondary user (far away from the primary user, and miss-detects the energy (beacon level)). To eliminate such a case, we introduce beacon signal relaying. The first secondary user (closer to the primary user) retransmits the beacon it receives from the primary user to the second secondary user so that second secondary user can detect the spectrum even more efficiently. The procedure has been discussed in more detail in Case III.



☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
● 10 dB/div

☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
● 10 dB/div

Figure 27: FFT plot of reception with interference owing to misdetection of beacon

Case III: From the diagram below (Figure 28), it can be seen that when the C-shell script ‘main’ executes the file *fhop_trx1.py*, the system scans the environment in a loop continuously until it finds an active spectrum (primary user transmitting data), at which point it will lock itself to the specific frequency (a bandwidth of 30 Hz is given, ranging from 80.000000 MHz to 80.000003 MHz). It will then execute and start *retransmitting* the data it is receiving from primary user at the locked frequency. This step allows the beacon signal transmitted from the primary user to be transmitted such that the second secondary user can detect the spectrum more efficiently. As a result of added efficiency of detection, the secondary user away from the primary user will not misdetect the environment and consequently will not transmit any data when the primary user will be active.

To verify the improvement in the spectrum detection by signal relaying, the relay beacon of the secondary user (closer to the primary user) was stopped randomly when primary user was active. The results show, as expected, that the second secondary user started to misdetect the environment due to signal fading and shadowing and started transmitting the data while primary user was active causing interference.

This proves that the additional feature of relaying the signal indeed improves the detection of signal (Figure 29) and hence reduces interference.

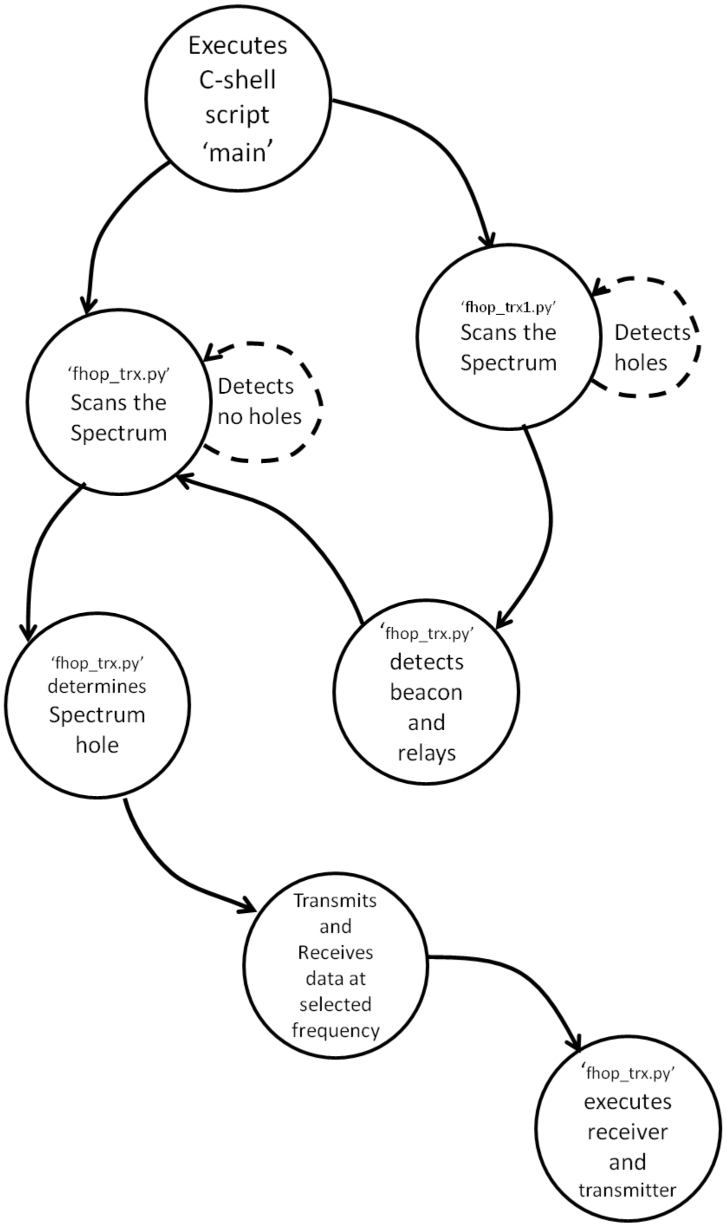
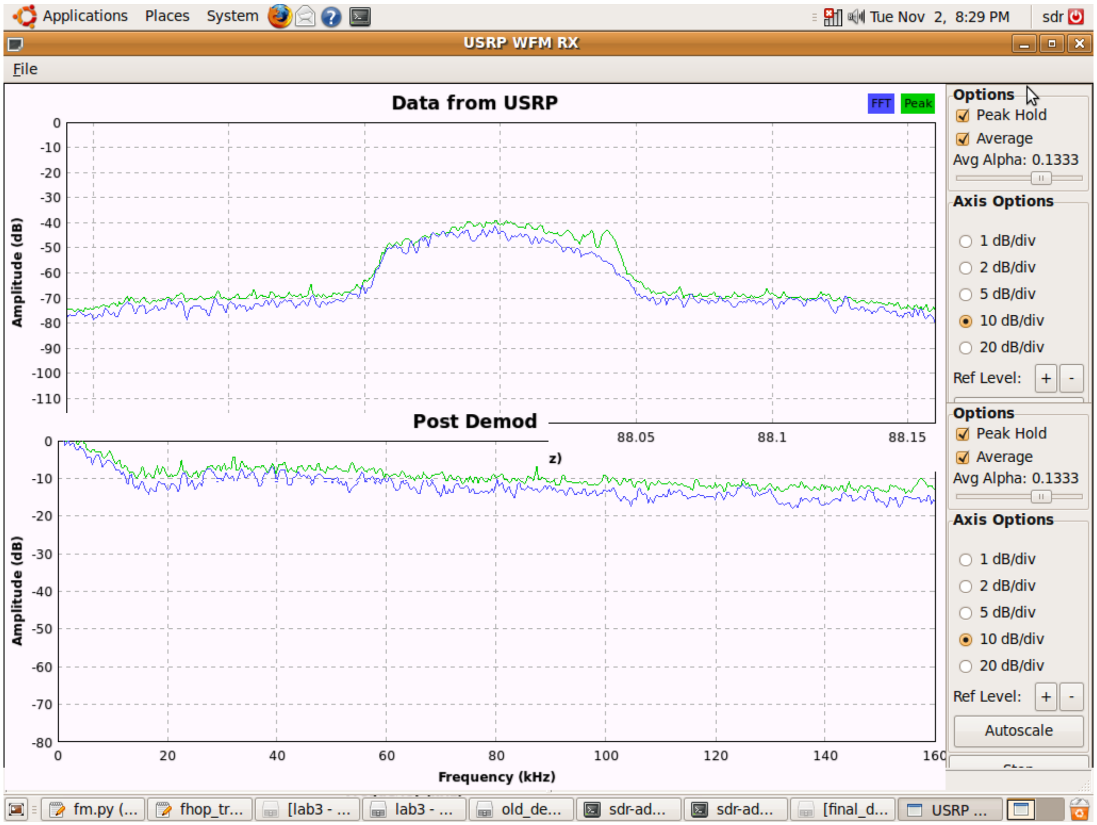

Executes C-shell script 'main'
'fhop_trx1.py' Scans the Spectrum
Detects holes
'fhop_trx.py' Scans the Spectrum
Detects no holes
'fhop_trx.py' detects beacon and relays
'fhop_trx.py' determines Spectrum hole
Transmits and Receives data at selected frequency
'fhop_trx.py' executes receiver and transmitter

Figure 28: State diagram of transmission and receiver with signal detection and relaying

Figure 29 illustrates the case of transmission after efficient detection of signal with relay beacon by second SU. In this case it efficiently sensed the signal strength when PU is inactive so that it can then transmit the signal.



Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div

Options
☑ Peak Hold
☑ Average
Avg Alpha: 0.1333
Axis Options
○ 1 dB/div
○ 2 dB/div
○ 5 dB/div
● 10 dB/div
○ 20 dB/div

Figure 29: Efficient detection and transmission of signal with relay beacon

4.2 Energy Level Tables from SU Scanning

Scanning of spectrum holes are carried out under different scenarios and their responses are discussed in Section 4.1. The scenarios are as follows (Figure 21):

I. Simple reception of data when secondary user (SU) does not transmit as it senses primary user's (PU's) signal to be continuously active;
II. Transmission by 1st SU when PU becomes idle;
III. Transmission by 2nd SU when it misdetects the signal because of path loss and/or shadowing resulting interference;
IV. Relay beacon: 1st SU retransmits when PU is active to enhance signal strength; and
V. Efficient signal scanning and transmission by 2nd SU in the scenario of relay beacon.

The received signal plots are demonstrated in each case through Figure-23, 24, 25, 27 and 29 with their respective state diagrams.

We will now display the readings of the observed signal strength during spectrum scanning and illustrate how they function in each case.

Under the scenarios (I to V mentioned above), observation values are recorded in different tables below (Tables 2 to 5), which explain the cases where the spectrum has detected the environment and determine when the PU is active or is in an idle state.

An important factor to be noted here is that the value for the threshold value to determine if the spectrum is idle or active was determined via experimental tests. To determine a threshold value, we first run the transmission file of the primary user and check the energy level values received by the secondary user. The next test was done by keeping the primary user in an idle state and checking the energy level values received by the secondary user. After many tests and trials, it was finally decided that any value above 300 units (arbitrary) would be considered as active spectrum, and anything below that would be considered as idle. Due to the poor environment conditions, these values did fluctuate a lot at times (energy levels of 30 units were detected during transmission, but this case showed up in a limited case, and they were ignored since in practical environment the transmission and reception signals would be much stronger).

Following are the illustration of different scenarios.

Case-I (Scanning when PU is active): Measures the energy level values (Table 2) detected by *first* secondary user when primary user is active. It can be seen that the secondary user is not transmitting any data unless it detects a spectrum hole.

As explained earlier, in this present case, the secondary user (SU) keeps on scanning for spectrum holes. However, as the primary user (PU) is always active, the SU always detected an energy level that is well above 300 units. In fact the values are of the order of 7600 unit or higher (Table 2). This indicated that PU is active all along and therefore SU did not transmit the signal.

Table 2: Scanning for detection of spectrum holes, none detected as PU is always active

Detecting spectrum by Secondary Users			
PU is always active			
Frequency	**Energy Level**	**Frequency**	**Energy Level**
88000000	13800.336265	88000016	10968.460823
88000001	14196.665286	88000017	11055.228929
88000002	16848.290363	88000018	9581.914221
88000003	14172.541305	88000019	7599.000400
88000004	13057.257114	88000020	7695.212960
88000005	9498.398895	88000021	11019.321112
88000006	8926.247758	88000022	15481.325741
88000007	10632.361293	88000023	15880.696211
88000008	10612.857870	88000024	15246.456260
88000009	10214.947515	88000025	17051.300013
88000010	9860.719208	88000026	17400.511770
88000011	10457.337862	88000027	16521.902391
88000012	10269.027289	88000028	14728.438908
88000013	9694.426690	88000029	12319.368533
88000014	9864.201638	88000000	11896.376349
88000015	10690.068225	*No spectrum hole detected*	

Case-II (Scans and detects spectrum hole): Table 3 displays the energy level values detected by *first* secondary user when primary user is active. It can be seen that the secondary user is not transmitting any data as long as primary user is active, but as soon as the primary user stops transmitting data, the secondary user detects spectrum holes and starts transmitting its own data (Table 3).

If we look at Table 2, it clearly showed that SU detects higher energy level value on initial scanning. But with time, the signal strength drastically reduces to the 1.86 level indicating that the PU became idle. On receiving such scanned information, SU started transmitting on same frequency.

Table 3: Scan and detect spectrum hole and transmit signal when PU becomes idle

Scanning of Spectrum holes by Secondary user				
Detection of spectrum hole as and when PU stops transmitting				
Frequency	**Energy Level**	**Frequency**	**Energy Level**	**Actions**
88000000	8654.0395099	88000024	5138.7871812	
88000001	7713.3916483	88000025	3413.8993824	
88000002	6931.7772115	88000026	1583.8941346	
88000003	7695.2009969	88000027	1106.5906811	
88000004	7940.8171550	88000028	735.4547642	
88000005	8485.7233197	88000029	342.2728272	
88000006	8759.9304076	88000000	159.8916101	
88000007	9288.3902115	88000001	112.1663089	
88000008	9279.2929673	88000002	75.0887831	
88000009	9673.4181165	88000003	36.0300764	
88000010	9136.7634676	88000004	25.8644597	
88000011	9251.5280298	88000005	17.8673104	
88000012	9415.6512500	88000006	9.2330295	
88000013	8615.2803064	88000007	6.7903148	
88000014	7101.7685927	88000008	3.5302162	
88000015	5771.1826093	88000009	1.5912562	Freq1 Locked
88000016	5969.6126808	88000009	1.7166072	Freq2 Locked
88000017	7390.7181985	88000009	1.6557329	Freq3 Locked
88000018	9710.8073202	88000009	1.8600915	Freq4 Locked
88000019	10840.1677694	88000009		
88000020	10723.5158255	tb stopped		
88000021	11866.5900093	tb waiting		
88000022	12221.1290540	tb new connect		
88000023	11079.2710307	tb started again		

Case III (Interference owing to miss-detection by second SU): In this case, the Table 4 displays the energy level values detected by the *second* secondary user when primary user is active. The first SU being nearer to can sense high energy level from PU as it is active, but because of poor environment condition, which causes path loss or shadowing, the second SU received weaker signal and misdetects it even though PU is still active (Table 4). It can be seen that the secondary user is still transmitting its data when the primary user is active irrespective of spectrum detection by the *second* secondary user. This, in turn, again produces signal interference.

Table 4: Misdetection of spectrum hole by SU when PU is still active

Scanning of spectrum holes by Secondary Users in absence of relay beacon		
Misdetect spectrum hole as energy level is very low in absence of beacon because of path loss and/or shadowing when PU is still active		
Frequency	**Energy Level**	**Actions**
88000000	7.309568459	Freq1 Locked
88000000	6.500118012	Freq2 Locked
88000000	6.290790309	Freq3 Locked
88000000	6.967237718	Freq4 Locked
88000000	5.942636258	Freq5 Locked
88000000	5.981474899	Freq6 Locked
88000000		
	tb stopped	
	tb waiting	
	tb new connect	
	tb started again	

Case-IV (Relay Transmission): The SU in this case is meant to relay (retransmit) the signal if it detects PU to be active. The reading of energy level values as detected by *first* secondary is displayed in Table 5, keeping PU always active. It can be seen that the secondary user is now scanning the spectrum to check if the primary user is active or idle; if active the user then locks itself at the specific frequency and starts re-transmitting the data it has received from the primary user. This way it relays the beacon so that distant SU (second SU) can receive enough energy level if PU is active.

Table 5: Relay beacon by first SU when PU is active

Scanning of Spectrum holes by Secondary user		
Retransmission of signal by SU when PU is active		
Frequency	**Energy Level**	**Actions**
88000000	8654.0395099	Freq1 Locked
88000000	7713.3916483	Freq2 Locked
88000000	6931.7772115	Freq3 Locked
88000000	7695.2009969	Freq4 Locked
	tb stopped	
	tb waiting	
	tb new connect	

	tb started again	

Case-V (Efficient detection by second SU because of relay beacon): Table 6 displays the energy level reading detected by the *second* secondary user when the primary user is active. It can be seen that the secondary user is now detecting the spectrum more efficiently and does not transmit its data until the primary user stops transmitting its own data. This resulted due to the re-transmission of the beacon signal by the *first* secondary user. The signal strength will be stronger now for second SU and it can efficiently detect whether PU is active or idle. This helps in reducing the interference between interference between primary and secondary users, and helps communication between users more efficient. The readings in Table 6 illustrate this phenomenon.

An important point to note here is the energy level detection unit is not around 1000. This is mainly due to high data loss. The relaying phenomenon can be realized by the fact that the energy level has increased by 5 times compared to Table 5.

Table 6: Efficient detection by distant SU in presence of relay beacon

Scanning of Spectrum holes by Secondary user			
Efficient detection of spectrum because of relay beacon as path loss and/or shadowing effect is now contained			
Frequency	**Energy Level**	**Frequency**	**Energy Level**
88000000	25.9546404	88000016	23.7485572
88000001	26.8311246	88000017	23.5921345
88000002	27.7609016	88000018	22.1043002
88000003	25.5649990	88000019	19.6328150
88000004	25.2899548	88000020	20.5787475
88000005	23.5531533	88000021	20.6564542
88000006	22.4746604	88000022	21.4525738
88000007	22.5357365	88000023	21.2607286
88000008	22.9870761	88000024	23.0835875
88000009	21.4924421	88000025	23.0180919
88000010	23.1547190	88000026	23.4111362
88000011	24.7725450	88000027	23.8778500
88000012	27.0528277	88000028	24.1346016
88000013	29.5229456	88000029	23.0661158
88000014	30.0563431	88000000	23.0252490
88000015	26.2937977	Efficient Detection	

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The hardware tests shows that the USRP with GNU Radio is capable of spectrum scanning and signal relaying to support the hypothesis. An experiment with interference phenomenon and spectrum scanning by CR is effectively carried out with a Software-Defined Radio system. It shows that SDR can be suitably customized to interface with a hardware transceiver. GNU Radio in conjunction with the USRP is a low cost platform facilitating adequate implementation flexibility and support for modular development. The work defines a baseline structure to work on a cognitive radio philosophy. The work studied the impact of interference in the case of simultaneous transmission and the process to scan and sense spectrum holes to suitably choose the available frequency band.

The project also helped in understanding the application of GNU Radio for deployment in several such research-oriented projects.

5.2 Future Work

Cognitive Radio is a near reality and its potential contribution to spectrum usage is immense. Intelligent spectrum sensing radio using GNU Radio and USRP provided a good basic structure for a cognitive radio system. With a similar enhanced setup, and by using additional USRPs, the study of spectrum scanning and sensing the spectrum holes for transmission is a working option. Also, the model of transmitting a dedicated beacon by primary user for locking of that particular frequency by other cognitive user is also another case of study. Such model setup with beacon comprising relatively higher number of cognitive users can be a case for study. Similar models with cooperative relay of beacon can also be analyzed so as to avoid misdetection of beacon by distant cognitive users.

Implementation of CR-spectrum mobility is also a topmost criterion for system development during our future work. Our future intention is to implement a system where the secondary user will automatically stop transmitting its data as soon as the primary user comes online. This phenomenon is called spectrum mobility, in which the secondary user instantaneously stops transmitting the data and restarts scanning for the frequency spectrum for a hole to transmit rest of the data.

BIBLIOGRAPHY

1] Spectrum Efficiency Working Group. Report of the Spectrum Efficiency Working Group. Technical report, FCC, November 2002.

2] Mitola, J; Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio. PhD thesis, Royal Institute of Technology (KTH), May 2000.

3] Blossom, Eric. "Software Radio." Comsec. Blossom Research, LLC. Feb. 2008 http://www.comsec.com/

4] Shen, Dawei. "Tutorial 4: the USRP Board." Introduction. SDR Documentation. Notre Dame, IN: University of Notre Dame, 2005. Oct. 2007 http://www.nd.edu/~jnl/sdr/docs/

5] Classification by tier system in Software-Defined Radio: http://www.radio-electronics.com/info/receivers/sdr/software-defined-radios-tutorial.php

6] Patton, Lee K. A GNU Radio Based Software-Defined Radar. Wright State University. 2007.

7] Mian Omer; Intelligent Spectrum Sensor Radio: http://rave.ohiolink.edu/etdc/view?acc_num=wright1215360432

8] M. Gastpar, "On capacity under receive and spatial spectrum-sharing constraints," *IEEE Trans. Inform. Theory*, vol. 53, pp. 471–487, Feb. 2007.

9] Y. Xing, C. Marthur, M. Haleem, R. Chandramouli, and K. Subbalakshmi, "Dynamic spectrum access with QoS and interference temperature constraints," *IEEE Trans. Mobile Comput.*, vol. 6, no. 4, pp. 423–433, Apr. 2007.

10] W. Weng, T. Peng, and W. Wang, "Optimal power control under interference temperature constraints in cognitive radio network," in *Proc. IEEE Wireless Communications and Networking Conf.*, Hong Kong, China, Mar. 2007, pp. 116–120.

11] Mai Vu, Saeed S. Ghassemzadeh, and Vahid Tarokh, "Interference in a cognitive network with beacon", WCNC 2008, IEEE, pp. 876–881; April 2008.

12] http://dev.emcelettronica.com/gnu-radio-open-source-software-defined-radio

13] Ettus, Matt. "USRP Datasheet." Ettus Research LLC. Oct. 2007: http://www.ettus.com

14] USRP Peripherals: http://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral

15] Daughterboards as transceivers:http://www.ettus.com/downloads/ettus_ds_transceiver_dbrds_v6c.pdf

16] Field Programmable Gate Array-FPGA: http://en.wikipedia.org/wiki/Field-programmable_gate_array

17] Analog Devices AD9862—12/14-Bit Mixed Signal Front-End (MxFE®) Processor for Broadband Communications. http://www.analog.com/en/prod/0%2C2877%2CAD9862%2C00.html.

18] Python Programming Language—Official Website. Dec. 2007: http://docs.python.org/tutorial/

19] Reed, J H. Software Radio: a Modern Approach to Radio Engineering. Prentice Hall, 2002

20] "FM Broadcast Band." Wikipedia. Feb. 2008 http://en.wikipedia.org/wiki/FM_broadcast_band

Appendix-A

MATLAB Files:

A1: γ vs. Interference, I_0 -(Reference Figure 5)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
alpha = 2.1;
R=100000;
P=2.6;
A = 2*pi*lambda*beta*P/(alpha-2);
g = 0.01:0.01:10;ep = 0.2;Ro = 5;
%g = 0.2;ep = 0.01:0.01:10;Ro = 5;
%g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
B= 1./(ep.^(alpha-2));
C = exp(-g.*((ep+Ro).^alpha)).*(ep+Ro).^(2-alpha);
D1 = gammainc(2/alpha,g.*((ep+Ro).^alpha));
D2 = gammainc(2/alpha,g.*((R+Ro).^alpha));
D = g.^((alpha-2)/alpha).*(D1-D2);
E = 1./R^(alpha-2);
F = exp(-g.*((R+Ro).^alpha).*(R+Ro).^(2-alpha));
I = A *(B - E - C + F) + D ;
semilogx(g,I);
xlabel('gamma')
ylabel('Interference Io')
gtext({'alpha = 2.1','Ro = 0.5','epsilon = 0.2','R = 100,000'})
```

A2: ε vs. Interference, I_0 -(Reference Figure 6)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
alpha = 2.1;
R=100000;
P=2.6;
A = 2*pi*lambda*beta*P/(alpha-2);
```

```
%g = 0.01:0.01:10;ep = 0.2;Ro = 5;
g = 0.2;ep = 0.01:0.01:10;Ro = 5;
%g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
B= 1./(ep.^(alpha-2));
C = exp(-g.*((ep+Ro).^alpha)).*(ep+Ro).^(2-alpha);
D1 = gammainc(2/alpha,g.*((ep+Ro).^alpha));
D2 = gammainc(2/alpha,g.*((R+Ro).^alpha));
D = g.^((alpha-2)/alpha).*(D1-D2);
E = 1./R^(alpha-2);
F = exp(-g.*((R+Ro).^alpha).*(R+Ro).^(2-alpha));
I = A *(B - E - C + F) + D ;
semilogx(ep,I);
xlabel('epsilon')
ylabel('Interference Io')
gtext({'alpha = 2.1','Ro = 0.5','gamma = 0.2','R = 100,000'})
```

A3: R_0 vs. Interference, I_0 - (Reference Figure 7)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
alpha = 2.1;
R=100000;
P=2.6;
A = 2*pi*lambda*beta*P/(alpha-2);
%g = 0.01:0.01:10;ep = 0.2;Ro = 5;
%g = 0.2;ep = 0.01:0.01:10;Ro = 5;
g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
B= 1./(ep.^(alpha-2));
C = exp(-g.*((ep+Ro).^alpha)).*(ep+Ro).^(2-alpha);
D1 = gammainc(2/alpha,g.*((ep+Ro).^alpha));
D2 = gammainc(2/alpha,g.*((R+Ro).^alpha));
D = g.^((alpha-2)/alpha).*(D1-D2);
E = 1./R^(alpha-2);
F = exp(-g.*((R+Ro).^alpha).*(R+Ro).^(2-alpha));
I = A *(B - E - C + F) + D ;
semilogx(Ro,I);
xlabel('Ro')
ylabel('Interference Io')
gtext({'alpha = 2.1','epsilon = 0.2','gamma = 0.2','R = 100,000'})
```

A4: γ vs. var[I_0] (Reference Figure 8)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
```

```
alpha = 2.1;
R=100000;
P=2.6;
g = 0.01:0.01:10;ep = 0.2;Ro = 5;
%g = 0.2;ep = 0.01:0.01:10;Ro = 5;
%g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
A = 1/(alpha-1).*exp(-g.*(ep+Ro)^alpha).*(ep+Ro)^(-2.*(alpha-1))
B = ((alpha.*g)./((alpha-1).*(alpha-2))).*exp(-
g.*(ep+Ro)^alpha).*(ep+Ro)^(2-alpha)
C = ((alpha.*g.^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,g.*((ep+Ro).^alpha))

A1 = 1/(alpha-1).*exp((-2.*g).*(ep)^alpha).*(ep)^(-2.*(alpha-1))
B1 = ((alpha.*(2.*g)./((alpha-1).*(alpha-2))).*exp(-
(2.*g).*(ep)^alpha).*(ep)^(2-alpha))
C1 = ((alpha.*(2.*g).^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,(2.*g).*((ep).^alpha))

D = 6*(P^2)*(beta^2)*lambda*pi;
E = (1/2*(alpha-1))*1/(ep)^2.*(alpha-1);
F = exp(-2.*g.*Ro^alpha);
var = D.*(E - (A - B + C) + 0.5.*F.*(A1-B1+C1)) ;
semilogx(g,var)
xlabel('gamma')
ylabel('Interference varience Var(Io)')
gtext({'alpha = 2.1','Ro = 0.5','epsilon = 0.2','R = 100,000'})
```

A5: R_0 vs. var[I_0] -(Reference Figure 9)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
alpha = 2.1;
R=100000;
P=2.6;
%g = 0.01:0.01:10;ep = 0.2;Ro = 5;
%g = 0.2;ep = 0.01:0.01:10;Ro = 5;
g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
A = 1/(alpha-1).*exp(-g.*(ep+Ro).^alpha).*(ep+Ro).^(-2.*(alpha-1))
B = ((alpha.*g)./((alpha-1).*(alpha-2))).*exp(-
g.*(ep+Ro).^alpha).*(ep+Ro).^(2-alpha)
C = ((alpha.*g.^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,g.*((ep+Ro).^alpha))
```

```
A1 = 1./(alpha-1).*exp((-2.*g).*(ep).^alpha).*(ep).^(-2.*(alpha-1))
B1 = ((alpha.*(2.*g)./((alpha-1).*(alpha-2))).*exp(-
(2.*g).*(ep).^alpha).*(ep).^(2-alpha))
C1 = ((alpha.*(2.*g).^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,(2.*g).*((ep).^alpha))

D = 6*(P^2)*(beta^2)*lambda*pi;
E = (1./2.*(alpha-1)).*1./(ep).^2.*(alpha-1);
F = exp(-2.*g.*Ro.^alpha);
var = D.*(E - (A - B + C) + 0.5.*F.*(A1-B1+C1)) ;
semilogx(Ro,var)
xlabel('Ro')
ylabel('Interference varience Var(Io)')
gtext({'alpha = 2.1','gamma = 0.2','epsilon = 0.2','R = 100,000'})
```

A6: ε_0 vs. var[I_0] -(Reference Figure 10)

```
%h = (A/d^a/2).*hbar;
lambda =.6;
beta = .5; % active factor of cognitive user
alpha = 2.1;
R=100000;
P=2.6;
%g = 0.01:0.01:10;ep = 0.2;Ro = 5;
g = 0.2;ep = 0.01:0.01:10;Ro = 5;
%g = 0.2;ep = 0.2;Ro = 0.01:0.01:10;
A = 1/(alpha-1).*exp(-g.*(ep+Ro).^alpha).*(ep+Ro).^(-2.*(alpha-1))
B = ((alpha.*g)./((alpha-1).*(alpha-2))).*exp(-
g.*(ep+Ro).^alpha).*(ep+Ro).^(2-alpha)
C = ((alpha.*g.^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,g.*((ep+Ro).^alpha))

A1 = 1./(alpha-1).*exp((-2.*g).*(ep).^alpha).*(ep).^(-2.*(alpha-1))
B1 = ((alpha.*(2.*g)./((alpha-1).*(alpha-2))).*exp(-
(2.*g).*(ep).^alpha).*(ep).^(2-alpha))
C1 = ((alpha.*(2.*g).^(2-2/alpha))./((alpha-1).*(alpha-
2))).*gammainc(2/alpha,(2.*g).*((ep).^alpha))

D = 6*(P^2)*(beta^2)*lambda*pi;
E = (1./2.*(alpha-1)).*1./(ep).^2.*(alpha-1);
F = exp(-2.*g.*Ro^alpha);
var = D.*(E - (A - B + C) + 0.5.*F.*(A1-B1+C1)) ;
semilogx(ep,var)
xlabel('epsilon')
ylabel('Interference varience Var(Io)')
gtext({'alpha = 2.1','Ro = 5','gamma = 0.2','R = 100,000'})
```

Appendix-B

B1: fm_rcv.py (receiver file)

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

from gnuradio import gr, gru, eng_notation, optfir
from gnuradio import audio
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
import math
import wx

def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    Try for one of these, in order: TV_RX, BASIC_RX, whatever is on
side A.

    @return a subdev_spec
    """
    return usrp.pick_subdev(u, (usrp_dbid.TV_RX,
                                usrp_dbid.TV_RX_REV_2,
                      usrp_dbid.TV_RX_REV_3,
                                usrp_dbid.BASIC_RX))
```

```
class wfm_rx_block (stdgui2.std_top_block):
    def __init__(self,frame,panel,vbox,argv):
        stdgui2.std_top_block.__init__ (self,frame,panel,vbox,argv)

        parser=OptionParser(option_class=eng_option)
        parser.add_option("-R", "--rx-subdev-spec", type="subdev",
default="A",
                          help="select USRP Rx side A or B
(default=A)")
        parser.add_option("-f", "--freq", type="eng_float",
default=90.7e6,
                          help="set frequency to FREQ",
metavar="FREQ")
        parser.add_option("-g", "--gain", type="eng_float",
default=40,
                          help="set gain in dB (default is midpoint)")
        parser.add_option("-V", "--volume", type="eng_float",
default=None,
                          help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string",
default="",
                          help="pcm device name.  E.g., hw:0,0 or
surround51 or /dev/dsp")

        (options, args) = parser.parse_args()
        if len(args) != 0:
            parser.print_help()
            sys.exit(1)

        self.frame = frame
        self.panel = panel

        self.vol = 1
        self.state = "FREQ"
        self.freq = 0

        # build graph

        self.u = usrp.source_c()                    # usrp is data
source

        adc_rate = self.u.adc_rate()                # 64 MS/s
        usrp_decim = 200
        self.u.set_decim_rate(usrp_decim)
        usrp_rate = adc_rate / usrp_decim           # 320 kS/s
        chanfilt_decim = 1
        demod_rate = usrp_rate / chanfilt_decim
        audio_decimation = 10
        audio_rate = demod_rate / audio_decimation  # 32 kHz
        if options.rx_subdev_spec is None:
            options.rx_subdev_spec = pick_subdevice(self.u)

        self.u.set_mux(usrp.determine_rx_mux_value(self.u, 
options.rx_subdev_spec))
        self.subdev = usrp.selected_subdev(self.u, 
options.rx_subdev_spec)
        print "Using RX d'board %s" % (self.subdev.side_and_name(),)
        dbid = self.subdev.dbid()
        if not (dbid == usrp_dbid.BASIC_RX or
                dbid == usrp_dbid.TV_RX or
                dbid == usrp_dbid.TV_RX_REV_2 or
                dbid == usrp_dbid.TV_RX_REV_3):
            print "This daughterboard does not cover the required 
frequency range"
            print "for this application.  Please use a BasicRX or TVRX 
daughterboard."
            raw_input("Press ENTER to continue anyway, or Ctrl-C to 
exit.")

        chan_filt_coeffs = optfir.low_pass (1,           # gain
                                            usrp_rate,   # sampling 
rate
                                            80e3,        # passband 
cutoff
                                            115e3,       # stopband 
cutoff
                                            0.1,         # passband 
ripple
                                            60)          # stopband 
attenuation
        #print len(chan_filt_coeffs)
        chan_filt = gr.fir_filter_ccf (chanfilt_decim, 
chan_filt_coeffs)

        self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)


        self.volume_control = gr.multiply_const_ff(self.vol)

        # sound card as final sink
        audio_sink = audio.sink (int 
(audio_rate),options.audio_output,False)  # ok_to_block
      file_sink = gr.wavfile_sink ("chann1.wav", 2, audio_rate, 16)  # 
ok_to_block

        # now wire it all together
        self.connect (self.u, chan_filt, self.guts, 
self.volume_control, audio_sink)
      self.connect (self.volume_control, file_sink)

        self._build_gui(vbox, usrp_rate, demod_rate, audio_rate)
        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.subdev.gain_range()
            options.gain = float(g[0]+g[1])/2

        if options.volume is None:
            g = self.volume_range()
            options.volume = float(g[0]+g[1])/2

        if abs(options.freq) < 1e6:
            options.freq *= 1e6

        # set initial values

        self.set_gain(options.gain)
        self.set_vol(options.volume)
        if not(self.set_freq(options.freq)):
            self._set_status_msg("Failed to set initial frequency")


    def _set_status_msg(self, msg, which=0):
        self.frame.GetStatusBar().SetStatusText(msg, which)


    def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

        def _form_set_freq(kv):
            return self.set_freq(kv['freq'])


        if 1:
            self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data
from USRP",
                                               fft_size=512,
sample_rate=usrp_rate,
                                  ref_scale=32768.0, ref_level=0,
y_divs=12)
            self.connect (self.u, self.src_fft)
            vbox.Add (self.src_fft.win, 4, wx.EXPAND)

        if 1:
            post_filt_fft = fftsink2.fft_sink_f(self.panel,
title="Post Demod",
                                                fft_size=1024,
sample_rate=usrp_rate,
                                                y_per_div=10,
ref_level=0)
            self.connect (self.guts.fm_demod, post_filt_fft)
            vbox.Add (post_filt_fft.win, 4, wx.EXPAND)

        if 0:
            post_deemph_fft = fftsink2.fft_sink_f(self.panel,
title="Post Deemph",
                                                  fft_size=512,
sample_rate=audio_rate,
                                                  y_per_div=10,
ref_level=-20)
            self.connect (self.guts.deemph, post_deemph_fft)
            vbox.Add (post_deemph_fft.win, 4, wx.EXPAND)


        # control area form at bottom
        self.myform = myform = form.form()

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)
        myform['freq'] = form.float_field(
            parent=self.panel, sizer=hbox, label="Freq", weight=1,
            callback=myform.check_input_and_call(_form_set_freq,
self._set_status_msg))

        hbox.Add((5,0), 0)
        myform['freq_slider'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
weight=3,
                                        range=(87.9e6, 108.1e6,
0.1e6),
                                        callback=self.set_freq)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)

        myform['volume'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
label="Volume",
                                        weight=3,
range=self.volume_range(),
                                        callback=self.set_vol)
        hbox.Add((5,0), 1)

        myform['gain'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
label="Gain",
                                        weight=3,
range=self.subdev.gain_range(),
                                        callback=self.set_gain)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        try:
            self.knob = powermate.powermate(self.frame)
            self.rot = 0
            powermate.EVT_POWERMATE_ROTATE (self.frame,
self.on_rotate)
            powermate.EVT_POWERMATE_BUTTON (self.frame,
self.on_button)
        except:
            print "FYI: No Powermate or Contour Knob found"


    def on_rotate (self, event):
        self.rot += event.delta
        if (self.state == "FREQ"):
            if self.rot >= 3:
                self.set_freq(self.freq + .1e6)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_freq(self.freq - .1e6)
                self.rot += 3
        else:
            step = self.volume_range()[2]
            if self.rot >= 3:
                self.set_vol(self.vol + step)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_vol(self.vol - step)
                self.rot += 3

    def on_button (self, event):
        if event.value == 0:        # button up
            return
        self.rot = 0
        if self.state == "FREQ":
            self.state = "VOL"
        else:
            self.state = "FREQ"
        self.update_status_bar ()


    def set_vol (self, vol):
        g = self.volume_range()
        self.vol = max(g[0], min(g[1], vol))
        self.volume_control.set_k(10**(self.vol/10))
        self.myform['volume'].set_value(self.vol)
        self.update_status_bar ()

    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool
        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital down converter.
        """
        r = usrp.tune(self.u, 0, self.subdev, target_freq)

        if r:
            self.freq = target_freq
            self.myform['freq'].set_value(target_freq)         #
update displayed value
            self.myform['freq_slider'].set_value(target_freq)  #
update displayed value
            self.update_status_bar()
            self._set_status_msg("OK", 0)
            return True

        self._set_status_msg("Failed", 0)
        return False

    def set_gain(self, gain):
        self.myform['gain'].set_value(gain)     # update displayed
value
        self.subdev.set_gain(gain)

    def update_status_bar (self):
        msg = "Volume:%r  Setting:%s" % (self.vol, self.state)
        self._set_status_msg(msg, 1)
        self.src_fft.set_baseband_freq(self.freq)

    def volume_range(self):
        return (-20.0, 0.0, 0.5)


if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP WFM RX")
    app.MainLoop ()
```

B2: fm_tx.py (transmitter file)

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

"""
Transmit N simultaneous narrow band FM signals.

They will be centered at the frequency specified on the command line,
and will spaced at 25kHz steps from there.

The program opens N files with names audio-N.dat where N is in [0,7].
These files should contain floating point audio samples in the range
[-1,1]
sampled at 32kS/sec.  You can create files like this using
audio_to_file.py
"""

from gnuradio import gr, eng_notation
from gnuradio import usrp
from gnuradio import audio
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from optparse import OptionParser
from usrpm import usrp_dbid
import math
import sys

from gnuradio.wxgui import stdgui2, fftsink2
#from gnuradio import tx_debug_gui
import wx


########################################################
# instantiate one transmit chain for each call

class pipeline(gr.hier_block2):
    def __init__(self, filename, audio_rate, if_rate):

        gr.hier_block2.__init__(self, "pipeline",
                                gr.io_signature(0, 0, 0),
# Input signature
                                gr.io_signature(1, 1,
gr.sizeof_gr_complex)) # Output signature
```

```
      #src = gr.sig_source_c(if_rate, gr.GR_SIN_WAVE, 2e3, 1, 0)
        #src = gr.file_source (gr.sizeof_float, filename, True)
        src = gr.wavfile_source (filename, True)
      gain = gr.multiply_const_ff (30)
        fmtx = blks2.wfm_tx (audio_rate, if_rate)#, max_dev=75e3,
tau=75e-6)
      self.connect(src, gain, fmtx, self)
      #self.connect(src, self)

class fm_tx_block(gr.top_block):
    def __init__(self):
      gr.top_block.__init__(self)

        parser = OptionParser (option_class=eng_option)
        parser.add_option("-f", "--freq", type="eng_float",
default=88e6,
                          help="set Tx frequency to FREQ [required]",
metavar="FREQ")
        (options, args) = parser.parse_args ()

        if len(args) != 0:
            parser.print_help()
            sys.exit(1)
      if options.freq is None:
            sys.stderr.write("fm_tx4: must specify frequency with -f
FREQ\n")
            parser.print_help()
            sys.exit(1)

        # ----------------------------------------------------------------
----
        # Set up constants and parameters

        self.u = usrp.sink_c ()       # the USRP sink (consumes
samples)

        self.dac_rate = self.u.dac_rate()                    # 128
MS/s
        self.usrp_interp = 400
        self.u.set_interp_rate(self.usrp_interp)
        self.usrp_rate = self.dac_rate / self.usrp_interp    # 320
kS/s
        self.sw_interp = 10
        self.audio_rate = self.usrp_rate / self.sw_interp    # 32 kS/s

        # determine the daughterboard subdevice we're using
        tx_subdev_spec = usrp.pick_tx_subdevice(self.u)

        m = usrp.determine_tx_mux_value(self.u, tx_subdev_spec)
        #print "mux = %#04x" % (m,)
        self.u.set_mux(m)
        self.subdev = usrp.selected_subdev(self.u, tx_subdev_spec)
        print "Using TX d'board %s" % (self.subdev.side_and_name(),)

        self.subdev.set_gain(self.subdev.gain_range()[1])    # set max
Tx gain
        if not self.set_freq(options.freq):
            freq_range = self.subdev.freq_range()
            print "Failed to set frequency to %s.  Daughterboard
supports %s to %s" % (
                eng_notation.num_to_str(options.freq),
                eng_notation.num_to_str(freq_range[0]),
                eng_notation.num_to_str(freq_range[1]))
            raise SystemExit
        self.subdev.set_enable(True)                         # enable
transmitter

        # Instantiate 1 NBFM channels
        t = pipeline("am_rec_data.wav", self.audio_rate,
self.usrp_rate)

        gain = gr.multiply_const_cc (32e3)

        # connect it all
        self.connect (t, gain, self.u)



    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool

        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital up converter.  Finally, we
feed
        any residual_freq to the s/w freq translater.
        """

        r = self.u.tune(self.subdev.which(), self.subdev, target_freq)
        if r:
            print "r.baseband_freq =",
eng_notation.num_to_str(r.baseband_freq)
            print "r.dxc_freq      =",
eng_notation.num_to_str(r.dxc_freq)
            print "r.residual_freq =",
eng_notation.num_to_str(r.residual_freq)
            print "r.inverted      =", r.inverted
            # Could use residual_freq in s/w freq translator
            return True

        return False

if __name__ == '__main__':
    try:
      fm_tx_block().run()
    except KeyboardInterrupt:
      pass
```

B3: fhop_trx.py (Spectrum scan for holes by hopping)

```
#!/usr/bin/env python
#
# Copyright 2010 Arnab Banik
# The Pennsylvania State University
#
# fhop_rx.py - Frequency Hopping receiver module
#
# AUTHOR: Arnab Banik
# DATE: 18th October, 2010
# DESCRIPTION: (import <filename>; help(<filename>)
"""
Demonstration of frequency hopping using the USRP1 hardware. Requires
the RX Basic module in locaiton A.

TODO:: <<< Add how it works later>>>
"""
#---------------------------------------------------------------------

from gnuradio import gr, gru, audio, optfir, eng_notation
from gnuradio import usrp
from gnuradio import blks2
from optparse import OptionParser
from gnuradio.eng_option import eng_option
import math
import sys
import random
import time
import timer

class pipeline(gr.hier_block2):
    def __init__(self, filename, audio_rate2, if_rate):

      gr.hier_block2.__init__(self, "pipeline",
                      gr.io_signature(0,0,0),                  # Input
signature
                      gr.io_signature(1,1,gr.sizeof_gr_complex)) #
Output signature
```

```
      src = gr.wavfile_source (filename, True)
        gain = gr.multiply_const_ff (30)
        fmtx = blks2.wfm_tx (audio_rate2, if_rate)
      self.connect(src, gain, fmtx, self)

class freq_hop():
    def __init__(self):
        # Parameters
        self.fir_decim = 10 # Decimation rate of the FIR filer
        self.debug_mode = 1 # Enable/Disable timing
        self.lock_state = 1
      self.freq_flag = 0
      self.freq1 = 0
      self.freq2 = 0

        self.spin_period = 20 # Spin period in ms
        self.hop_period = 1000 # Hop period in ms
        self.signal_threshold = 5 # Signal threshold level
        self.channel = 0 # Initial hop channel

        self.hop_set = range(88000000, 88000030, 1)
        self.freq = self.hop_set[self.channel]

      parser = OptionParser(option_class=eng_option)
      parser.add_option("-R", "--rx-subdev-spec", type="subdev",
default="A",
                          help="select USRP Rx side A or B
(default=A)")
      parser.add_option("-T", "--tx-subdev-spec", type="subdev",
default="A",
                          help="select USRP Tx side A or B
(default=A)")
      (options,args) = parser.parse_args()
      if len(args) != 0:
          parser.print_help()
          sys.exit(1)

        # Setup of USRP RX
        self.rx = usrp.source_c()
      self.u = usrp.sink_c()

        adc_rate = self.rx.adc_rate()  # For USRP1, 64 MHz
        self.dac_rate = self.u.dac_rate() # For USRP1, 128 MHz
      usrp_decim = 200
      self.usrp_interp = 400
        self.rx.set_decim_rate(usrp_decim)
      self.u.set_interp_rate(self.usrp_interp)
        usrp_rate1 = adc_rate / usrp_decim # 320 KHz
      self.usrp_rate2 = self.dac_rate/self.usrp_interp
      chanfilt_decim = 1
        demod_rate = usrp_rate1 / chanfilt_decim     # 320 kHz
        audio_decimation = 10
      self.sw_interp = 10
        audio_rate1 = demod_rate / audio_decimation  # 32 kHz
      self.audio_rate2 = self.usrp_rate2/ self.sw_interp  # 32 kHz

        rx_subdev = usrp.pick_rx_subdevice(self.rx)
        mux1 = usrp.determine_rx_mux_value(self.rx, rx_subdev)
        self.rx.set_mux(mux1)
        self.subdev1 = usrp.selected_subdev(self.rx, rx_subdev)
        print "Using RX Board: %s" % (self.subdev1.side_and_name(),)

      tx_subdev = usrp.pick_tx_subdevice(self.u)
        mux2 = usrp.determine_tx_mux_value(self.u, tx_subdev)
        self.u.set_mux(mux2)
        self.subdev2 = usrp.selected_subdev(self.u, tx_subdev)
        print "Using TX Board: %s" % (self.subdev2.side_and_name(),)


        # Set initial frequency and gain
        self.subdev1.set_gain( self.subdev1.gain_range()[1] ) # Sets
to max gain
        self.subdev2.set_gain( self.subdev2.gain_range()[1] ) # Sets
to max gain
        r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
      y = self.u.tune(self.subdev2.which(), self.subdev2, self.freq)
        if r and y:
            print "Baseband", r.baseband_freq
            print "DXC:", r.dxc_freq
            print "Residual:", r.residual_freq
            print "Inverted:", r.inverted
        else:
            print "Frequency out of range!"
            return

        # Enable transmitter
        self.subdev1.set_enable(True)
        self.subdev2.set_enable(True)

        # Generate Filter
        fir_taps = gr.firdes.low_pass( 1.0, usrp_rate1, 15e3, 10e3,
gr.firdes.WIN_HANN )
        self.lpf_fir = gr.fir_filter_ccf( self.fir_decim, fir_taps )



        # Probe block
        self.magnitude = gr.probe_avg_mag_sqrd_c(-60,0.001)

        # Processing Block
        self.tb = gr.top_block()
        self.tb.connect(self.rx, self.lpf_fir)
        self.tb.connect(self.lpf_fir, self.magnitude)

     volume = 1
     chan_filt_coeffs = optfir.low_pass (1,            # gain
                                            usrp_rate1,   # sampling
rate
                                            80e3,         # passband
cutoff
                                            115e3,        # stopband
cutoff
                                            1.0,         # passband
ripple
                                            60)          # stopband
attenuation

        self.chan_filt = gr.fir_filter_ccf (chanfilt_decim,
chan_filt_coeffs)
     self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)

        #self.am_demod = gr.complex_to_mag()
     self.t = pipeline("am_rec_data.wav",self.audio_rate2,
self.usrp_rate2)
     self.gain = gr.multiply_const_cc (32e3)
        self.volume_control = gr.multiply_const_ff(volume)
     self.audio_sink_FM = audio.sink(audio_rate1,"", False)
     self.file_sink = gr.wavfile_sink ("chann3.wav",2,audio_rate1,16)
        self.tb.start()


    def hop_lock(self, timer):
        if self.debug_mode:
            print self.magnitude.level()
         print self.lock_state
         print self.freq
        # Is this a new freq?
        # Lock State
        # -- 0 - timeout of current hop, need to inc hop_freq
        # -- 1 - Searching for new frequency
     # Determine if signal above threshold
        if self.magnitude.level() < self.signal_threshold and
self.freq_flag == 0:
            if self.debug_mode:
                print "Freq1 Locked"
         timer.Interval( self.hop_period )
         self.freq1=self.channel
         self.freq_flag = 1
         return

     if self.freq_flag == 1 and self.magnitude.level() <
self.signal_threshold:
         self.freq2 = self.channel
         if self.debug_mode:
          print "Freq2 Locked"
         self.freq_flag = 2
         return

     if self.freq_flag == 2 and self.magnitude.level() <
self.signal_threshold:
         self.freq2 = self.channel
         if self.debug_mode:
          print "Freq2 Locked"
         self.freq_flag = 3
         return

     if self.freq_flag == 3 and self.magnitude.level() <
self.signal_threshold:
         self.freq2 = self.channel
         if self.debug_mode:
          print "Freq2 Locked"
         self.freq_flag = 4
         return


     if self.freq_flag == 4:
         self.channel = (self.freq1 + self.freq2)/2
         self.freq = self.hop_set[self.channel]
            r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
            y = self.u.tune(self.subdev2.which(), self.subdev2,
self.freq)
         if r and y: pass
            else: print "Freq Out of Range"

         self.tb.stop()
            print "tb stopped"
         self.tb.wait()
            print "tb waiting"
         self.tb.connect(self.rx, self.chan_filt, self.guts,
self.volume_control, self.audio_sink_FM)
         self.tb.connect(self.volume_control,self.file_sink)
         self.tb.connect (self.t,self.gain,self.u)

            print "tb new connect"
         self.tb.start()
            print "tb started again"
         tmr.Stop()
         return

     if self.lock_state == 1:
            self.hop_inc()
            timer.Interval( self.spin_period )
            return

    def hop_inc(self):
        self.channel += 1
        if self.channel >= len(self.hop_set):
            self.channel = 0
      self.freq = self.hop_set[self.channel]
        r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
        if r: pass
        else: print "Freq Out of Range"

if __name__ == '__main__':
    try:
        prog = freq_hop()
        tmr = timer.EventTimer( 500, prog.hop_lock )
        tmr.Start()
        time.sleep(100)
        tmr.Stop()
    except KeyboardInterrupt:
        tmr.Stop()
```

B4: fhop_trx1.py (Spectrum scan for relay beacon)

```
#!/usr/bin/env python
#
# Copyright 2010 Arnab Banik
# The Pennsylvania State University
#
# fhop_rx.py - Frequency Hopping receiver module
#
# AUTHOR: Arnab Banik
# DATE: 18th October, 2010
# DESCRIPTION: (import <filename>; help(<filename>)
"""
Demonstration of frequency hopping using the USRP1 hardware. Requires
the RX Basic module in locaiton A.

TODO:: <<< Add how it works later>>>
"""
#---------------------------------------------------------------------

from gnuradio import gr, gru, audio, optfir, eng_notation
from gnuradio import usrp
from gnuradio import blks2
from optparse import OptionParser
from gnuradio.eng_option import eng_option
import math
import sys
import random
import time
import timer
```

```
class pipeline(gr.hier_block2):
    def __init__(self, filename, audio_rate2, if_rate):

     gr.hier_block2.__init__(self, "pipeline",
                      gr.io_signature(0,0,0),                  # Input
signature
                      gr.io_signature(1,1,gr.sizeof_gr_complex)) #
Output signature

     src = gr.wavfile_source (filename, True)
        gain = gr.multiply_const_ff (30)
        fmtx = blks2.wfm_tx (audio_rate2, if_rate)
     self.connect(src, gain, fmtx, self)

class freq_hop():
    def __init__(self):
        # Parameters
        self.fir_decim = 10 # Decimation rate of the FIR filer
        self.debug_mode = 1 # Enable/Disable timing
        self.lock_state = 1
     self.freq_flag = 0
     self.freq1 = 0
     self.freq2 = 0

        self.spin_period = 20 # Spin period in ms
        self.hop_period = 1000 # Hop period in ms
        self.signal_threshold = 30 # Signal threshold level
        self.channel = 0 # Initial hop channel

        self.hop_set = range(88000000, 88000030, 1)
        self.freq = self.hop_set[self.channel]

     parser = OptionParser(option_class=eng_option)
     parser.add_option("-R", "--rx-subdev-spec", type="subdev",
default="B",
                          help="select USRP Rx side A or B
(default=A)")
     parser.add_option("-T", "--tx-subdev-spec", type="subdev",
default="B",
                          help="select USRP Tx side A or B
(default=A)")
     (options,args) = parser.parse_args()
     if len(args) != 0:
         parser.print_help()
         sys.exit(1)

        # Setup of USRP RX
        self.rx = usrp.source_c()
     self.u = usrp.sink_c()

        adc_rate = self.rx.adc_rate()  # For USRP1, 64 MHz
        self.dac_rate = self.u.dac_rate() # For USRP1, 128 MHz
```

```
      usrp_decim = 200
      self.usrp_interp = 400
        self.rx.set_decim_rate(usrp_decim)
      self.u.set_interp_rate(self.usrp_interp)
        usrp_rate1 = adc_rate / usrp_decim # 320 KHz
      self.usrp_rate2 = self.dac_rate/self.usrp_interp
      chanfilt_decim = 1
        demod_rate = usrp_rate1 / chanfilt_decim     # 320 kHz
        audio_decimation = 10
      self.sw_interp = 10
        audio_rate1 = demod_rate / audio_decimation  # 32 kHz
      self.audio_rate2 = self.usrp_rate2/ self.sw_interp  # 32 kHz

        rx_subdev = usrp.pick_rx_subdevice(self.rx)
        mux1 = usrp.determine_rx_mux_value(self.rx, rx_subdev)
        self.rx.set_mux(mux1)
        self.subdev1 = usrp.selected_subdev(self.rx, rx_subdev)
        print "Using RX Board: %s" % (self.subdev1.side_and_name(),)

      tx_subdev = usrp.pick_tx_subdevice(self.u)
        mux2 = usrp.determine_tx_mux_value(self.u, tx_subdev)
        self.u.set_mux(mux2)
        self.subdev2 = usrp.selected_subdev(self.u, tx_subdev)
        print "Using TX Board: %s" % (self.subdev2.side_and_name(),)


        # Set initial frequency and gain
        self.subdev1.set_gain( self.subdev1.gain_range()[1] ) # Sets
to max gain
        self.subdev2.set_gain( self.subdev2.gain_range()[1] ) # Sets
to max gain
        r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
      y = self.u.tune(self.subdev2.which(), self.subdev2, self.freq)
        if r and y:
            print "Baseband", r.baseband_freq
            print "DXC:", r.dxc_freq
            print "Residual:", r.residual_freq
            print "Inverted:", r.inverted
        else:
            print "Frequency out of range!"
            return

        # Enable transmitter
        self.subdev1.set_enable(True)
        self.subdev2.set_enable(True)

        # Generate Filter
        fir_taps = gr.firdes.low_pass( 1.0, usrp_rate1, 15e3, 10e3,
gr.firdes.WIN_HANN )
        self.lpf_fir = gr.fir_filter_ccf( self.fir_decim, fir_taps )
        # Probe block
        self.magnitude = gr.probe_avg_mag_sqrd_c(-60,0.01)

        # Processing Block
        self.tb = gr.top_block()
        self.tb.connect(self.rx, self.lpf_fir)
        self.tb.connect(self.lpf_fir, self.magnitude)

      volume = 0
      chan_filt_coeffs = optfir.low_pass (1,           # gain
                                             usrp_rate1,   # sampling
rate
                                             80e3,         # passband
cutoff
                                             115e3,        # stopband
cutoff
                                             1.0,         # passband
ripple
                                             60)          # stopband
attenuation

        self.chan_filt = gr.fir_filter_ccf (chanfilt_decim,
chan_filt_coeffs)
      self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)

        #self.am_demod = gr.complex_to_mag()
      self.t = pipeline("am_rec_data.wav",self.audio_rate2,
self.usrp_rate2)
      self.gain = gr.multiply_const_cc (2e3)
        self.volume_control = gr.multiply_const_ff(volume)
      self.audio_sink_FM = audio.sink(audio_rate1,"", False)
      #self.file_sink = gr.wavfile_sink ("chann3.wav",2,audio_rate1,16)
        self.tb.start()


    def hop_lock(self, timer):
        if self.debug_mode:
            print self.magnitude.level()
         print self.lock_state
         print self.freq
        # Is this a new freq?
        # Lock State
        # -- 0 - timeout of current hop, need to inc hop_freq
        # -- 1 - Searching for new frequency
      # Determine if signal above threshold
        if self.magnitude.level() > self.signal_threshold and
self.freq_flag == 0:
            if self.debug_mode:
                print "Freq1 Locked"
         timer.Interval( self.hop_period )
```

```
          self.freq1=self.channel
          self.freq_flag = 1
          return

      if self.freq_flag == 1 and self.magnitude.level() >
self.signal_threshold:
          self.freq2 = self.channel
          if self.debug_mode:
           print "Freq2 Locked"
          self.freq_flag = 2
          return


      if self.freq_flag == 2:
          self.channel = (self.freq1 + self.freq2)/2
          self.freq = self.hop_set[self.channel]
            r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
            y = self.u.tune(self.subdev2.which(), self.subdev2,
self.freq)
          if r and y: pass
            else: print "Freq Out of Range"

          self.tb.stop()
            print "tb stopped"
          self.tb.wait()
            print "tb waiting"
          self.tb.connect(self.rx, self.chan_filt, self.guts,
self.volume_control, self.audio_sink_FM)
          #self.tb.connect(self.volume_control,self.file_sink)
          self.tb.connect (self.t,self.gain,self.u)

            print "tb new connect"
          self.tb.start()
            print "tb started again"
          tmr.Stop()
          return

      if self.lock_state == 1:
            self.hop_inc()
            timer.Interval( self.spin_period )
            return

    def hop_inc(self):
        self.channel += 1
        if self.channel >= len(self.hop_set):
            self.channel = 0
      self.freq = self.hop_set[self.channel]
        r = self.rx.tune(self.subdev1.which(), self.subdev1,
self.freq)
        if r: pass
        else: print "Freq Out of Range"
```

```
if __name__ == '__main__':
    try:
        prog = freq_hop()
        tmr = timer.EventTimer( 500, prog.hop_lock )
        tmr.Start()
        time.sleep(100)
        tmr.Stop()
    except KeyboardInterrupt:
        tmr.Stop()
```

B5: timer.py

```
#!/usr/bin/env python
#
# Copyright 2010 Arnab Banik
# The Pennsylvania State University
#
# timer.py - Implementation of a continuous event timer
#
# AUTHOR: Arnab Banik
# DATE: 10th October, 2010
# DESCRIPTION: (use: import <file>; help(file) )
"""
Uses the threading module to implement a continuous timer which
calls a function after its timout, repeatedly until stopped by Stop().

USAGE:
-- Import Module
   import timer

-- Create TimerClass
   timer = EventTimer( interval_in_ms, function_to_call )
     interval is milliseconds between function calls
     function to call must be in the current scope of the class

-- Start the timer
   timer.Start()

-- Stop the timer
   timer.Stop()

-- NOTES:
   the main thread of execution (the one which instanciates the
   EventTimer class) must not exit. Call time.sleep(xx) to keep it
   alive if neccesary.
"""
#-------------------------------------------------------------------

import threading
import time

class EventTimer():
    def __init__(self, interval, fcn_call):
        self.event = threading.Event()
        self.thread = threading.Thread( target=self._fcn )
        # I subtract 4ms from the interval to account for the time it
        #takes to run self._fcn -- it should be pretty accurate +/-
1ms
        self.interval = interval/1000.0 - 0.004
        self.function = fcn_call
        self.old_time = time.time()

    def Start(self, interval=-1):
        if interval >= 0:
            self.interval = interval
        self.thread.start()

    def Stop(self):
        self.event.set()

    def Interval(self, period):
        self.interval = period/1000.0 - 0.004

    def _fcn(self):
        while True:
            self.event.wait(self.interval)
            if self.event.isSet():
                break
            self.old_time = self.function(self)

def _foo(t):
    now = time.time()
    global dif;
    dif.append( now - t.oldtime )
    print "Exe: %f -- Avg: %f" % ((now - t.oldtime),
sum(dif)/len(dif))
    return now

if __name__ == '__main__':
    dif = []
    t = EventTimer(100, _foo)
    t.Start()
    time.sleep(100)
    t.Stop()
```

VITAE

ARNAB KUMAR BANIK

EDUCATION **Bachelor of Science in Electrical Engineering (with Honors)** Expected Graduation: December, 2010
Minor in Nanotechnology
Schreyer Honors College
The Pennsylvania State University, University Park, PA

Dean's List: *Six* out of eight semesters

Advanced Engineering courses:

Communication systems I, II
Computer networking
Software defined radio
Nems/Mems
Nanophotonics
Wireless Communication
Microwave Engineering
Engineering Senior Design
Semiconductor IC Fabrication

THESIS: **Interference reduction with relay beacon in cognitive radio networks**
(Using a practical software-defined radio environment)
Thesis Supervisor: Dr. Sven G. Bilén
Honors Advisor: Dr. Jeffrey Louis Schiano

EXPERIENCE: **Arris Inc., State College, PA (2nd August,10 – till end of Fall,2010)**

- Characterizing the linear performance of devices (Javelin in present context) in the prescribed bandwidth and also to analyze their gain, flatness, surge resistance and return loss stability particularly at those higher frequencies.

Schlumberger, Kuwait (summer, 2007)

- Completed 4-week internship with primary focus on troubleshooting and diagnosis
- Worked on quality check of Gamma ray detector device, Neutron detector device, MDT tools etc in the lab. Assembled WAFE (Wireline Acquisition Front End) machine from begin to end.

CLASS PROJECTS

- Routing Protocol with prediction based mobility model in Vehicular Ad Hoc Network (VANET) – using VanetMobiSim and NS-2 software.
- Completed independent studies in radio challenge group where I worked to minimize the signal loss due to harsh environment by combining relayed and direct link signals for improvement in Bit Error Rate (using MATLAB).
- Designed balanced microwave amplifier operating at 2.4 GHz using ADS.
- Currently working on two projects: one on designing an autonomous robot capable of collecting debris such as bolts, metal from aero plane and runway, tracking GPS locations; and the other is to fabricate an N-MOS IC chip from a wafer.
- Build a digital wireless communication network to transfer data from one computer to another with a full duplex scheme.

ACTIVITIES & HONORS
Etta Kappa Nu (HKN) Association
Tau Beta Pi Engineering Honor Society
National Society of Collegiate Scholars (NSCS), Washington
Alpha Lambda Delta National academic Honors Society(2005-08)
Honors leadership team (HLT) member (2007-08)
Treasurer, International Student's Association (2007-08)

TECHNICAL SKILLS
C++
Multisim
MATLAB
AutoCAD
CCS C
Python
PicBasic Pro
ADS
VanetMobisim
NS-2
GNU Radio
LINUX