USRP based Cognitive Radio Testbed using OpenBTS

M. Tech. Dissertation

Submitted in partial fulfilment of the requirements for the degree of
Master of Technology

by

Abrar Ahmad
113310017

Supervisor

Prof. S N Merchant

Department of Electrical Engineering
IIT Bombay

June 2014

Abstract

Wireless networks are currently regulated by fixed spectrum assignment policy which results in inefficient utilization of the spectrum. In the last few years there has been a drastic increase in the mobile services, which in turn has increased the demand for limited radio spectrum. Hence there is a need to change the conventional static spectrum assignment policy and make efficient utilization of spectrum. Cognitive Radio (CR) emerged as a new paradigm to address the spectrum underutilization problem. CR enables opportunistic use of the radio-frequency spectrum by allowing Secondary/Un-licensed users to utilize licensed bands under the condition that they should not cause the interference with the Primary/Licensed users. Secondary users utilize the detected free bands and leave them when the corresponding primary radio emerges.

A USRP based two frequency CR testbed and a four frequency CR testbed has been developed using GNURadio and OpenBTS to demonstrate these properties of cognitive radio. Primary and secondary users are made to coexist with no effect on the primary radio. Energy detection spectrum sensing technique is used to detect the free bands also known as the spectrum holes. We demonstrate that secondary users utilize these spectrum holes and vacate them whenever primary users want to use them.

Acknowledgement

This thesis wouldnt have been possible without the help of a few people who have contributed in one way or the other to the completion of this research. Firstly I would like to express my sincere gratitude to my guide Prof. S. N. Merchant for his persistent guidance, support, encouragement and vital suggestions during this research work. I thank him for introducing me to the world of wireless communication and cognitive radio. Apart from his technical input I thank him for his motivation in the project to make it an enjoyable learning experience. I would also like to thank Prof V. M. Gadre for his support and guidance.

I would like to thank my project partner Swrangsar Basumatary who helped me in resolving issues that came up in this research. I would also like to thank my parents for their immense support without which this wouldnt have been possible. I cannot thank enough, almighty God who provided me with the strength to make this possible. Also, I would like to mention SPANN lab members for giving their resources and time in enabling to complete my thesis.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Due to the rapid increase of mobile phones and other wireless communication devices, there is a need for efficient utilization of the available radio spectrum. The Spectrum Policy Task Force, a group under the Federal Communications Commission (FCC) in the United States, published a report in 2002 saying [2]:

> "In many bands, spectrum access is a more significant problem than physical scarcity of spectrum, in large part due to legacy command-and-control regulation that limits the ability of potential spectrum users to obtain such access."

If we scan the spectrum in metropolitan cities which are heavily used regions, we find that some frequency bands are unoccupied most of the time[13]. These are referred to as spectrum holes. A spectrum hole is a band of frequencies assigned to a primary user, but, at a particular time and specific geographic location, the band is not being utilized by that user[6].

This problem of inefficient utilization of spectrum can be solved by allowing secondary users which are non licensed, to access these spectrum holes. Cognitive radio which includes software defined radio, is a means to accomplish this by utilizing these spectrum holes intelligently and efficiently[8][4][11]. It uses one of the spectrum sensing techniques to identify the spectrum holes in the radio spectrum.

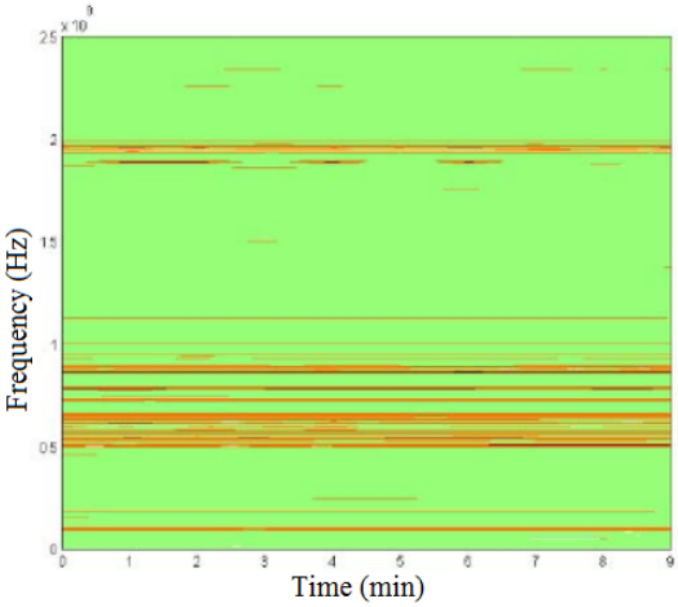

Figure 1.1: Frequency usage of Spectrum Band

1.2 Cognitive Radio

A cognitive radio is an intelligent radio whose primary objective is efficient utilization of the radio spectrum. It can be programmed and configured dynamically. It works on the principle of understanding-by-building to learn from the surrounding environment and adapt to changes in the RF stimuli by making corresponding changes in operating parameters. The transceiver is designed to find an unoccupied channel in the vicinity and utilize it for transmission. It enables coexistence of primary licensed users and secondary unlicensed users. Whenever a primary user wants to occupy the channel which is currently in use by secondary users, it finds some other unoccupied channel in the vicinity and secondary users migrate seamlessly to this new channel thus vacating the previously used channel for primary users.

1.3 Contribution of thesis

An experimental setup is developed which demonstrates the presence of secondary users along with primary users in the existing GSM network and utilizing the already existing resources there by increasing the total mobiles in the network.

1. A two frequency band cognitive system is developed where secondary users migrate to frequency $f2$ if frequency $f1$ is occupied an viceverca.

2. A two frequency system is extended to a four frequency system where we demonstrate that primary users are occupying two bands out of these four and secondary users occupy one out of the other two free bands.

3. We have used energy detection spectrum sensing technique and CUSUM peak detection technique to detect the presence of primary users. Band occupied by secondary users is continuously monitored to check if primary users are trying to occupy that band and as soon as the request from primary users is detected a new free band is found out in the vicinity and utilized by secondary users for transmission there by vacating the band for primary users .

1.4 ORGANIZATION

The rest of this document is organized as follows. Chapter 2 briefly describes the GSM architecture and its Um interface. Chapter 3 gives a literature survey on Universal Software Radio Peripheral (USRP N210) the hardware used in this project. Literature survey done on the GNU Radio software package and OpenBTS software is described in Chapter 4 and 5 respectively. Chapter 6 covers spectrum sensing techniques to detect the presence of primary users in the channel. Chapter 7 covers an implementation of cognitive radio using GNU Radio and OpenBTS. It describes the experimental setup for our project in the beginning followed by detailed description of what we have achieved in this project along with a flow chart of our work. The final chapter of this thesis is the conclusion of our project followed by future work.

Chapter 2

Overview of traditional GSM networks

2.1 What is GSM?

GSM, or Global System for Mobile Communications, is a European standard for the Mobile telecommunications and it is considered as one of the most popular standard worldwide.There are thirteen different frequency bands defined in GSM. However, the 850 MHz, 900 MHz, 1800 MHz, and 1900 MHz bands are the most commonly used. The frequency bands employed within each of the four ranges are similarly organized. They differ essentially only in the frequencies, such that various synergy effects can be taken advantage of; hence, here we give some details only for the usage in the 900 MHz band.

In the 900 MHz band, a total of 70 MHz bandwidth is allocated, two 25-MHz frequency bands for uplink and downlink and a 20 MHz unused guard band between them. The MS transmits in the 890 to 915 MHz range (uplink) and the BTS transmits in the 935 to 960 MHz (downlink) band. This corresponds to 124 duplex channels, where each channel within a BTS is referred to as an Absolute Radio Frequency Channel Number (ARFCN). This number describes a pair of frequencies, one uplink and one downlink, and is given a channel index between C0 and C123, with C0 designated as the beacon channel. An ARFCN could be used to calculate the exact frequency (in MHz) of the radio channel. In the GSM 900 band, this is computed by the following equations:

$$F_{uplink}(n) = 890 + 0.2 * n \qquad 1 \leq n \leq 124$$
$$F_{downlink}(n) = F_{uplink}(n) + 45 \qquad 1 \leq n \leq 124$$

Similar formulae are also defined for the other GSM frequency bands.

The principle component groups of a GSM network are as follows:

- The Mobile Station (MS)
- The Base Station System (BSS)
- The Network Switching System(NSS)

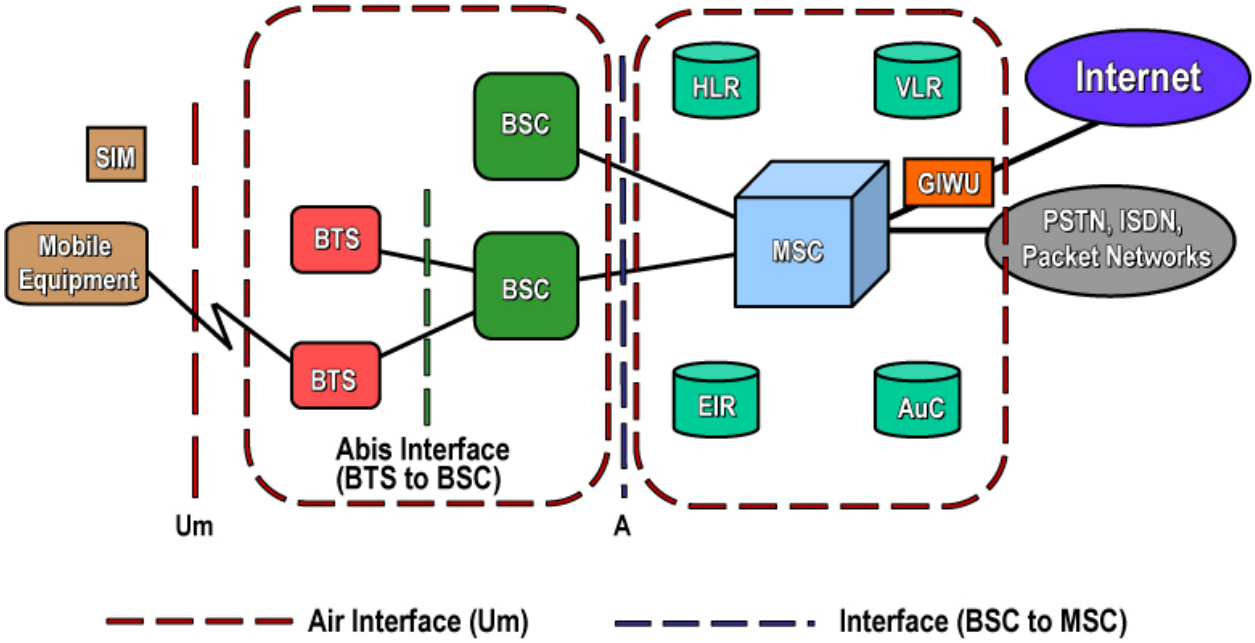

Figure 2.1: The conventional GSM architecture.
Source: `http://www.hill2dot0.com/wiki/index.php?title=Image:G2407_GSM-Architecture.jpg` [Accessed on Oct 22, 2013]

2.1.1 Mobile Station

The MS consists of two parts, the Mobile Equipment (ME) and Subscriber Identity module (SIM). The ME has an identity number called the International Mobile Equipment Identity

(IMEI) associated with it, which is unique for that particular device and permanently stored in it.The SIM card consists the International Mobile Subscriber Identity(IMSI) number which is used to identify the subscriber to the system. The IMEI and the IMSI are independent of each other and hence allow personal mobility.

2.1.2 Base Station System

The GSM Base Station System is the equipment located at a cell site. It comprises of a combination of digital and RF equipment. The BSS provides the link between the MS and the Mobile Services Switching Centre (MSC). The BSS consists mainly of:

The Base Transceiver Station(BTS) The BTS contains the RF components that provide the air interface for a particular cell. This is the part of the GSM network which communicates with the MS. The antenna is included as part of the BTS.

The Base Station Controller(BSC) The BSC provides the control for the BSS. The BSC communicates directly with the MSC. The BSC may control single or multiple BTSs. Crucial functions like radio channel link establishment, frequency hopping, and handovers from one cell to another.

2.1.3 Network Switching Subsystem

The Network Switching Subsystem includes the main switching functions of the GSM network. It also contains the databases required for subscriber data and mobility management. Its main function is to manage communications between the GSM network and other telecommunications networks. The main components of the Network Switching System are:

Mobile Services Switching Centre(MSC)

The MSC does call-switching and its overall purpose is the same as that of any telephone exchange. When the MSC provides the interface between the PSTN and the BSSs in the GSM network it will be known as a Gateway MSC. In this position it will provide the switching required for all MS originated or terminated traffic. Each MSC provides service to

MSs located within a defined geographic coverage area. One MSC is capable of supporting a regional capital with approximately one million inhabitants. The functions carried out by the MSC are: Call Processing, Operations and Maintenance Support, Internetwork Interworking and Billing

Home Location Register (HLR)

The HLR is a central database that contains the details of each mobile phone subscriber that is authorized to use the GSM core network. The IMSI of each SIM acts as a primary key to each HLR record. Each MSISDN is also a primary key to the HLR record. The HLR data is stored as long as a subscriber remains with the mobile phone operator. Data stored in the HLR against each IMSI are, GSM services that the subscriber has requested, GPRS settings to allow the subscriber to access packet services, current location of the subscriber, etc.

Visitor Location Register (VLR)

The VLR is a database of the subscribers who have roamed into the jurisdiction of the MSC which it serves. Each main base station in the network is served by exactly one VLR, hence a subscriber cannot be present in more than one VLR at a time.The data stored in the VLR has either been received from the HLR, or collected from the MS. Data stored includes: IMSI (the subscriber's identity number), authentication data, MSISDN, GSM services that the subscriber is allowed to access, the HLR address of the subscriber.

2.2 Um Interface

The Um interface is the air interface of the GSM mobile telephone standard. It is the interface between the MS and the BTS. It is called Um because it is the mobile analog to the U interface of ISDN. Um is defined in the GSM 04.xx and 05.xx series of specifications.

The layers of GSM are initially defined in GSM 04.01 Section 7 and roughly follow the OSI model. Um is defined in the lower three layers of the model.

2.2.1 Physical Layer (L1)

The Um physical layer is defined in the GSM 05.xx series of specifications, with the introduction and overview in GSM 05.01. For most channels, Um L1 transmits and receives 184-bit control frames or 260-bit vocoder frames over the radio interface in 148-bit bursts with one burst per timeslot. There are three sublayers:

Radiomodem

This is the actual radio transceiver. GSM uses 8PSK modulation with 1 bit per symbol which produces a 13/48 MHz (270.833 kHz or 270.833 K symbols/second) symbol rate and a channel spacing of 200 kHz. Since adjacent channels overlap, the standard does not allow adjacent channels to be used in the same cell. The standard defines several bands ranging from 400 MHz to 1990 MHz.GSM is frequency duplexed, meaning that the network and MS transmit on different frequencies, allowing the BTS to transmit and receive at the same time. Transmission from the network to the MS is called the "downlink" and that from the MS to the network is called the "uplink". The GSM uplink and downlink bands are separated by 45 or 50 MHz. Uplink/downlink channel pairs are identified by an index called the ARFCN. Within the BTS, these ARFCNs are given arbitrary carrier indexes C0..Cn-1, with C0 designated as a Beacon Channel and always operated at constant power. The radio channel is time-multiplexed into 8 timeslots, each with a duration of 156.25 symbol periods. These 8 timeslots form a frame of 1,250 symbol periods. The capacity associated with a single timeslot on a single ARFCN is called a physical channel (PCH) and referred to as "CnTm" where n is a carrier index and m is a timeslot index (0-7).Each timeslot is occupied by a radio burst with a guard interval, two payload fields, tail bits, and a midamble.

Multiplexing and Timing

GSM uses TDMA to subdivide each radio channel into as many as 16 traffic channels or as many as 64 control channels. The multiplexing patterns are defined in GSM 05.02.Each physical channel is time-multiplexed into multiple logical channels according to the rules of GSM 05.02. Traffic channel multiplexing follows a 26-frame (0.12 second) cycle called a "multiframe". Control channels follow a 51-frame multiframe cycle. The C0T0 physical

channel carries the synchronization channel(SCH), which encodes the timing state of the BTS to facilitate synchronization to the TDMA pattern.

FEC Coding

The coding sublayer provides forward error correction. As a general rule, each GSM channel uses a block parity code (usually a Fire code), a rate-1/2, 4th-order convolutional code and a 4-burst or 8-burst interleaver.

2.2.2 Data Link Layer(L2)

The Um data link layer, LAPDm, is defined in GSM 04.05 and 04.06. LAPDm is the mobile analog to ISDN's LAPD.

2.2.3 Network layer(L3)

The Um network layer is defined in GSM 04.07 and 04.08 and has three sublayers. A subscriber terminal must establish a connection in each sublayer before accessing the next higher sublayer.

Radio Resource (RR) This sublayer manages the assignment and release of logical channels on the radio.

Mobility Management (MM) This sublayer authenticates users and tracks their movements from cell to cell. It is normally terminated in the VLR or HLR.

Call Control (CC) This sublayer connects telephone calls and is taken directly from ITU-T Q.931. GSM 04.08 Annex E provides a table of corresponding paragraphs in GSM 04.08 and ITU-T Q.931 along with a summary of differences between the two. The CC sublayer is terminated in the MSC.

The access order is RR, MM, CC. The release order is the reverse of that. Note that none of these sublayers terminate in the BTS itself. The standard GSM BTS operates only in layers 1 and 2.

Chapter 3

Software Defined Radio

Software defined radio also known as software radio or SDR is a radio in which some or all of the physical layer functions are software defined. Due to the exponential increase in the ways and means by which people need to communicate - data communications, voice communications, video communications, broadcast messaging, command and control communications, emergency response communications and etc, there is a constant need of modifying radio devices easily and cost efficiently. The traditional hardware radio system consist a variety of analogy elements such as mixers, filters, amplifers, converters, modulators and demodulators etc which are costly and are not flexible. Thus traditional hardware based radio devices limit cross-functionality and can only be modified only through physical intervention. This results in higher production costs and minimal flexibility. The solution to this is Software Defined Radio which is easily modifiable. Technologies such as Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP) and General-Purpose Processor (GPP) are used to build the software radio elements.

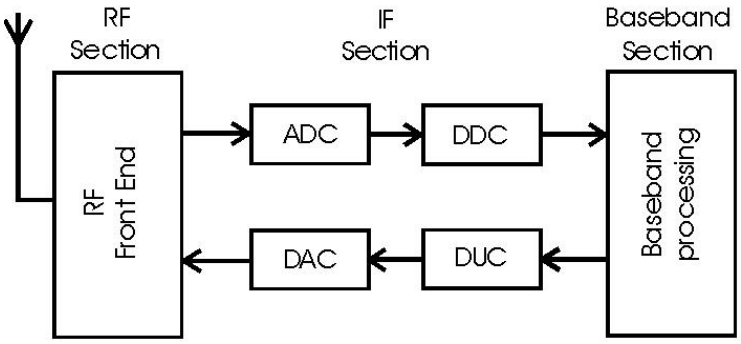

Figure 3.1: Block diagram of SDR.

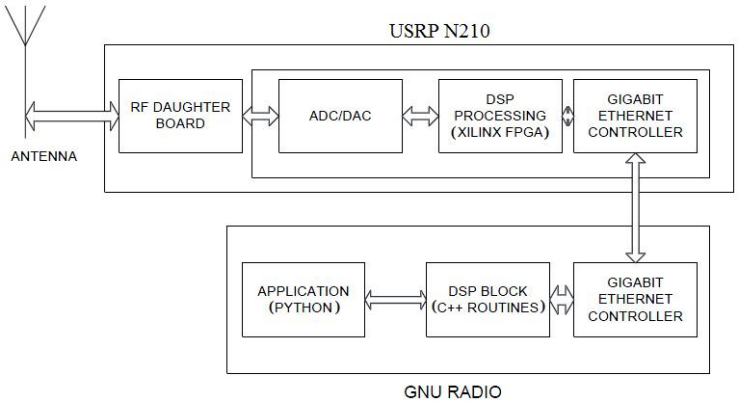

Figure 3.2: Block diagram for USRP operation with GNURadio.

The software defined radio (SDR) contains a number of basic functional blocks. The radio in genera can be divided into three basic sections, namely the front end, the IF section and the base-band section as shown in Figure 3.1. The front end section uses analogue RF circuitry and it is responsible for receiving and transmitting the signal at the frequency of operation. IF section performs the digital to analog conversion and vice versa. It also contains the signal processing like Filtering, modulation and demodulation, Digital up conversion (DUC), Digital down conversion (DDC) etc. The last stage of the radio is the baseband processor. It is at this point that the digital data is processed [10][1]. We have used GNURadio and USRP N210 to confgure the SDR to implement the Cognitive Radio test-bed.

A block diagram of a USRP-based SDR transceiver built with a GNURadio flow graph is shown in Figure 3.2. USRP kit is used as a hardware while GNURadio package is used for the baseband signal processing tasks. The next sections describe USRP kits and GNURadio package.

3.1 USRP

The USRP (Universal Software Radio Peripheral) is intended to provide a low-cost, high quality hardware platform for software radio. It is designed and marketed by Ettus Research, LLC. It is commonly used by research labs, universities, and hobbyists. The USRP platform is designed for RF applications from DC to 6 GHz. USRPs connect to a host computer through a high-speed USB or Gigabit Ethernet link, which the host-based software uses to control the USRP hardware and transmit/receive data.

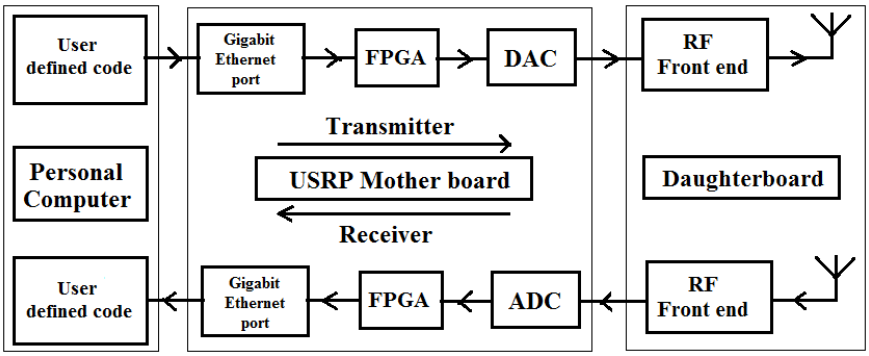

Figure 3.3: Block diagram of USRP.
Source: Kranthi Ananthula, Experimental setup of Cognitive Radio Test-Bed using Software Defined Radio, M. Tech. Dissertation, 2013.

The USRP Hardware Driver (UHD) is the official driver for all Ettus Research products. The UHD supports Linux, Mac OS X and Windows.

In this project we are using a particular model of USRP product known as the USRP N210.

3.1.1 USRP N210

The USRP N200 and N210 are the highest performing class of hardware of the USRP family of products, which enables engineers to rapidly design and implement powerful, flexible software radio systems. The N200 and N210 hardware is ideally suited for applications requiring high RF performance and great bandwidth. Such applications include physical layer prototyping, dynamic spectrum access and cognitive radio, spectrum monitoring, record and playback, and even networked sensor deployment. The Networked Series products offers MIMO capability with high bandwidth and dynamic range. The Gigabit Ethernet interface serves as the connection between the N200/N210 and the host computer. This enables the user to realize 50 MS/s of real-time bandwidth in the receive and transmit directions, simultaneously (full duplex).

3.2 GNURadio package

GNURadio is an open source software toolkit for building software defined radios. GNURadio is build with the aim to bring the code as close to the antenna as possible and thereby turn

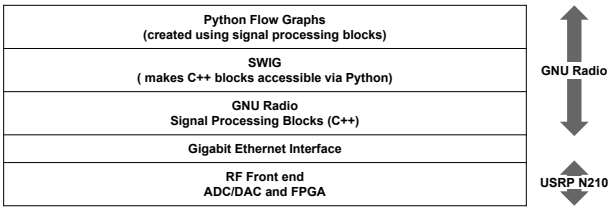

Figure 3.4: Architecture of GNU Radio

hardware problems into software problems. GNURadio has software equivalents of real world radio system components like filters, demodulators, equalizers and other building blocks for signal processing, as well as a framework for data flow control between them. We can build a software defined radio by connecting these blocks together.

Basically GNURadio does all the signal processing. Functions provided by GNURadio includes Mathematical operations, Interleaving, delay blocks, Filters, FFT blocks, Automatic Gain Control (AGC) blocks, Modulation and demodulation (FM, AM, PSK, QAM, GMSK, OFDM and etc), Interpolation and decimation. Apart from signal processing blocks, GNURadio also provides support for various signal sources and sinks, such as Signal generators, Noise generators, Pseudo random number generators, USRP source and sink, Graphical sinks, Audio source and sink, File source and sink and etc [7]. GNURadio applications are primarily written using thePython programming language.

USRP is the hardware used along with GNURadio for SDR and is either at the beginning of the flow graph (implementation of a receiver) or the end (transmitter).

Chapter 4

OpenBTS

4.1 What is OpenBTS?

OpenBTS is a Unix application that uses a software radio to present a GSM Um interface to handsets and uses a SIP softswitch or PBX to connect calls.(You might even say that OpenBTS is a simplified form of IMS that works with 2G feature-phone handsets). The combination of the global -standard GSM air interface with low cost VoIP backhaul forms the basis of a new type of cellular network that can be deployed and operated at substantially lower cost than existing technologies in many applications, especially rural cellular deployments and private cellular networks in remote areas.

4.2 The OpenBTS Application Suite

A complete OpenBTS P2.8 installation comprises several distinct applications:

OpenBTS The actual OpenBTS application, containing most of the GSM stack above the radio modem.

Transceiver The software radio modem and hardware control interface.

Asterisk The VoIP PBX in the standard public release configuration.

Smqueue The store-and-forward server for text messaging.

Subscriber Registry A database of subscriber information that replaces both the Asterisk SIP registry and the GSM Home Location Register (HLR).

Other Servers Other optional GSM services, beyond speech and text messaging, are supported through external servers.

The OpenBTS and Transceiver applications must run inside each GSM/SIP access point. The Asterisk and the subscriber registry applications are communicated through the filesystem and therefore must run on the same computer, but that computer can be remote from the access point. smqueue and the other servers can run anywhere and may have multiple instances.

4.2.1 OpenBTS

The OpenBTS application contains:

- L1 Time division multiplexing(TDM) functions (GSM 05.02)
- L1 Forward error correction(FEC) functions (GSM 05.03)
- L1 closed loop power and timing controls (GSM 05.08 and 05.10)
- L2 Link access protocol on Dm-channel (LAPDm) (GSM 04.06)
- L3 radio resource management functions (GSM 04.08)
- L3 GSM-SIP gateway for mobility management
- L3 GSM-SIP gateway for call control
- L4 GSM-SIP gateway for text messaging

The general design approach of OpenBTS is avoid implementing any function above L3, so at L3 or L4 every subprotocol of GSM is either terminated locally or translated through a gateway to some other protocol for handling by an external application. Similarly, OpenBTS itself does not contain any speech transcoding functions above the L1 FEC parts.

4.2.2 Transceiver

The transceiver application performs the radio modem functions of GSM 05.05 and manages the Gigabit Ethernet interface (USB2 interface, in case of USRP1 or older models) to the radio hardware.

4.2.3 Asterisk

OpenBTS uses a SIP switch or PBX to perform the call control functions that would normally be performed by the mobile switching center in a conventional GSM network, although in most network configurations this switching function is distributed over multiple switches. These switches also provide transcoding services. In OpenBTS P2.8 the standard SIP switch is Asterisk 1.8.

4.2.4 Subscriber Registry

OpenBTS uses a modified SIP registry as a substitute for the home location register found in a conventional GSM network. OpenBTS also relies on Asterisk for any transcoding functions.

4.2.5 Smqueue

Smqueue is a store-and-forward server that is used for text messaging in the OpenBTS system. Smqueue is required to send a text message from one MS to another, or to provide reliable delivery of text messages to an MS from any source.

4.2.6 Network Organization

In the simplest network, with a single access point, all of the applications in the suite run inside the access point on the same embedded computer. Figure 4.1 describes this.

In larger network, with more than one access points, one of the BTS can behave as a master and provide servers to the rest of them. Figure 4.2 describes a network with two access

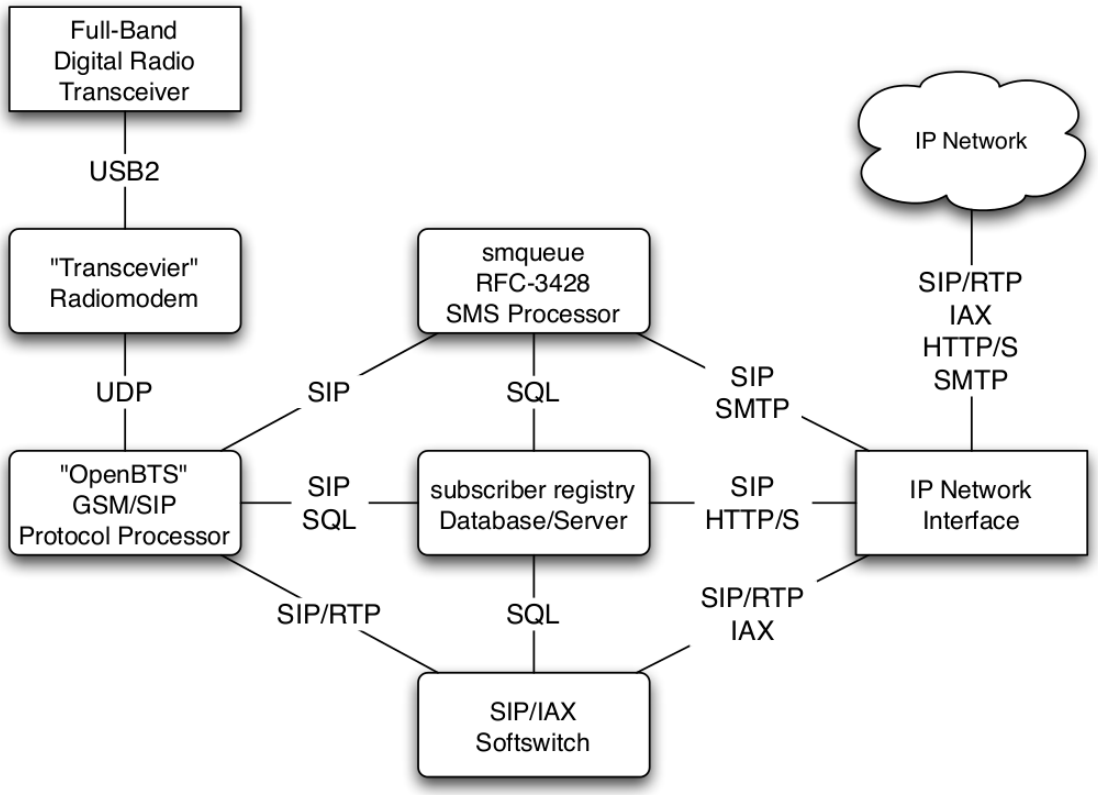

Figure 4.1: **Network Organization of OpenBTS**: The smqueue block handles SMSs. The subscriber registry database contains the details of all registered users and it maps the registered IMSIs to corresponding dialling numbers. The softswitch connects speech calls (e.g. Asterisk, FreeSwitch). The transceiver performs radio modem functions and manages the Gigabit Ethernet interface (USB2 interface, in case of USRP1 or older models) to the USRP device. The OpenBTS itself is the GSM implementation from the TDMA part of L1 up through L3 and the L3/L4 boundary. It has a SIP interface to communicate with the other blocks like smqueue, subscriber registry, etc.
Source: `https://wush.net/trac/rangepublic/attachment/wiki/WikiStart/PReleaseManual.pdf` [Accessed on Oct 22, 2013]

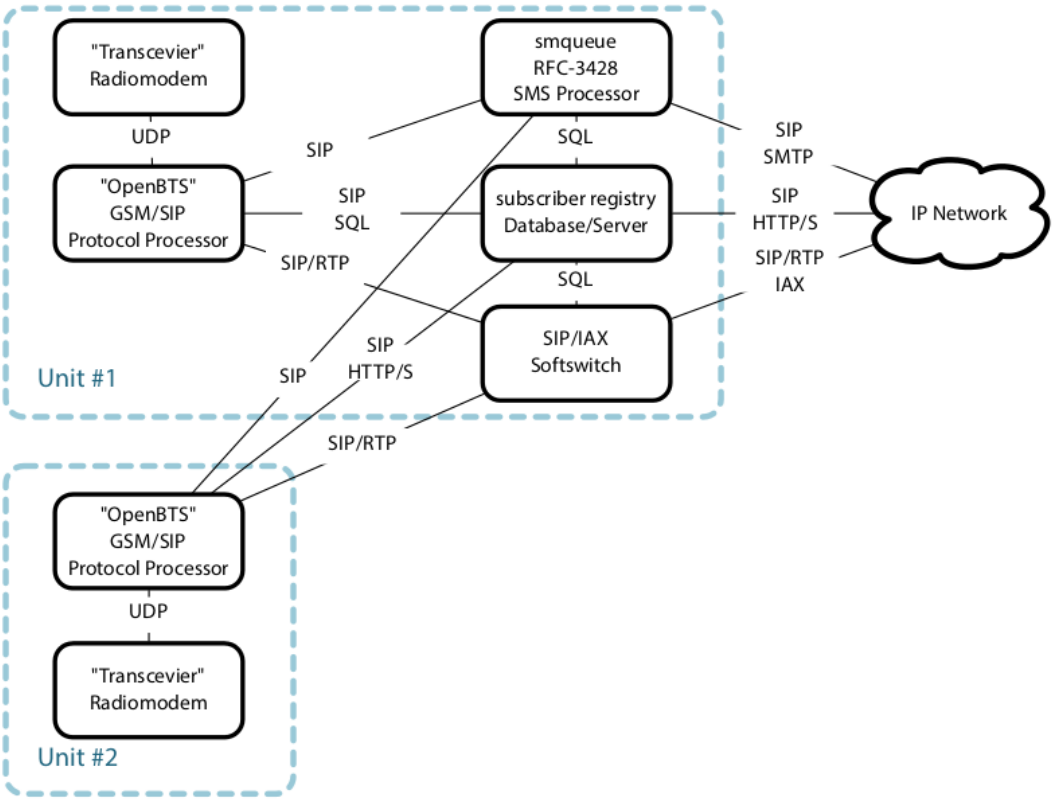

Figure 4.2: Two access points with unit #1 providing servers for both.
Source: `https://wush.net/trac/rangepublic/attachment/wiki/WikiStart/OpenBTS-4.0-Manual.pdf` [Accessed on May 27, 2014]

points where a master access points is providing servers to the other one. The Transceiver applications and the OpenBTS must run in each GSM/SIP access point. The Asterisk and the Subscriber Registry applications (SIPAuthServe) communicate via the filesystem and hence must run in the same computer, but that computer can be remote to the access point. SMQueue and other servers can run in any access point and can have multiple instances.

Chapter 5

Spectrum Sensing

Due to limited availability of spectrum resource, there is a serious impact on the emerging mobile applications. Hence there is a need to efficiently utilize the available radio spectrum. The problem right now is not the physical scarcity of the radio spectrum rather the inefficient use of the spectrum. Solution to this is cognitive radio. The major problem in cognitive radio is to detect spectrum holes which can be utilized by secondary users for communication and to enable secondary users to quit the frequency band as soon as the corresponding primary radio emerges. This technique is called spectrum sensing. Spectrum sensing is the first step to implement cognitive radio system.

There are various methods for local spectrum sensing proposed by researchers.

The following section describes three important methods:

1. Energy detection
2. Matched filter detection
3. Cyclostationarity detection

5.1 Energy Detection

Measuring the energy of a particular band is one of the simplest techniques to detect the presence of primary users in that band. It is one of the most widely used technique to

detect spectrum holes as it requires no a priori knowledge of the primary radio. Apart from this major advantage the technique is very cost efficient and less complex compared to other techniques. We calculate the energy of the received radio spectrum and this energy is compared with a predefined energy detection threshold to conclude whether primary user is present or absent in the frequency of interest. This technique is an optimal one when we have absolutely no knowledge of the user occupying the channel in advance. The following block diagram describes energy detection technique:

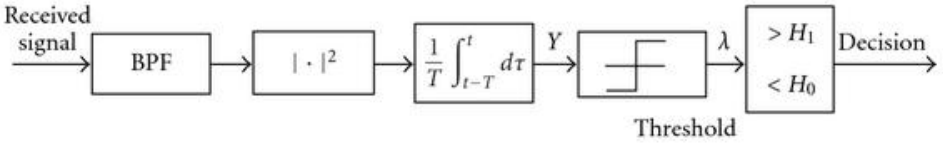

Figure 5.1: Block diagram of Energy Detection implementation.
Source: `http://www.hindawi.com/journals/ijvt/2011/630467.fig.003.jpg` [Accessed on Oct 22, 2013]

As described in the energy detection block diagram, it is basically a hypothesis testing problem with two possible hypothesis $H0$ and $H1$. Hypothesis $H1$ concludes the presence of primary users in the band of interest and hypothesis $H0$ concludes their absence. And energy detection technique is basically about distinguishing between these two hypotheses[5].

$$x(t) = n(t); \qquad H0$$
$$x(t) = hs(t) + n(t); \qquad H1$$

$x(t)$ is signal received by secondary user and $s(t)$ is primary radio signal, $n(t)$ is additive white Gaussian noise (AWGN) and h is the amplitude gain of the channel. $s(t)$ and $n(t)$ are assumed to be independent of each other. Signal detection is performed using an energy detector and compute decision statistics Y which corresponds to energy collected in observation time T and bandwidth W and comparing this statistics to a predetermined threshold γ. In our project energy detection is implemented using average periodogram analysis which is covered in later part of this chapter.

5.2 Matched filter detection

Matched filter is a linear filter used to match a particular transit waveform with the reference signal. The output is maximum when the match happens. When there is a priori knowledge of primary radio, matched filter technique is applied. Matched filter operation is equivalent to correlation operation in which the incoming signal is convolved with a filter whose impulse response is mirror and shifted version of reference signal. This output is then compared with the threshold for primary user detection. It is mathematically defined as:

$$Y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

x is the unknown signal and h is the impulse response of matched filter which is matched to reference signal for maximizing SNR.

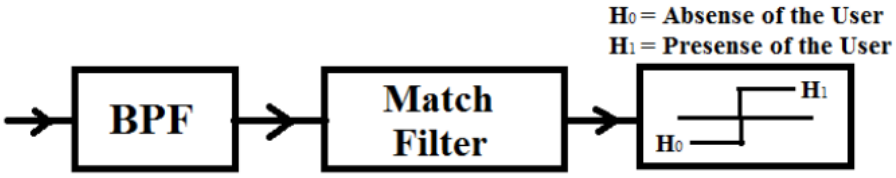

Figure 5.2: Block diagram of Matched Filter implementation.

There is a constraint on this technique. We need to have a prior information about the primary radio to perform matched filtering. However matched filter requires demodulation of primary signal which means it has information of primary radio both at the PHY layer and the MAC layer like operating frequency, modulation type, packet format bandwidth etc. But the cumbersome part is it has to achieve coherency with primary user by means of timing and carrier synchronization. This coherent detection is still achievable since primary signals have pilots, preambles etc to serve the purpose. The advantage of matched filter detection is when the information of the primary user signal is known, it is optimal detection in stationary Gaussian noise. But the performance of matched filter detection depends on the accuracy of the information of primary radio. This technique also requires cognitive radio to have dedicated receiver for every type of primary user which in turn results in complex hardware and large power consumption[14].

5.3 Cyclostationarity detection

This technique utilizes periodicity property of the received signal to detect the presence of primary users. The property of Periodicity is generally exhibited by communication signals due to sinusoidal carriers, pulse train, hopping sequences etc. Due to this underlying periodicity most of the communication signals can be modeled as cyclostationary processes. This technique can detect a primary signal with a particular modulation type even in a background of noise and other modulated signals.

Cyclostationary feature detection is robust to noise and is a better performer than energy based detection in low SNR regions. This technique also requires a priori knowledge of the primary signal. Also this technique is computationally highly complex and the sensing time is also quiete long. Due the these reasons it is less commonly used than energy based detection.

Block diagram of cyclostationary detection is given in figure 5.3 [14].

Figure 5.3: Block diagram of Cyclostationary Detection implementation.

The detection is done by finding a unique cyclic frequency of the spectral correlation function of the received signal[3]. The spectral correlation function is the Fourier transform of the cyclic autocorrelation function. The spectral correlation function is defined as:

$$S(f,\alpha) = \int_{-\infty}^{\infty} R_x^{\alpha}(\tau) e^{-2\pi f \tau} d\tau$$

Where the cyclic autocorrelation function is defined by:

$$R_x^{\alpha} = E\{x(t+\tau)x^*(t-\tau)e^{-2\pi\alpha t}\}$$

Here $x(t)$ is the signal received and α is the cyclic frequency. The spectral correlation function is also termed as cyclic spectrum.

We can also get the SCD function by considering a zero mean signal $x(t)$ whose time varying autocorrelation function $R_x(t,\tau)$ defined as [12]

$$R_x(t,\tau) = E\{x(t)x^*(t+\tau)\}$$

is periodic in time t and can be represented as a Fourier series

$$R_x(t,\tau) = \sum_{\alpha} R_x^{\alpha}(\tau)e^{i2\pi\alpha t}$$

for which the cyclic autocorrelation function is defined as

$$R_x^{\alpha}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{\frac{T}{2}}^{\frac{T}{2}} R_x(t,\tau)e^{-i2\pi\alpha t}dT$$

Again the Fourier transform of $R_x^{\alpha}(\tau)$ is the SCD defined as

$$S_x^{\alpha}(\tau) = \int_{-\infty}^{\infty} R_x^{\alpha}(\tau)e^{-i2\pi f\tau}d\tau$$

This is a two-dimensional transform unlike power spectral density which is one-dimensional. For successful detection under cyclo-stationary based spectrum sensing, we need a priori knowledge of the cyclo-stationary features of the received signal only. However matched filtering is the optimal solution when we completely know about received signal in advance. This technique doesn't work well when the underlying noise is stationary. More over channel fading destroys the property of cyclo stationarty of the received signal and is also susceptible to sampling clock offset.

5.4 Comparision of various spectrum detection techniques

Figure 5.4 compares various spectrum sensing techniques on the basis of their accuracy and complexity. We can see that energy detection technique is the least accurate and least complex of all where as matched filtering is most complex and highly accurate. Other techniques lie in between with some having more accuracy and some less complex. There is

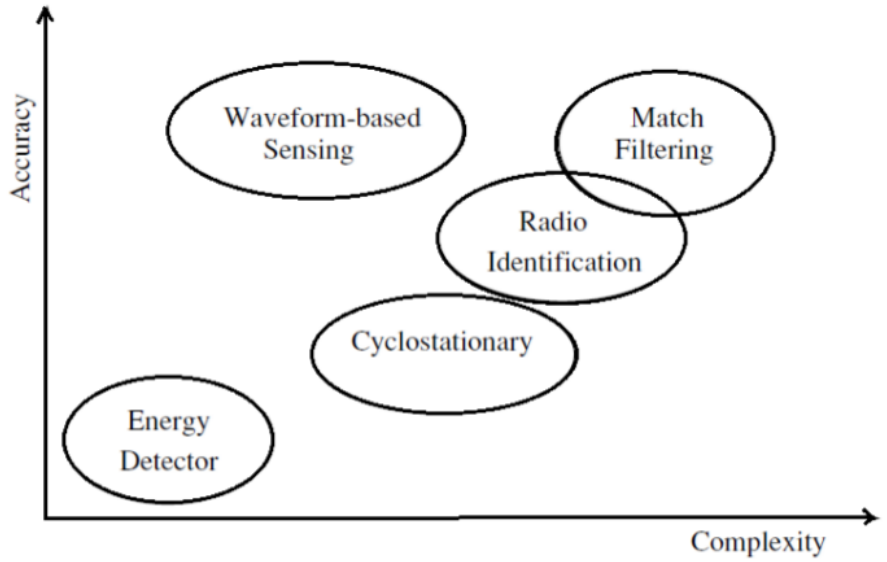

Figure 5.4: Comparison of sensing methods

no ideal detector that suits all occasions. Thus decisions, compromises and tradeoffs must be made depending on primary radio type, transmission and propagation characteristics, characteristics of secondary user receiver, and etc[9].

5.5 Implementaton of energy detection technique

We have used energy detection technique for spectrum sensing in our project for the detection of primary users in the band of interest. Average periodigram analysis is a method to implement energy detection technique of spectrum sensing. This section describes average periodogram analysis. Implementation of wide band spectrum analyzer using this technique is also described in this section.

5.5.1 Average periodogram analysis

Average Periodogram analysis estimates the power spectrum of the received signal and it is based on the Discrete Fourier Transform (DFT) of finite length segments of signal. In this technique signal is sectioned into finite length segments and periodogram of each segment is calculated which are also referred to as modified periodograms. Then an average of all these

modified periodogram is calculated[15].

Let $X[n]; n = 0, 1...L-1$ be the discrete time signal which is divided into M finite length segments of equal length, where N is the length of each segment i.e. $MN = L; X_r[n]; n = 0, 1...N-1$ is the rth segment and $W[n]; n = 0, 1...N-1$ is the window applied to each segment. The modified periodogram for the rth segment is,

$$I_r[k] = \frac{1}{NU} |V_r[k]|^2 \qquad k = 0, 1...N-1$$

where $V_r[k]$ is a N point DFT and U is normalization factor i.e. , $V_r[k] = DFT\{W[n] * X[n]\}$ and $U = \frac{1}{N}(\sum_{n-0}^{N-1}(W[n])^2)$. The PSD of $X[n]$ sequence is then the time averaged periodogram estimate ,

$$I[k] = \frac{1}{M} \left| \sum_{r=0}^{M-1} X_r[k] \right|$$

5.6 Wide band spectrum analyzer

GNU radio packages provide a tool for wide band spectrum sensing called usrp_spectrum_sense.py. It is used as a basic code for wide band spectrum analyzer implementation. The output of this code is the magnitude squared of the FFT. This means for each FFT bin[i] the output is $Y[i] = re[X[i]] * re[X[i]] + im[X[i]] * im[X[i]]$. We can calculate the power by taking square root of the output. We need N time samples of $x(t)$ sampled at a sampling frequency of F_s to use N point complex FFT $X(\omega)$ analysis. An appropriate window function is to be selected to reduce spectral leakage and applied to these time samples. The output of the complex FFT will represent the frequency spectrum content as follows: The first value of the FFT output ($bin0 == X[0]$) is the passband centre frequency The first half of FFT spectrum ($X[1]$ to $X[N/2-1]$) contains the positive baseband frequencies, which corresponds to the passband spectrum from centre frequency to $+F_s/2$. The second half of the FFT ($X[N/2]$ to $X[N-1]$) contains the negative baseband frequencies, i.e. from $-F_s/2$ to centre frequency.

For our project purpose, we collected 1024 samples using a tuner centered at uplink frequency of our interest, say 900 MHz. 1024 is choosen as the number of FFT points because the number of FFT points has to be a power of 2 for the fast execution of the FFT algorithm. Default sampling frequency is set as 1 MHz. The frequency resolution is therefore: 1 MHz / 1024 = 976.56 KHz. The decimation is defined as dsp rate divided by sample rate. The

UHD driver requires the decimation value to be an even number. The dsp rate is the actual hardware-level sampling rate of the USRP kit. It is the rate at which the USRP device takes analog samples from the external world and converts them to digital form. The dsp rate of the USRP is 100 MHz. Hence we chose sampling frequency to be 1 MHz which gives a decimation value of: 100 MHz / 1 MHz = 100.

A GSM band is of 200 KHz. So, in order to calculate the energy in a particular frequency channel of interest, we need to find the average of all the bin values which lie in the 200 KHz band centered at that frequency.

Chapter 6

Implementation of cognitive radio using OpenBTS

In our project we are able to successfully demonstrate the coexistence of primary users and secondary users in the GSM band. In order to accomplish this, implementation of a cognitive radio which detects the spectrum holes in the radio spectrum and enables secondary users to utilize these for communication is needed. An experimental setup is developed for this demonstration using OpenBTS and GNU radio software and USRP N210 as hardware. First a two frequency system is developed which is then expanded to a four frequency system.

6.1 Two frequency system

6.1.1 SYSTEM DESCRIPTION

The figure above describes the experimental setup for two frequency system. It consist of one primary and one secondary subsystems. The primary subsystem has only OpenBTS software and one USRP for RF front. Where as secondary subsystem has OpenBTS along with GNU radio and two USRP kits for each of these softwares as hardware RF front. This secondary subsystem has cognitive capabilities. To provide cognitive capabilities it was necessary for OpenBTS and GNU radio to run together in the same computer and communicate with each other which was challenging. Secondary subsystem continuously senses the frequency band of interest and takes decision depending upon the analysis of the data collected and changes its

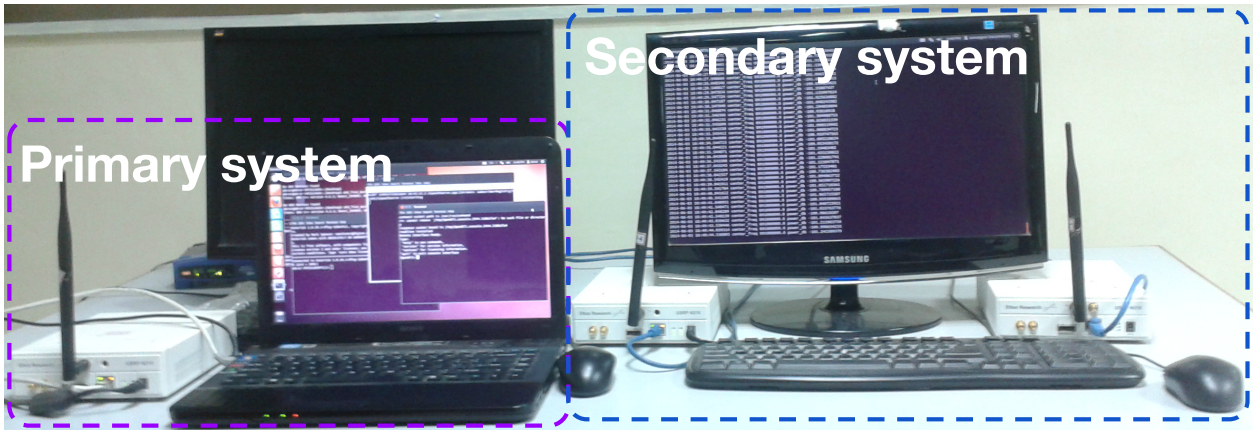

Figure 6.1: Experimental setup of the two frequency system

parameters accordingly so that primary and secondary users coexist. The spectrum sensing is accomplished by using GNU radio. GNU radio and OpenBTS of the secondary subsystem were made to coordinate and behave in appropriate manner and take dynamic decisions as and when required to make over all system behave cognitively.

6.1.2 TESTING

For two frequency cognitive system, two GSM bands are used with centre frequency 945MHz (F_1) and 950MHz (F_2). Secondary users are made to occupy one of these two bands say F_1. Then we make primary users enter the same band. This results in an increase in energy levels in this band which is sensed by the secondary subsystem as it is continuously scanning this band. Immediately secondary users are shifted to other frequency band (F_2) thereby vacating F_1 for primary users. Hence a two frequency cognitive system demonstrating coexistence of a pair of primary and secondary users is accomplished. The whole technique is described using a flow graph shown in figure 6.2:

6.2 Four frequency system

The two frequency system is expanded to a four frequency system by adding another primary subsystem to the two frequency experimental setup and having a four frequency radio spectrum instead of two.

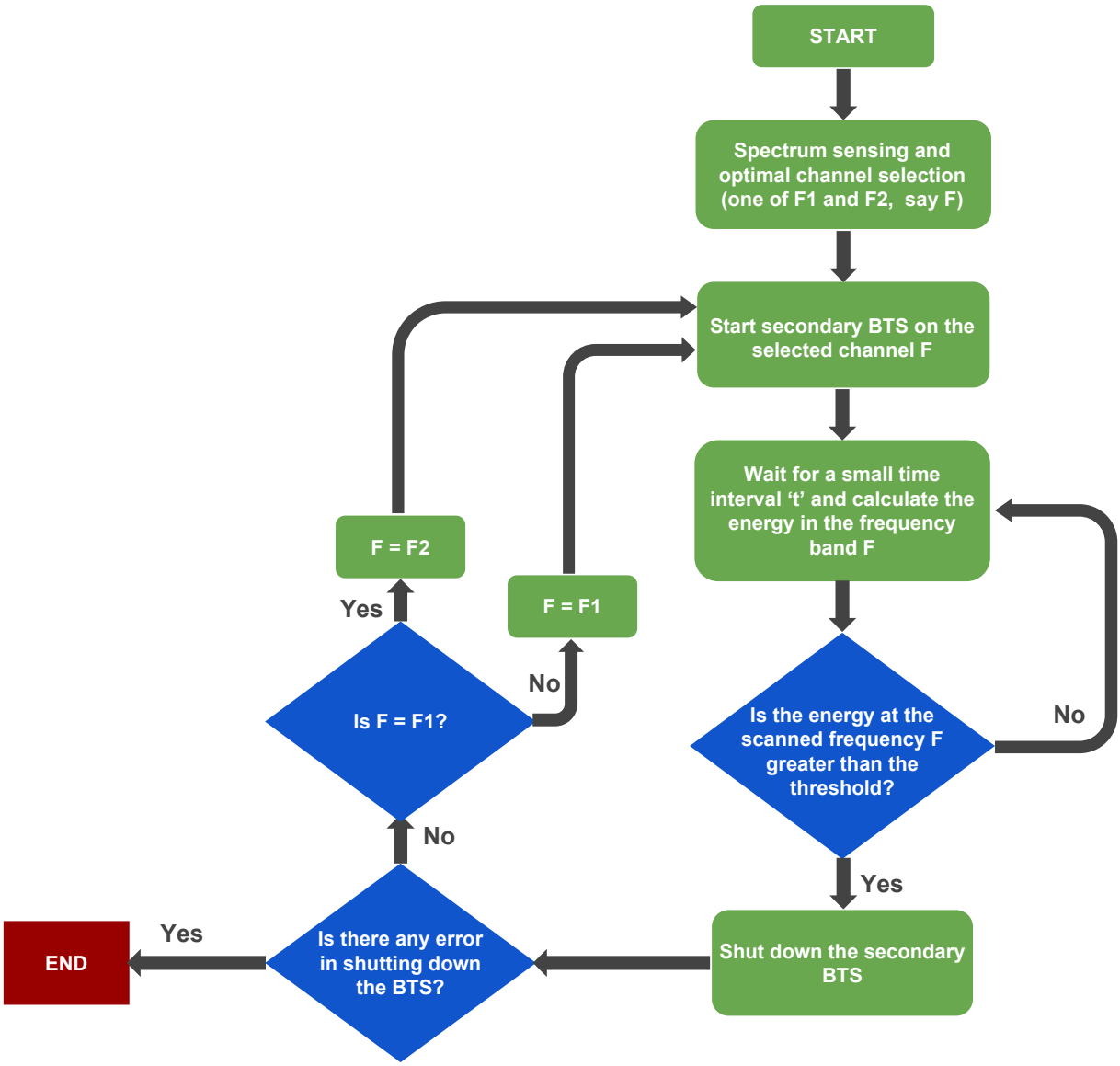

Figure 6.2: Flowchart for two frequency system

6.2.1 SYSTEM DESCRIPTION

The experimental setup of the four freqency system has has two primary subsystems and one secondary subsystem as described in the figure above. Primary subsystem has OpenBTS and one USRP kit and secondary subsystem has OpenBTS and GNU radio software and two USRP kits as we had previously in two frequency system. Here we have four GSM bands with centre frequencies F_1=936MHz, F_2=943MHz, F_3=950MHz, F_4=957MHz as the radio spectrum.

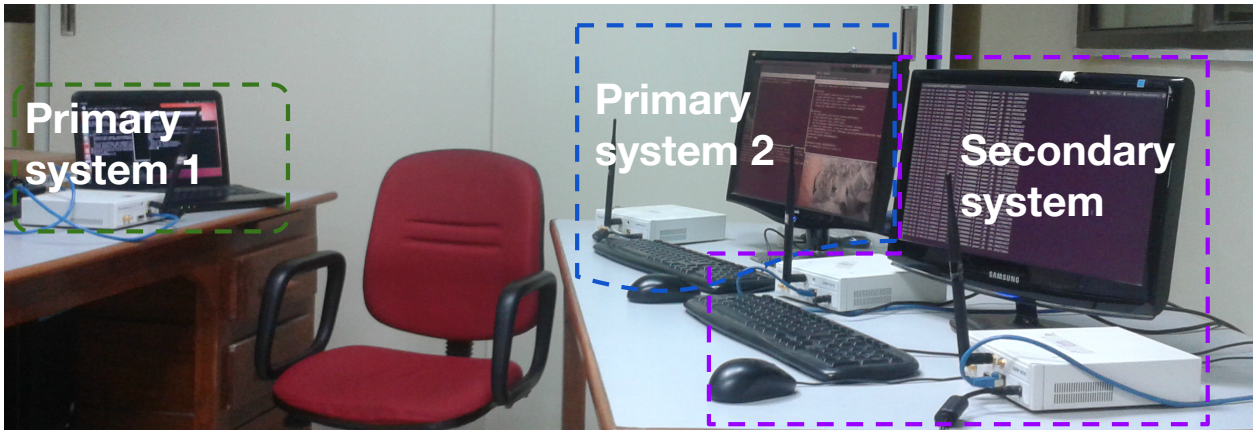

Figure 6.3: Experimental setup of the four frequency system

6.2.2 Testing

Here we make one pair of primary users occupy one of the four frequencies say F_2. We make secondary users use one frequency say F_1. Now other pair of primary users are made try and enter frequency F_1 for communication. This is sensed by the secondary system and it tries to migrate secondary users to F_2 which is also occupied. Our secondary system detects that F_2 is occupied and therefore continues to find a spectrum hole in a four frequency spectrum. It finds that frequency F_3 is unoccupied and thus allows secondary users to enter F_3 and utilize it for communication. The difference between a four frequency system and a two frequency system is that in a four frequency system the secondary subsystem has to first confirm absence of primary users in the band before making secondary users migrate to that band . This was not the case in two frequency system as it has only one pair of primary users. Hence the other band where secondary users will be made to migrate is always unoccupied at the time of switching secondary radio to that band. So there is no need to check for the peresence of primary users in that band.

The flow graph in figure 6.4 describes the four frequency cognitive system:

The spectrum sensing is done by energy detection technique and it was required that a proper threshold is set for decision making. A number of readings were taken to decide the noise level, energy level when only primary users are occupying and also energy levels when both primary users and secondary users exist in the same band for a short duration of time. The threshold value depends on the power transmitted by the users and their distance from the USRP kits which is RF front for GNU radio. This distance dependency can be removed

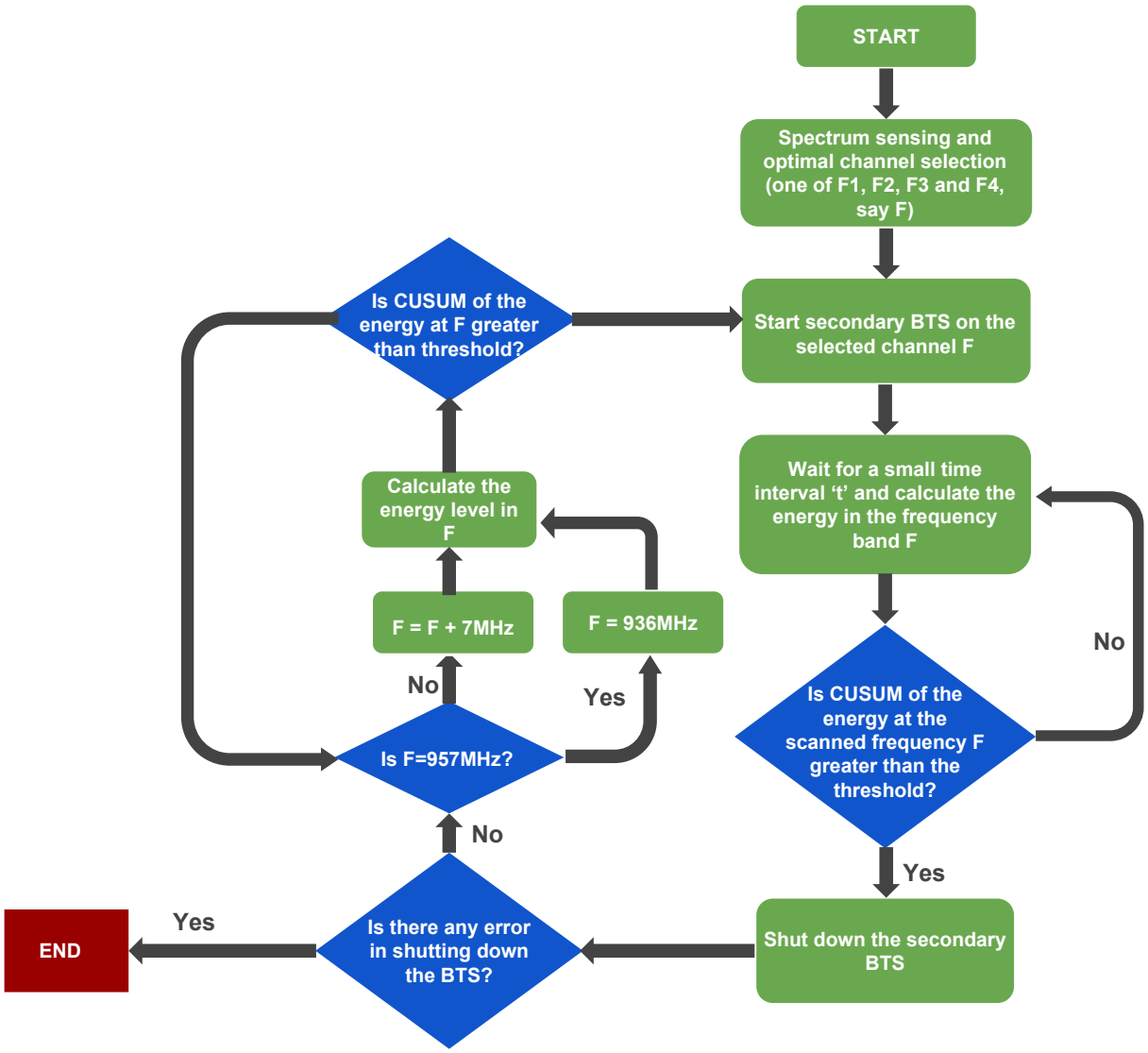

Figure 6.4: Flowchart for four frequency system

by setting the threshold much lower so that even if the users move far away, the decision making is not affected.

6.3 CUSUM

CUMSUM peak detection is also applied after energy detection to ensure that the detected high energy in the band of interest is not due to some irrelevant reasons like random fluctuations in noise power etc, but due to the presence of primary radio in that band. This ensures high accuracy in primary radio detection and correct decision making. CUSUM is basically a sequential analysis technique to detect change. It is a criterion for deciding when to take corrective action. As the name implies CUSUM involves calculating cumulative sum. This makes it sequential. The samples from a process x_n are assigned weights ω_n and summed as followed,

$$S_0 = 0;$$
$$S_{n+1} = max(0, S_n + x_n - \omega_n)$$

When the value of S exceeds a certain threshold value a change in value has been found. This formula detects change only in positive direction. To detect change in negative direction we have to do min operation instead of max operation and the change is detected when value goes below the threshold.

6.4 Tasks undertaken over the year

The next section of this chapter gives a step by step description of the tasks we accomplished during our project tenure.

1. We began by exploring what cognitive radio is and how it can be used in the already existing radio. We did a literature survey on the ongoing work in the field of cognitive radio.

2. We learned how to use the GNURadio software package starting from its installation procedure. We also designed an FM receiver using GNURadio Companion to get used to the software. Also we tried and learnt the codes of already existing signal processing blocks that GNU Radio provides.

3. GNURadio applications are primarily written in the Python programming language and hence we learned the language, Python.

4. USRP N210 kit is used as hardware in our project. We got used to this kit and also carried out range testing of this kit to ensure distance is not a major factor in our decision making algorithm and can be neglected.

5. The next task was to understand the working of OpenBTS software. Starting with the installation of this software we registered our GSM SIM cards in the local network established by OpenBTS. We could perform calling and sending SMS between our phones using the local network established by OpenBTS with USRP kit as its radio interface.

6. Since spectrum sensing is major part of cognitive radio, literature survey on various spectrum sensing techniques was done. We chose energy detection spectrum sensing technique for our project and so we did detailed study of a technique called Average Periodogram Analysis to implement this method. We also simulated this technique in Matlab using various windowing methods and understand results.

7. After all this the problem statement was designed and a flow graph of how this problem will be approached was constructed. We also decided upon the experimental setup required for this problem. Detailed discription of all this is included in the previous sections of this chapter.

8. First key step to approach the problem was to run Open BTS and GNU Radio together in the same computer with two USRP kits connected one for OpenBTS and the other for GNURadio. It was a little tricky because we had to find out whether it was possible to run two USRP kits on the same computer simultaneously. Fortunately, it turned out it was possible if the two kits had two different IP addresses. We then tried to figure out how to burn a different IP address to a USRP kit and managed to do the same.

9. The next step was to build a two frequency cognitive system with a pair of primary and secondary users communicating in parallel and primary pair having higher priority

when ever both try to use same radio band. Detailed description is in the previous sections of this chapter.

10. This two frequency system was expanded to four frequency system with two primary pair of users and one secondary pair of users coexisting. This demonstrated that both primary and secondary users can exist in the same GSM Network without affecting the existing system.

Chapter 7

Conclusion

Cognitive radio is the solution to current day problem of inefficient utilization of the radio spectrum. It identifies the spectrum holes and enables secondary users to utilize these holes for communication thereby increasing total number of mobile users. We have demonstrated these cognitive radio capabilities using a two frequency and a four frequency cognitive radio test bed and successfully testing them. We demonstrated that the secondary users quit the frequency channel which they are utilizing for communication as soon as the corresponding primary radio emerges thereby giving higher priority to the primary users.

Appendix A

Codes

A.1 secondaryBTS.py

```
#!/usr/bin/env python
#
# Copyright 2005,2007,2011 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, eng_notation
from gnuradio import blocks
from gnuradio import audio
from gnuradio import filter
from gnuradio import fft
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import sys
import math
import struct
import threading
import time
import sqlite3
import os
import subprocess
from datetime import datetime

sys.stderr.write("Warning: this may have issues on some machines+
    Python version combinations to seg fault due to the callback in
    bin statitics.\n\n")

class ThreadClass(threading.Thread):
    def run(self):
        return

class tune(gr.feval_dd):
```

```
    """
    This class allows C++ code to callback into python.
    """
    def __init__(self, tb):
        gr.feval_dd.__init__(self)
        self.tb = tb

    def eval(self, ignore):
        """
        This method is called from blocks.bin_statistics_f when it
            wants
        to change the center frequency.  This method tunes the
           front
        end to the new center frequency, and returns the new
           frequency
        as its result.
        """

        try:
            # We use this try block so that if something goes
               wrong
            # from here down, at least we'll have a prayer of
               knowing
            # what went wrong.  Without this, you get a very
            # mysterious:
            #
            #   terminate called after throwing an instance of
            #   'Swig::DirectorMethodException' Aborted
            #
            # message on stderr.  Not exactly helpful ;)

            new_freq = self.tb.set_next_freq()

            # wait until msgq is empty before continuing
```

```
            while(self.tb.msgq.full_p()):
                #print "msgq full, holding.."
                time.sleep(0.1)

            return new_freq

        except Exception, e:
            print "tune: Exception: ", e


class parse_msg(object):
    def __init__(self, msg):
        self.center_freq = msg.arg1()
        self.vlen = int(msg.arg2())
        assert(msg.length() == self.vlen * gr.sizeof_float)

        # FIXME consider using NumPy array
        t = msg.to_string()
        self.raw_data = t
        self.data = struct.unpack('%df' % (self.vlen,), t)


class my_top_block(gr.top_block):

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

        usage = "usage: %prog [options] down_freq"
        parser = OptionParser(option_class=eng_option, usage=usage
            )
        parser.add_option("-a", "--args", type="string", default="
            addr=192.168.20.2",
                          help="UHD device device address args [
                              default=%default]")
```

```
parser.add_option("", "--spec", type="string", default=
    None,
                  help="Subdevice of UHD device where
                      appropriate")
parser.add_option("-A", "--antenna", type="string",
    default=None,
                      help="select Rx Antenna where
                          appropriate")
parser.add_option("-s", "--samp-rate", type="eng_float",
    default=1e6,
                      help="set sample rate [default=%default]
                          ")
parser.add_option("-g", "--gain", type="eng_float",
    default=None,
                      help="set gain in dB (default is
                          midpoint)")
parser.add_option("", "--tune-delay", type="eng_float",
                      default=0.25, metavar="SECS",
                      help="time to delay (in seconds) after
                          changing frequency [default=%default]
                          ")
parser.add_option("", "--dwell-delay", type="eng_float",
                      default=0.25, metavar="SECS",
                      help="time to dwell (in seconds) at a
                          given frequency [default=%default]")
parser.add_option("-b", "--channel-bandwidth", type="
    eng_float",
                      default=976.56, metavar="Hz",
                      help="channel bandwidth of fft bins in
                          Hz [default=%default]")
parser.add_option("-l", "--lo-offset", type="eng_float",
                      default=0, metavar="Hz",
                      help="lo_offset in Hz [default=%default]
                          ")
```

```
parser.add_option("-q", "--squelch-threshold", type="
    eng_float",
                      default=None, metavar="dB",
                      help="squelch threshold in dB [default=%
                          default]")
parser.add_option("-F", "--fft-size", type="int", default=
    None,
                      help="specify number of FFT bins [
                          default=samp_rate/channel_bw]")
parser.add_option("", "--real-time", action="store_true",
    default=False,
                      help="Attempt to enable real-time
                          scheduling")


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

self.channel_bandwidth = options.channel_bandwidth

self.down_freq = eng_notation.str_to_num(args[0])
self.up_freq = (self.down_freq) - 45e6



if not options.real_time:
    realtime = False
else:
    # Attempt to enable realtime scheduling
    r = gr.enable_realtime_scheduling()
    if r == gr.RT_OK:
        realtime = True
```

```
    else:
        realtime = False
        print "Note: failed to enable realtime scheduling"

# build graph
self.u = uhd.usrp_source(device_addr=options.args,
                         stream_args=uhd.stream_args('fc32
                             '))

# Set the subdevice spec
if(options.spec):
    self.u.set_subdev_spec(options.spec, 0)

# Set the antenna
if(options.antenna):
    self.u.set_antenna(options.antenna, 0)

self.u.set_samp_rate(options.samp_rate)

self.usrp_rate = usrp_rate = self.u.get_samp_rate()

self.lo_offset = options.lo_offset

if options.fft_size is None:
    self.fft_size = int(self.usrp_rate/self.
        channel_bandwidth)
else:
    self.fft_size = options.fft_size

self.squelch_threshold = options.squelch_threshold

s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.
    fft_size)
```

```
        mywindow = filter.window.blackmanharris(self.fft_size)
        ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
        power = 0
        for tap in mywindow:
            power += tap*tap

        c2mag = blocks.complex_to_mag_squared(self.fft_size)


        tune_delay  = max(0, int(round(options.tune_delay *
           usrp_rate / self.fft_size)))  # in fft_frames
        dwell_delay = max(1, int(round(options.dwell_delay *
           usrp_rate / self.fft_size))) # in fft_frames

        self.msgq = gr.msg_queue(1)
        self._tune_callback = tune(self)        # hang on to this
           to keep it from being GC'd
        stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
                                        self._tune_callback,
                                           tune_delay,
                                        dwell_delay)

        # FIXME leave out the log10 until we speed it up
    #self.connect(self.u, s2v, ffter, c2mag, log, stats)
    self.connect(self.u, s2v, ffter, c2mag, stats)

        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.u.get_gain_range()
            options.gain = float(g.start()+g.stop())/2.0

        self.set_gain(options.gain)
        print "gain =", options.gain
```

```
    def set_next_freq(self):
        target_freq = self.up_freq


        if not self.set_freq(target_freq):
            print "Failed to set frequency to", target_freq
            sys.exit(1)

        return target_freq


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

        Args:
            target_freq: frequency in Hz
        @rypte: bool
        """

        r = self.u.set_center_freq(uhd.tune_request(target_freq,
           rf_freq=(target_freq + self.lo_offset),rf_freq_policy=
           uhd.tune_request.POLICY_MANUAL))
        if r:
            return True

        return False

    def set_gain(self, gain):
        self.u.set_gain(gain)



def main_loop(tb):
```

```
    startOpenBTS(tb.down_freq,tb)



def sub_loop(tb):

    # use a counter to make sure power is less than threshold
    # lowPowerCount = 0
    # lowPowerCountMax = 10
    print 'fft_size', tb.fft_size
    N = tb.fft_size
    mid = N // 2
    cusum = 0
    counter = 0


    while 1:

        # Get the next message sent from the C++ code (blocking
            call).
        # It contains the center frequency and the mag squared of
            the fft
        m = parse_msg(tb.msgq.delete_head())

        # m.center_freq is the center frequency at the time of
            capture
        # m.data are the mag_squared of the fft output
        # m.raw_data is a string that contains the binary floats.
        # You could write this as binary to a file.



        center_freq = m.center_freq
```

```
bins = 102
power_data = 0
noise_floor_db = 0        ##  10*math.log10(min(m.data)/tb.
    usrp_rate)

for i in range(1, bins+1):
    power_data += m.data[mid-i] + m.data[mid+i]
power_data += m.data[mid]
power_data /= ((2*bins) + 1)

power_db = 10*math.log10(power_data/tb.usrp_rate) -
    noise_floor_db
power_threshold = -70.0


#if (power_db > tb.squelch_threshold) and (power_db >
    power_threshold):
    #print datetime.now(), "center_freq", center_freq, "
        power_db", power_db, "in use"
    # lowPowerCount = 0
#else:
print datetime.now(), "center_freq", center_freq, "
    power_db", power_db
    # lowPowerCount += 1

#     if (lowPowerCount > lowPowerCountMax):
#         down_freq = center_freq + 45e6
#         startOpenBTS(down_freq)
#         break

#cusum cusum cusum is here
cusum = max(0, cusum + power_db - power_threshold)
if (cusum > 0):
    counter += 1
```

```
            if (counter > 2):
                print "CUSUM is now positive!!!"
                down_freq = center_freq + 45e6
                quitOpenBTS(down_freq, tb)
                break
        else:
            counter = 0



def startOpenBTS(downFrequency,tb):
    arfcn=int((downFrequency-935e6)/2e5)
    if (arfcn < 0):
        print "ARFCN must be > 0 !!!"
        sys.exit(1)
    print 'ARFCN=', arfcn
    #DB modifications
    t=(arfcn,)
    conn=sqlite3.connect("/etc/OpenBTS/OpenBTS.db")
    cursor=conn.cursor()
    cursor.execute("update config set valuestring=? where
        keystring='GSM.Radio.C0'",t)
    conn.commit()

    #start the OpenBTS
    f=subprocess.Popen(os.path.expanduser('~/ddpOpenBTS/runOpenBTS
        .sh'))
    f.wait()
    tb.msgq.delete_head()
    time.sleep(0.25)
    sub_loop(tb)
```

```
def quitOpenBTS(downFreq, tb):
    f=subprocess.Popen(os.path.expanduser('~/ddpOpenBTS/
        quitOpenBTS.sh'))
    f.wait()
    newDownFreq = getNewChannel(downFreq, tb)
    startOpenBTS(newDownFreq, tb)



def getNewChannel(downFreq, tb):
    newDownFreq = downFreq + 7e6
    if newDownFreq > 960e6:
        newDownFreq = 936e6

    tb.up_freq = newDownFreq - 45e6
    print "new tb.up_freq: ", tb.up_freq
    tb.msgq.delete_head()
    time.sleep(0.25)

    print 'fft size', tb.fft_size
    N = tb.fft_size
    mid = N // 2
    cusum = 0
    counter = 0
    loopcounter = 0


    while loopcounter < 10:

        # Get the next message sent from the C++ code (blocking
            call).
        # It contains the center frequency and the mag squared of
            the fft
        m = parse_msg(tb.msgq.delete_head())
```

```
        center_freq = m.center_freq
        bins = 102
        power_data = 0


        for i in range(1, bins+1):
            power_data += m.data[mid-i] + m.data[mid+i]
        power_data += m.data[mid]
        power_data /= ((2*bins) + 1)

        power_db = 10*math.log10(power_data/tb.usrp_rate)
        power_threshold = -70.0


        print datetime.now(), "center_freq", center_freq, "
            power_db", power_db
        print "precheck"

        #cusum cusum cusum is here
        cusum = max(0, cusum + power_db - power_threshold)
        loopcounter += 1
        if (cusum > 0):
            counter += 1
            if (counter > 2):
                print "CUSUM is now positive!!!"
                newDownFreq = getNewChannel(newDownFreq, tb)
                break
        else:
            counter = 0
    return newDownFreq
```

```
if __name__ == '__main__':
    t = ThreadClass()
    t.start()

    tb = my_top_block()
    try:
        tb.start()
        main_loop(tb)

    except KeyboardInterrupt:
        pass
```

A.2 primaryBTS.py

```
#!/usr/bin/env python


import sys
import sqlite3
import os
import re

def main_loop():
    usage = "usage: %prog channel_freq"
    if len(sys.argv) != 2:
        print 'usage:', sys.argv[0], 'channel_freq'
        sys.exit(1)



    center_freq = int(re.match(r'\d+', sys.argv[1]).group())*1e6
    startOpenBTS(center_freq)
```

```
def startOpenBTS(frequency):

    arfcn=int((frequency-935e6)/2e5)
    print 'ARFCN=', arfcn

    #DB modifications
    t=(arfcn,)
    conn=sqlite3.connect("/etc/OpenBTS/OpenBTS.db")
    cursor=conn.cursor()
    cursor.execute("update config set valuestring=? where
        keystring='GSM.Radio.C0'",t)
    conn.commit()

    #start the OpenBTS
    f=os.popen('~/ddpOpenBTS/runOpenBTS.sh')
    f.close()


if __name__ == '__main__':

    try:
        main_loop()

    except KeyboardInterrupt:
        pass
```

A.3 runOpenBTS.sh

```
#!/bin/bash

sudo echo "Hi, this script starts OpenBTS in Ubuntu 12.04!"
sudo service asterisk restart
sudo gnome-terminal -x sh -c "sudo asterisk -r" &
```

```
#cd ~/OpenBTS/
#sudo gnome-terminal --tab -e "sudo smqueue/trunk/smqueue/smqueue"
    \
#   --tab -e "sudo subscriberRegistry/trunk/sipauthserve" &

cd ~/OpenBTS/openbts/trunk/apps/
sudo gnome-terminal --tab -e "sudo ../../../smqueue/trunk/smqueue/
   smqueue" \
    --tab -e "sudo ../../../subscriberRegistry/trunk/sipauthserve"
        &

#sudo gnome-terminal -x sh -c "sudo ./OpenBTS" &
#sudo gnome-terminal -x sh -c "sudo ./OpenBTSCLI" &
sudo gnome-terminal --tab -e "sudo ./OpenBTS" \
    --tab -e "sudo ./OpenBTSCLI" &
cd ~
```

A.4 quitOpenBTS.sh

```
#!/bin/bash

sudo echo "Hi, this script turns OpenBTS off in Ubuntu 12.04!"

sudo killall transceiver smqueue sipauthserve OpenBTSCLI asterisk
```

Appendix B

Installation procedures

To install either GNURadio or OpenBTS, you first need to have the UHD (driver software for the USRP) installed. The installation procedures given in this chapter are for the Ubuntu 12.04 LTS desktop operating system only. For other operating systems, please check on the internet.

B.1 UHD

```
# Install the runtime dependencies:

sudo apt-get install python libboost-all-dev libusb-1.0-0-dev
   python-cheetah \
doxygen python-docutils git cmake

# Go to your home folder and git clone the UHD repository:

cd ~
git clone https://github.com/EttusResearch/uhd.git

# Generate Makefiles with CMake:

cd uhd/host
```

```
mkdir build
cd build
cmake ../

# Build and install:

make
make test
sudo make install
sudo ldconfig
```

B.2 OpenBTS

```
cd ~
sudo apt-get install subversion
mkdir OpenBTS
svn co http://wush.net/svn/range/software/public OpenBTS/

sudo apt-get install autoconf libtool libosip2-dev libortp-dev \
libusb-1.0-0-dev g++ sqlite3 libsqlite3-dev erlang libreadline6-
    dev \
libncurses5-dev asterisk

cd OpenBTS
cd a53/trunk
sudo make install


## for USRP N210 only, for other devices please check the internet
    <
cd openbts/trunk
autoreconf -i
./configure --with-uhd
make
```

```
#(from OpenBTS root)
cd apps
ln -s ../Transceiver52M/transceiver .

## for USRP N210 only, for other devices please check the internet
    >

sudo mkdir /etc/OpenBTS
sudo sqlite3 -init ./apps/OpenBTS.example.sql /etc/OpenBTS/OpenBTS
    .db ".quit"
sudo sqlite3 /etc/OpenBTS/OpenBTS.db .dump

sudo mkdir -p /var/lib/asterisk/sqlite3dir

cd ../..
cd subscriberRegistry/trunk
make

sudo sqlite3 -init subscriberRegistry.example.sql \
/etc/OpenBTS/sipauthserve.db ".quit"



cd ../..
cd smqueue/trunk
autoreconf -i
./configure
make

sudo sqlite3 -init smqueue/smqueue.example.sql /etc/OpenBTS/
    smqueue.db ".quit"
```

B.3 GNURadio

```
## for Ubuntu 12.04 only, for other versions of Ubuntu check the
   internet <

sudo apt-get install libfontconfig1-dev libxrender-dev \
libpulse-dev swig g++ automake autoconf libtool python-dev \
libfftw3-dev libcppunit-dev libboost1.48-all-dev libusb-dev \
libusb-1.0-0-dev fort77 libsdl1.2-dev python-wxgtk2.8 git-core \
libqt4-dev python-numpy ccache python-opengl libgsl0-dev python-
   cheetah \
python-lxml doxygen qt4-dev-tools libusb-1.0-0-dev libqwt5-qt4-dev
    \
libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4 cmake git-
   core wget \
libxi-dev python-docutils gtk2-engines-pixbuf r-base-dev python-tk
    \
liborc-0.4-0 liborc-0.4-dev libasound2-dev python-gtk2

## for Ubuntu 12.04 only, for other versions of Ubuntu check the
   internet >
cd ~
git clone http://git.gnuradio.org/git/gnuradio.git
cd gnuradio
mkdir build
cd build
cmake ../
make && make test
sudo make install
sudo ldconfig
```

Bibliography

[1] Kranthi Ananthula. Experimental setup of cognitive radio test-bed using software defined radio. Master's thesis, 2013.

[2] Federal Communications Commission. Spectrum policy task force. *ET Docket No. 02-135*, November 2002.

[3] S. M. Mishra D. Cabric and R. W. Brodersen. Implementation issues in spectrum sensing for cognitive radios. *Proc. IEEE Asilomar Conf. Signals, Sys., and Comp.*, November 2004.

[4] Joseph Mitola et al. Cognitive radio: Making software radios more personal. *IEEE Personal Communications*, 6(4):13–18, August 1999.

[5] M.A. Sarijari et. al. Energy detection sensing based on gnu radio and usrp: An analysis study. *Communications (MICC), 2009 IEEE 9th Malaysia International Conference on*, December 2009.

[6] Paul Kolodzy et al. Next generation communications: Kickoff meeting. In *Proc. DARPA*, October 2001.

[7] `http://gnuradio.org/redmine/projects/gnuradio/wiki`.

[8] Simon Haykin. Cognitive radio: Brain-empowered wireless communications. *IEEE Journal on selected areas in communications*, 23(2), 2005.

[9] Mehmet C. Vuran Ian F. Akyildiz, Won-Yeol Lee and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks*, 50, 2006.

[10] Andreas Miller. *DAB Software Receiver Implementation*. PhD thesis, ETH, 2008.

[11] Joseph Mitola. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.

[12] V. Prithiviraj, B. Sarankumar, A. Kalaiyarasan, P.P. Chandru, and N.N. Singh. Cyclostationary-based architectures for spectrum sensing in ieee 802.22 wran. *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, 2011.

[13] Gregory Staple and Kevin Werbach. The end of spectrum scarcity. *IEEE Spectrum*, 41(3):48–52, March 2004.

[14] Mansi Subhedar and Gajanan Birajdar. Spectrum sensing techniques in cognitive radio networks: A survey. *International Journal of Next-Generation Networks*, 3(2), June 2011.

[15] Peter D. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *Audio and Electroacoustics, IEEE Transactions on*, 15(2):70–73, June 1967.