OpenBTS based Cognitive Radio Test-Bed

Dual Degree Dissertation

Submitted in partial fulfilment of the requirements for the degrees of
Bachelor of Technology
(Electrical Engineering)
and
Master of Technology
(Communication and Signal Processing)

by

Swrangsar Basumatary
09d07040

Supervisor

Prof. S N Merchant

Department of Electrical Engineering
IIT Bombay

June 2014

Declaration of Academic Ethics

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Swrangsar Basumatary
(Roll 09d07040)

Date: . . . . . .

Abstract

The explosive proliferation of mobile phones and other wireless communication devices has rendered the spectrum an insufficient resource. To make things worse, there exists a widespread underutilization of the spectrum. Cognitive Radio (CR) offers to tackle this problem by finding the unused frequency bands in the spectrum (also known as spectrum holes) and allocating them to the secondary users (SUs) for use. Hence, CR makes the usage of spectrum more efficient.

We demonstrate the capabilities of CR by developing a 2-frequency CR Test-Bed and a 4-frequency CR Test-Bed. Primary Users (PUs) and SUs are made to coexist in the same frequency band. Energy detection method is used for the spectrum sensing. SUs switch from their frequency band of operation to an unused frequency band as soon as the activity of PUs is detected.

Acknowledgement

I would like to express my sincere gratitude to **Prof S N Merchant** for giving me the opportunity to work in this project and also for his invaluable guidance and encouragement during the course of the project. I am grateful to Prof V M Gadre for his valuable feedback and suggestions. I am also grateful to Prof V Rajbabu for making sure that we take our project work seriously.

I would like to specially mention my project partner Abrar Ahmad for all the help in practically carrying out this work. I appreciate the support rendered to me by SPANN lab and its members like Chaitanya Pande, Sudipto Mondal, Kartik Sudarshan, Sagar Sharma, Joy Khan and so on. Finally, I would like to thank my friends for their constant support and encouragement.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The electromagnetic radio spectrum is a natural resource that remains underutilized [1]. It is licensed by governments for use by transmitters and receivers. With the explosive proliferation of cell phones and other wireless communication devices, we cannot afford to be lavish in using our spectral resources anymore.

In November 2002, the Spectrum Policy Task Force, a group under the Federal Communications Commission(FCC) in the United States, published a report 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 were to scan the radio spectrum even in metropolitan places where it's heavily used, we would find that [3]:

1. some frequency bands are unoccupied most of the time,
2. some are only partially occupied and
3. the rest are heavily used.

The underutilization of spectral resources leads us to think in terms of *spectrum holes*, which are defined as [4]:

> *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.*

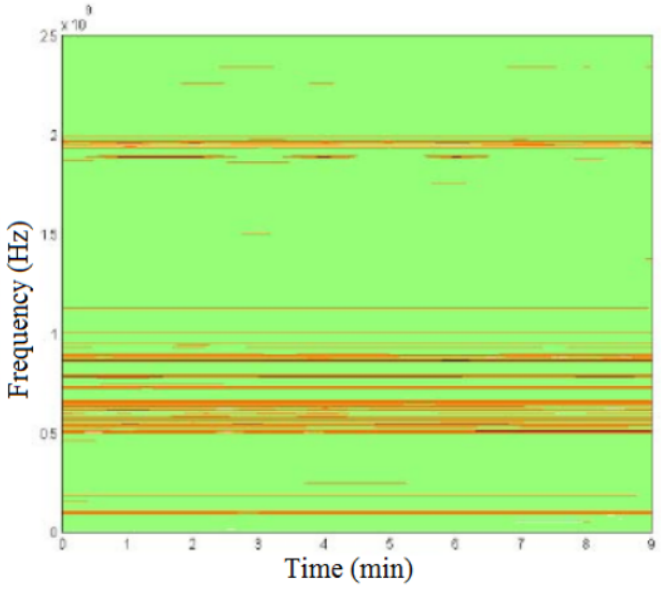

Figure 1.1: Frequency usage of the Spectrum

The spectrum can be better utilized by enabling secondary users (users who are not licensed to use the services) to access spectrum holes unoccupied by primary users at the location and the time in question. *Cognitive Radio*, which includes software-defined radio, has been promoted as the means to make efficient use of the spectrum by exploiting the existence of spectrum holes [1][5][6].

1.2 Cognitive Radio

One of the definitions of Cognitive Radio is [1]:

> *Cognitive radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and adapt its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operating parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind:*
>
> - *highly reliable communications whenever and wherever needed;*
> - *efficient utilization of the radio spectrum.*

Besides, a cognitive radio is also reconfigurable. This property of cognitive radio is provided by a platform known as *software-defined radio*. Software-defined Radio (SDR) is basically a combination of two key technologies: digital radio, and computer software.

1.3 Contribution of this Thesis

A cognitive base transceiver station is developed to demonstrate the efficient utilization of spectrum by allowing secondary users to make use of the frequency bands that are already licensed to primary users but that are not being used at that particular time and space.

1. A 2-frequency system is developed where as soon as the presence of primary users is detected in F_1 the secondary BTS switches from F_1 to F_2 and vice-versa.
2. The 2-frequency system is extended to a 4-frequency one where two of the four frequency channels always remain occupied. The secondary system switches to one of the two unused frequency channels.
3. For sensing the frequency channels the energy detection based spectrum sensing method has been used and for peak detection the method called CUSUM has been used. The frequency used by the secondary users is sensed continuously and as soon as the presence of primary users in that frequency is detected the secondary finds an underutilized frequency nearby and switches to that frequency.

1.4 Organization of this Thesis

The remaining chapters of this document are organized as follows. Chapter 2 briefly touches upon the GSM system architecture and the GSM protocol architecture. Chapter 3 introduces Software Defined Radio (SDR) and some of the tools that make SDR possible like the Universal Software Radio Peripheral (USRP N210) and the GNU Radio software package. Various parts of the OpenBTS software are described in Chapter 4. Chapter 5 reviews various techniques of spectrum sensing and gives a comparison among them. Chapter 6 describes the Cognitive Radio (CR) Test-Bed that we developed using OpenBTS. Flowgraphs of the algorithms developed while making the CR Test-Bed are also given in Chapter 6. Finally, Chapter 7 concludes this document.

Chapter 2

GSM

2.1 Overview

GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile), is a very popular standard that describes protocols for second generation (2G) digital cellular networks used by mobile phones. GSM networks usually operate in the 900 MHz, 1800 MHz or 1900 MHz bands. It supports a full data rate of 9.6 kbits/sec or 14.4 kbits/sec using better codecs.

2.2 System Architecture

A GSM Public Land Mobile Network (PLMN) consists of at least one Service Area managed by a Mobile Switching Center (MSC) connected to the Public Switched Telephone Network (PSTN) [7].

The network structure can be divided into the following discrete sections:

- Base Station Subsystem
- Network and Switching Subsystem
- Operation Subsystem

2.2.1 Base Station Subsystem (BSS)

A base station subsystem consists of

- a Base Station Controller (BSC) and

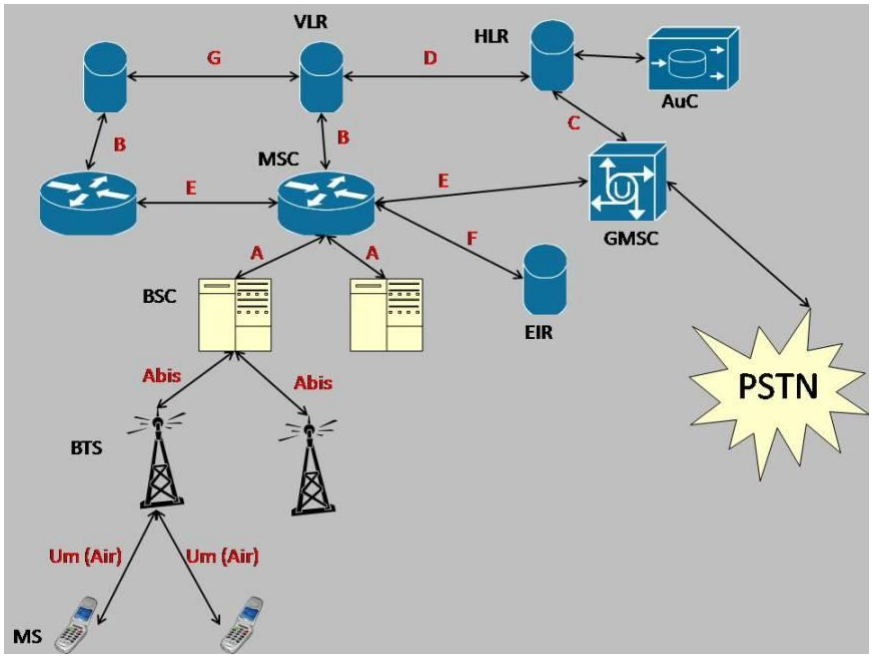

Figure 2.1: The architecture of a GSM Public Land Mobile Network (PLMN) [8]

- at least one Base Transceiver Station (BTS) for Mobile Stations (MS). A mobile station can be a cell phone, or any electronic equipment such as a Personal Digital Assistant (PDA) with a phone interface.

The area served by a BTS is called a Network Cell. One or more BTSs are managed by a BSC. A group of BSSs can be managed as a Location Area (Location Area) provided all those BSSs are being managed by the same MSC.

An MSC may also be connected via a Gateway MSC (GMSC) to other MSCs or the PSTN with the Integrated Services Digital Network (ISDN) option. The Inter-Working Function (IWF) of a GMSC makes it possible to connect the circuit switched data paths of a GSM network with the PSTN/ISDN.

2.2.2 Network and Switching Subsystem (NSS)

The NSS is made up of an MSC and a Visitor Location Register (VLR). An MSC

- sets up, controls and shuts down connections
- handles call charges
- manages additionals services like call forwarding, call blocking, etc.

A VLR contains all the subscriber data and location data of the phones being served by the accompanying MSC. The VLR also maintains data about the SIMs that do not belong to the network but have roamed into the network. The area served by an MSC is called

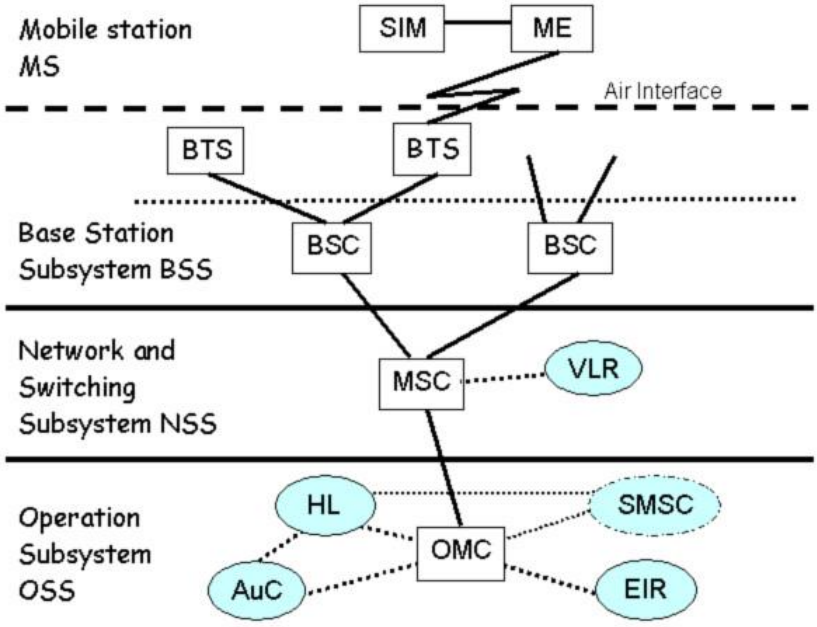

Figure 2.2: The GSM network architecture for a single MSC controlled Service Area [7].

a MSC/VLR service area.

2.2.3 The Operation Subsystem (OSS)

The OSS consists of the Operation and Maintenance Center (OMC), the Authentication Center (AuC), the Home Location Register (HLR) and the Equipment Identity Register (EIR).

The OSS is responsible for

- network management functions like service provisioning, network configuration, fault management, etc.
- billing calls
- administering subscribers

The AuC controls all the encryption algorithms used for verifying the SIMs. The EIR contains the serial numbers of all the MSs (mobile phones) being served. The HLR contains the subscriber data and location data of all the SIMs in different parts of the network.

2.3 Protocol Architecture

The data communication protocols in a GSM network are implemented to work over the bearer[1] data channel. The GSM protocol architecture is structured into three independent planes [9]:

- user plane
- control plane
- management plane

The user plane defines protocols for handling the voice and user data. At the Um interface, the traffic control channel (TCH) is used to carry the user data.

The control plane defines protocols for controlling connections by using signalling data. The signalling data are carried over logical channels called Dm-channels (wireless analog of the D-channels for wired interface). The spare capacities of the Dm-channels are used for carrying user data. Eventually all logical channels have to multiplexed onto the physical channel [9].

The management plane takes care of the coordination between different planes. It also manages functions related to the control and/or user planes. The management plane handles things like network configuration, network fault, etc.

2.3.1 Signalling Transmission

In GSM, the network nodes exchange signaling information with each other to establish, control and terminate connections. The various interfaces in a GSM network are:

- MS-BTS: Um
- BTS-BSC: Abis
- BSC-MSC: A
- MSC-VLR: B
- MSC-HLR: C
- VLR-HLR: D
- MSC-MSC: E
- MSC-EIR: F
- VLR-VLR: G

The Um interface is the only interface that uses the wireless physical medium for carrying signals. The rest of the interfaces all use wired and digital mediums.

[1] A bearer data channel is a channel that carries call content i.e. one that does not carry signaling.

DATA LINK LAYER (LAYER 2) PROTOCOLS

Link Access Protocol on Dm-channel (LAPDm) is a layer 2 protocol that provides safe, reliable connections to layer 3 protocols. It is a wireless-adapted version of the standard Link Access Protocol on D-channel (LAPD) of ISDN. It works in two modes: Unacknowledged and Acknowledged. In Unacknowledged mode it operates without acknowledgement, without error correction and without flow control. While in acknowledged mode, it asserts acknowledgement, error correction is done by resending and flow is controlled.

Message Transfer Part (MTP) is the standard ISDN message transport part of Signaling System 7 (SS7). The networking layers covered by MTP cannot be mapped one-to-one to the OSI model[2]. But it covers layer 1, layer 2 and parts of layer 3 from the OSI model. The parts of layer 3 not covered by MTP are covered by Signalling Connection Control Part (SCCP).

NETWORK LAYER (LAYER 3) PROTOCOLS

Radio Resource Management (RR) is a protocol that sets up, manages and terminates radio link channels. It is involved in measuring radio field strength, signal quality etc. It manages handover, modulation scheme, co-channel interference, etc. The goal is to utilize the limited spectral resources efficiently.

Mobility Management (MM) manages mobility of the mobile stations (MS). This protocol is used by the MS to communicate directly with the MSC bypassing the BSS. It works over an already established RR connection. It handles stuff like TMSI reallocation, authentication, IMSI attach/detach, roaming, location update procedure, etc.

Call Management (CM) protocol consists of the following parts:

- *Call Control (CC)* sets up, manages and ends calls. For each call a CC instance is created in the MS and another one in the MSC. CC instances communicate over already established MM and RR connections.
- *Short Message Service (SMS)* works over already established MM, RR and LAPDm connections.
- *Supplementary Services (SS)* provide upper layers the access to GSM supplementary services like call forwarding, call barring, etc.

Signal Connection Control Part (SCCP) is a SS7 protocol that provides routing, flow control, connection-orientation, error correction facilities etc. It works at the A-interface.

[2]Operation Systems Interconnection model

Base Station System Application Part (BSSAP) is a signaling protocol at the A interface supported by MTP and SCCP [9].

- *Direct Transfer Application Part (DTAP)* handles signaling between the MS and the MSC.
- *Base Station System Management Application Part (BSSMAP)* transfers management information from the BSC to the MSC.
- *Base Station System Operation and Management Application Part (BSSOMAP)* transports network management information from OMC to BSC.

Mobile Application Part (MAP) is an SS7 application-layer protocol for the various nodes in a GSM network [9]. It provides facilities such as:

- roaming support via location update, IMSI attach/detach, authentication
- call handling
- subscriber tracing
- SMS
- supplementary services

Chapter 3

Software Defined Radio

Software Defined Radio, SDR is a radio communication system where most of the hardware components have been replaced by software [10]. This isn't a new concept but recent advances in electronics technologies have made many things possible that were just theoretically possible before. In traditional radio systems, all the components are hardwired into the device. These components cannot be modified or tweaked easily. Most of them needs to be replaced instead. They are also much more expensive to reconfigure. But in an SDR, to change the functionality of the radio system all you need to do is rewrite its software. Thus it can be reconfigured easily and economically. The protocols used by the software based radio system can also be changed easily. Usually, in an SDR most of the hard work is handed over to a general purpose microprocessor instead of a special purpose hardware.

The traditional hardware radio system consists of elements such as mixers, filters, amplifiers, converters, modulators, etc. resulting in higher production costs and minimal flexibility. Whereas in SDR technologies like Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP) and General-Purpose Processor (GPP) are used to build the software radio elements.

The SDR contains a number of basic functional blocks. The SDR in general can be divided into three basic sections, namely the front end, the IF section and the base-band section. The front end section consits of analogue RF circuitry that is responsible for the reception and transmission of signals at the operating frequency. The IF section performs the digital to analog conversion and vice versa. It also does various signal processing tasks 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 gets processed [11][12]. We have used GNURadio and USRP N210 to configure the SDR used in implementing our Cognitive Radio Test-Bed.

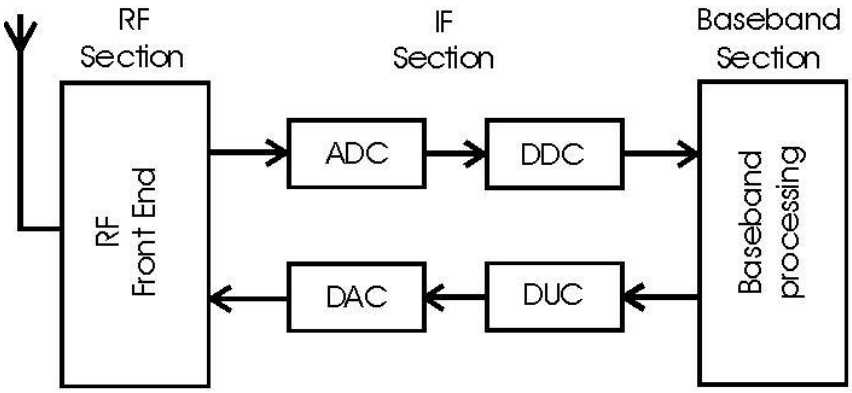

Figure 3.1: Block diagram of SDR.

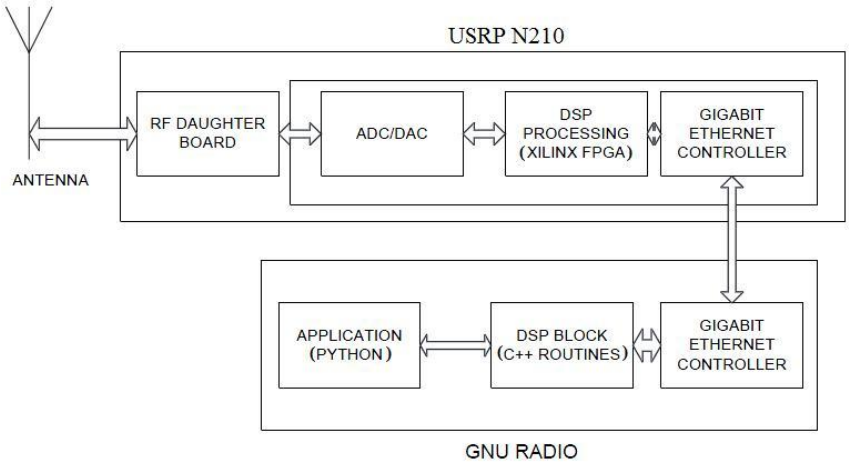

Figure 3.2: Block diagram for the operation of USRP with GNURadio.

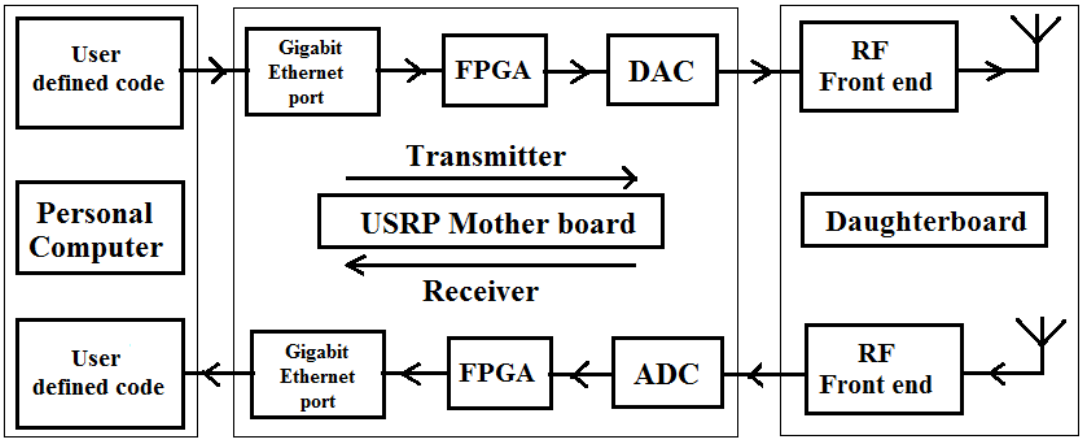

Figure 3.3: Block diagram of USRP [12].

A block diagram of a USRP-based SDR transceiver executing a GNURadio based application is shown in Figure 3.2. The USRP kit is the hardware interface and GNURadio is used for the baseband signal processing tasks.

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.

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 GNU Radio

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

3.2.1 What does GNU Radio do?

It does all the signal processing. You can use it to write applications to receive data out of digital streams or to push data into digital streams, which is then transmitted using hardware.

GNU Radio has software equivalents of real world radio system components like filters, demodulators, equalizers, etc. These are usually referred to as blocks. You can create a complex system by connecting various blocks. If you cannot find some specific blocks, you can even create your own blocks and add them.

Most of GNU Radio has been implemented using the Python programming language, and the performance-critical parts have been implemented using C++. Typically, a GNU Radio user writes his applications in Python, unless he has some performance-critical needs. Thus, GNU Radio gives its users an easy-to-use, rapid application development environment.

3.2.2 GNU Radio with USRP

The USRP and the host computer make up the hardware part of the SDR system. The host computer must run a compatible software package such as GNU Radio or Simulink to complete the SDR system. In this project we are using GNU Radio as the software platform.

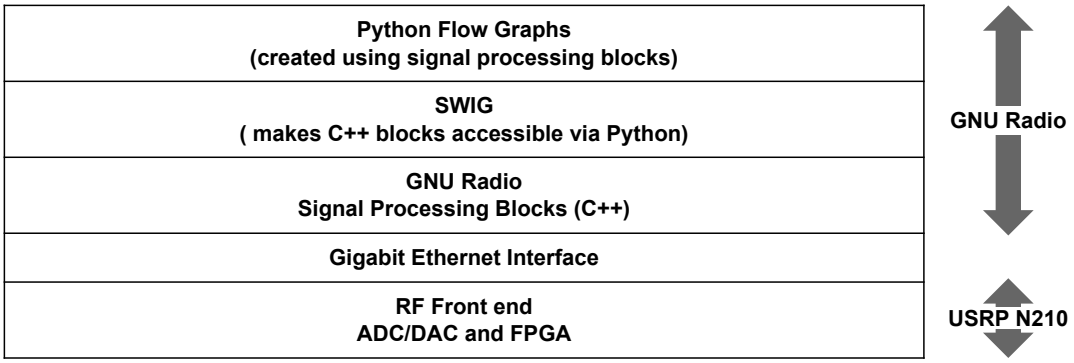

Figure 3.4: Architecture of GNU Radio

GNU Radio communicates with the USRP through the USRP Hardware Driver (UHD) software. The UHD provides a host driver and an Application Programming Interface (API) for the USRP. GNU Radio uses the UHD to set user-specified parameters like RF center frequency, antenna selection, gain, sampling rate, interpolation, decimation, etc.

Chapter 4

OpenBTS

OpenBTS is a software implementation of a GSM access point [13]. It allows common GSM-compatible mobile phones to be used a SIP endpoints to VoIP-based networks. It implements the lower three layers of the industry-standard GSM protocol stack. OpenBTS is an open source software written in C++. Some additional real-time components of it are written in Erlang.

OpenBTS was first developed with a goal to provide low-cost and easily deployable GSM networks in poor and rural ureas.

4.1 GSM architecture of OpenBTS

In an OpenBTS based network, the layers of conventional GSM network above layer 3 are replaced by OpenBTS itself. The functions of the BSC are handled internally. The call handling functionalities of the MSC are handed over to a VoIP softswitch or PBX like Asterisk. In fact, multiple OpenBTS networks can be set up sharing a common VoIP softswitch or PBX [13].

The GSM-based Um interface of OpenBTS does not use any standard GSM hardware. Instead OpenBTS uses software-defined radio transceivers for its Um interface. The USRP from Ettus Research was the first such hardware device to be used for OpenBTS Um interface [13].

4.2 The OpenBTS Application Suite

The OpenBTS Application Suite comes with several software applications that are listed as follows:

- **OpenBTS**
- **Transceiver**
- **SMQueue**
- **Asterisk**
- **SIPAuthServe**

Besides these, there are optional services supported through external servers and interfaced to OpenBTS through various protocols. For example, the RRLP server.

4.2.1 OpenBTS

The OpenBTS application contains various functions beginning from Layer 1 upto Layer 3/Layer 4. The Layer 1 functions are:

- TDM functions
- FEC functions
- closed loop power and timing controls

LAPDm is the only Layer 2 function implemented in the OpenBTS application.

The Layer 3 functions are:

- radio resource management
- mobility management
- call control

GSM-SIP gateway for text messaging is the Layer 4 function included in OpenBTS.

OpenBTS itself does not contain any speech transcoding functions above the L1 FEC parts.

4.2.2 Transceiver

The transceiver application performs the radiomodem 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 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 an MS from any source.

4.2.4 SIP router/PBX

OpenBTS uses a SIP router or PBX to perform the call control functions that are normally performed by the MSC in a conventional GSM network. These switches also provide transcoding services.

The SIP router used in OpenBTS is Asterisk by default. Though there are other PBXs available in the market like Yate, FreeSwitch, etc.

4.2.5 SIPAuthServe

An application that implements Subscriber Registry, the database of subscriber information that replaces both the Asterisk SIP registry and the GSM Home Location Register (HLR) found in a conventional GSM network.

4.3 Network organization

In the simplest network, with just a single access point, all the applications run on the same embedded computer as shown in figure 4.1.

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

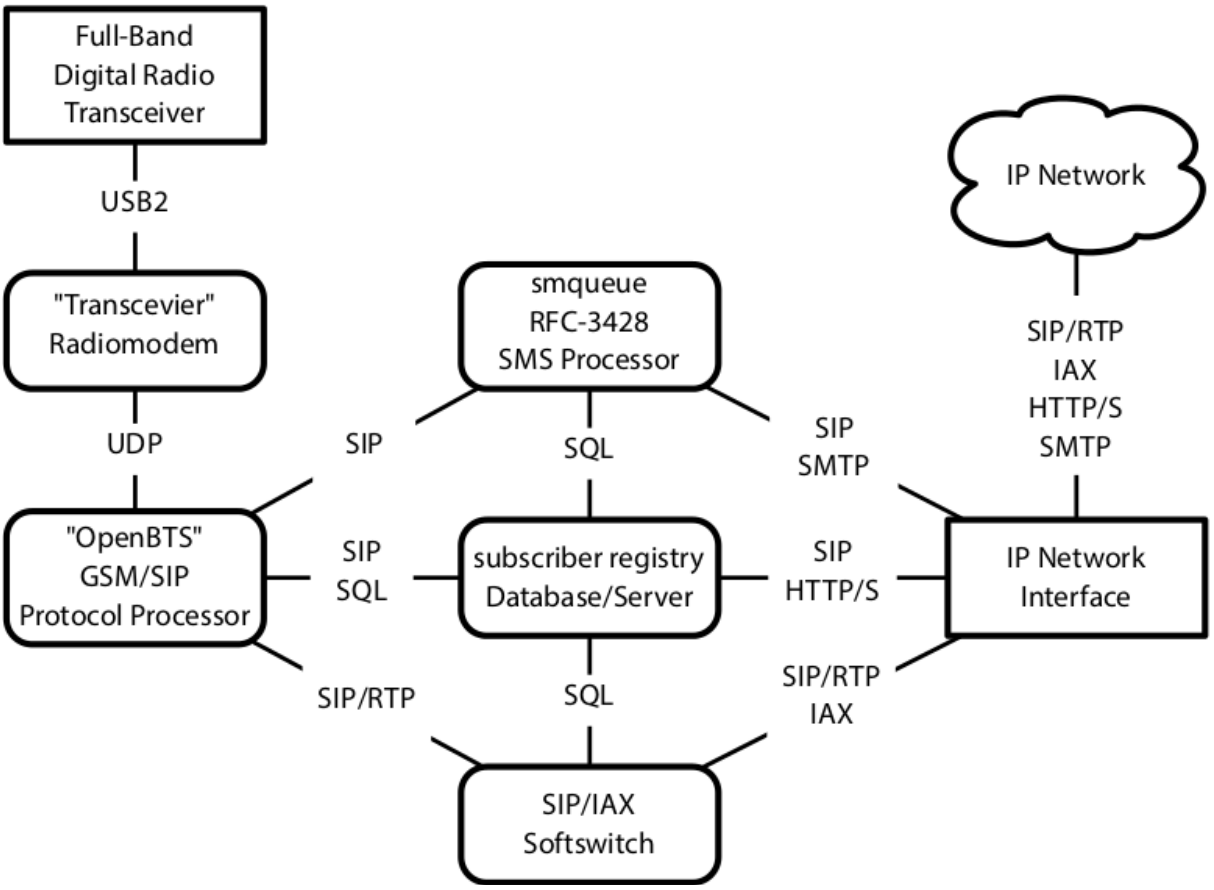

Figure 4.1: Components of the OpenBTS application suite and their communication channels as installed in each access point. Sharp-cornered boxes are hardware components. Round-cornered boxes are software components [14].

the filesystem, so they 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.

4.4 Asterisk

Asterisk is an open source software implementation of a Private Branch Exchange (PBX). Just like a real-world PBX, it allows attached phones to call each other and to connect to other telephone services like the PSTN and the VoIP. It supports various VoIP protocols like Session Initiation Protocol (SIP), H.323, etc. Besides VoIP protocols, it also supports traditional protocols like ISDN and SS7.

OpenBTS uses Asterisk to handle speech calls.

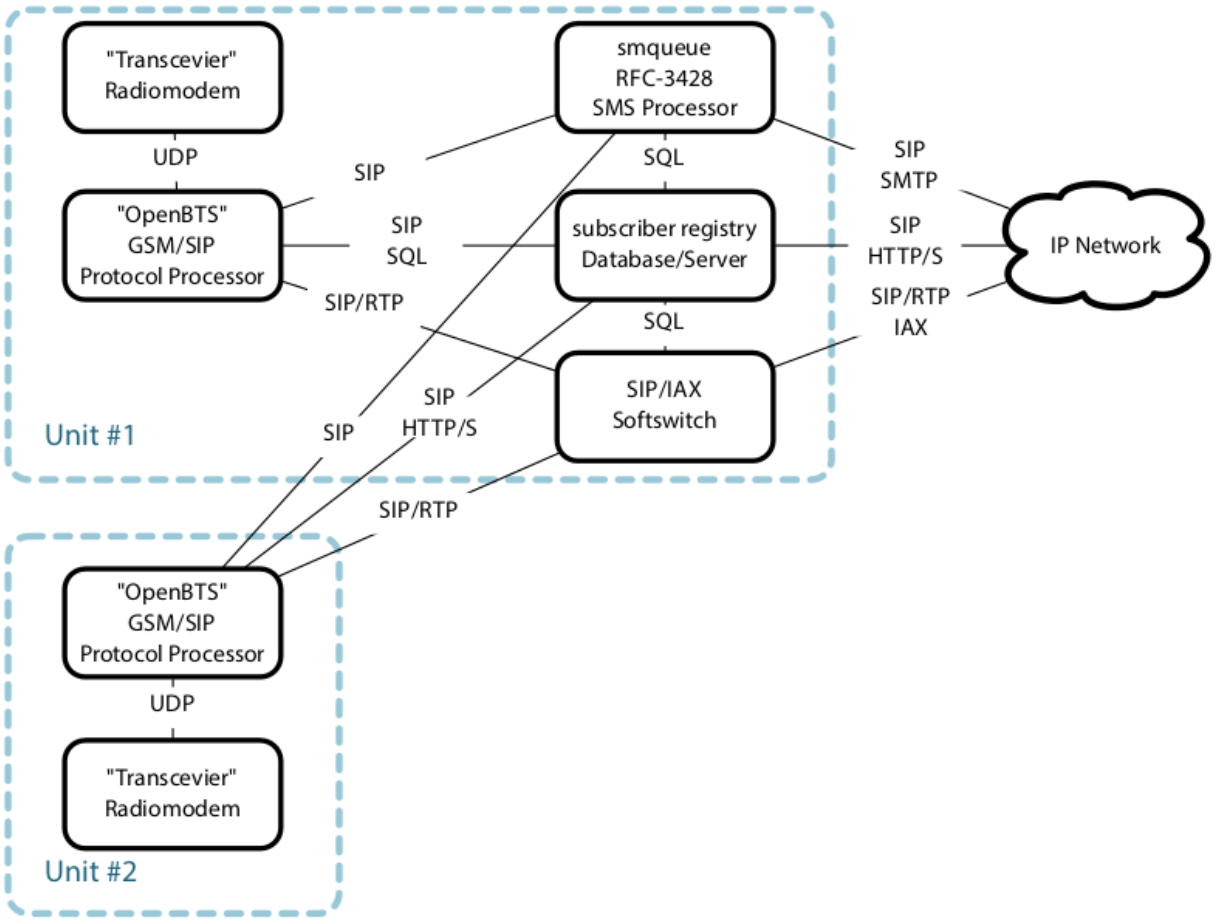

Figure 4.2: Two access points with unit #1 providing servers for both [14]

4.5 Configuration of Asterisk

Every phone operating in a GSM network has a Subscriber Identity Module (SIM) card. Every SIM has an identifier called IMSI, which stands for International Mobile Subscriber Identity.

Asterisk requires a unique name for each user. We use the IMSI number for the name of every Asterisk user because the IMSI numbers are unique to each GSM network user.

In order to register a SIM to an OpenBTS network, we need to edit two configuration files named sip.conf and extensions.conf. These files are located in the folder /etc/asterisk/ for a default installation of Asterisk.

We also have to update the database sqlite3.db located in /var/lib/asterisk/sqlite3dir/.

4.5.1 sip.conf

The sip.conf file contains user device configuration for every user of the Asterisk system. This file contains a section for each user. A typical section is as follows:

```
[IMSI123451234512345]
callerid=1111
canreinvite=no
type=friend
context=sip-external
allow=gsm
host=dynamic
dtmfmode=info
```

The first line of each section is the device name (aka extension name in Asterisk). We used the IMSI number of each SIM prefixed with the word "IMSI" just to ensure that the device names are unique to each user.

The `callerid` option sets dialling number of the device. The option `host` is used for locating the user device in the network. Setting the `host` as *dynamic* tells Asterisk that the user device will update its location to Asterisk automatically and rids us of having to define the address statically. Otherwise we can define the addrsess of the device statically by setting the value of `host` to a specific IP address.

The option `type` is used to determine how the user is matched to the incoming request for a configuration entry. The `type` option can take any of the following values:

- **peer -** The incoming request is matched using the IP address and the port number.

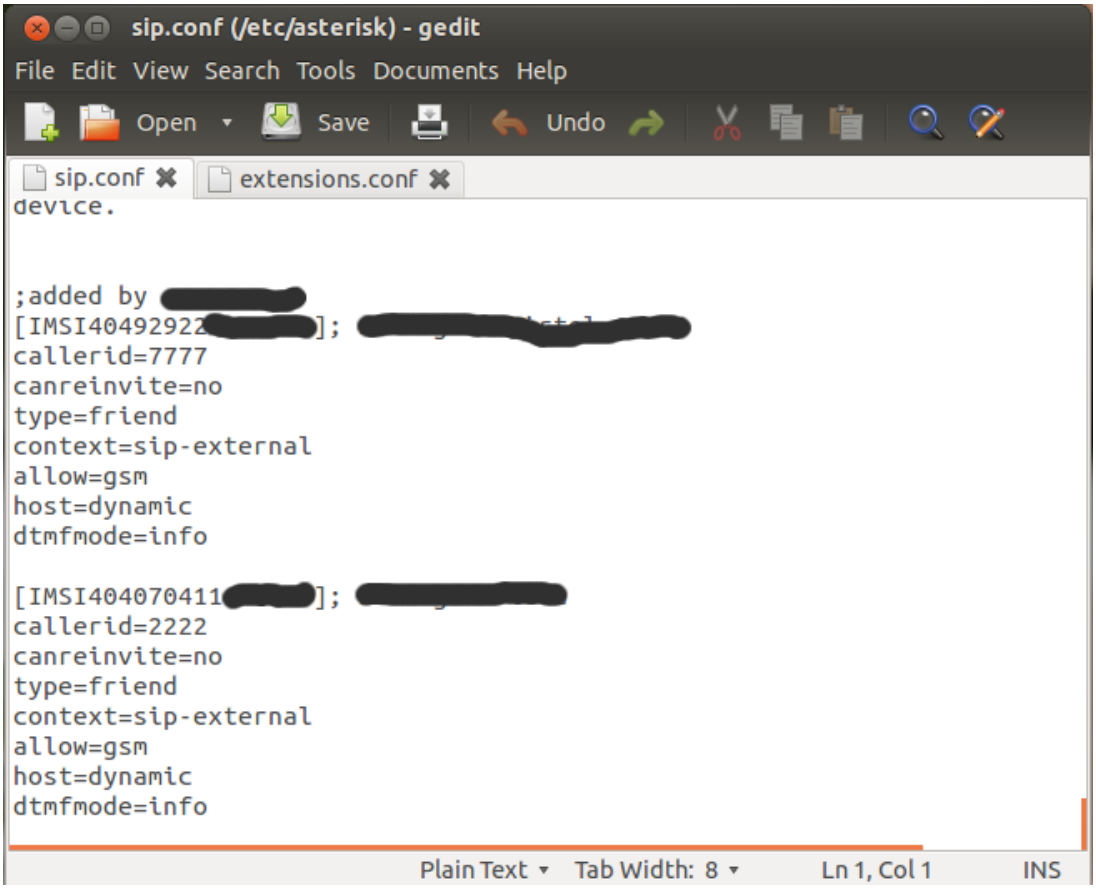

Figure 4.3: Screenshot for sip.conf

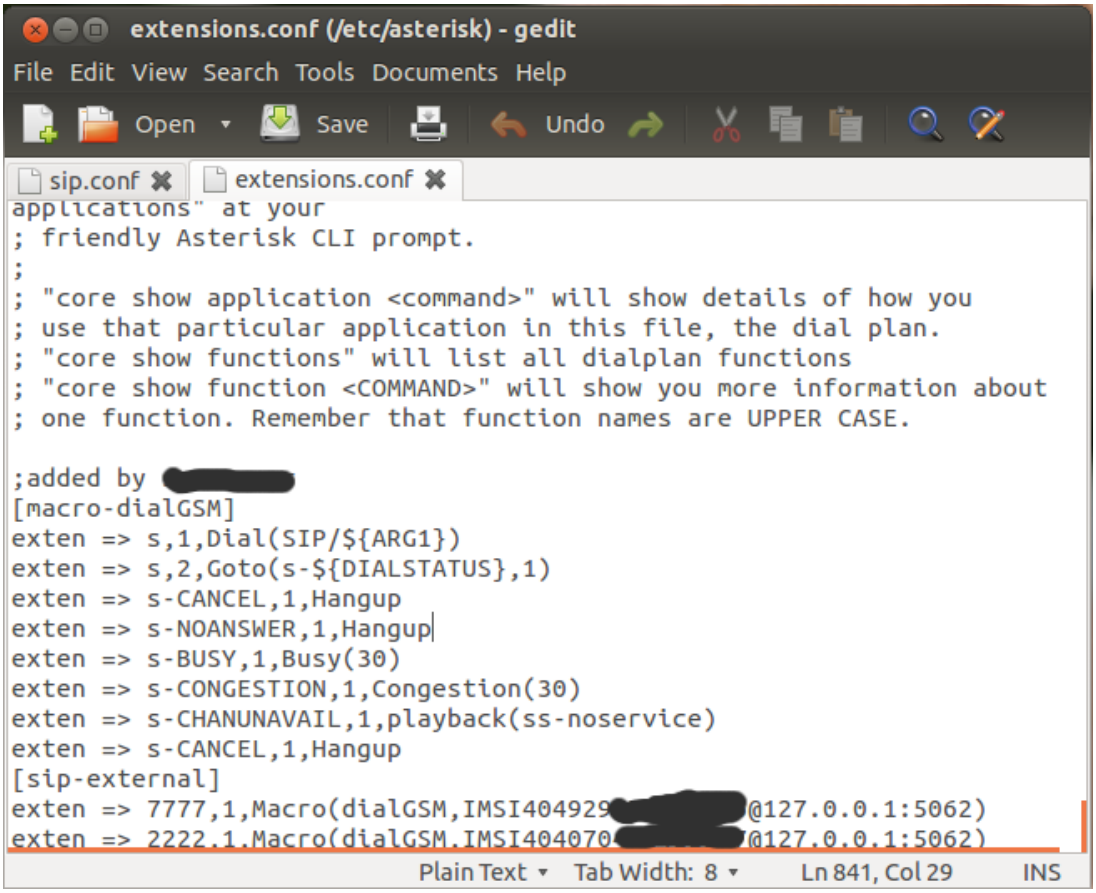

Figure 4.4: Screenshot for extensions.conf

- **user -** The incoming request is matched using the device name of the user.
- **friend -** The incoming request is matched on the name first, and then the IP address.

The requested user is handled by the dialplan in the `context` of the device configuration. The dialplan configurations are contained in the file extensions.conf. In our experiment, we have named the `context` as *sip-external* and we have put all the phones used in the experiment under this `context`. So, for each phone in our experiment, Asterisk will be using the dialplan listed in the extensions.conf file under the `context` of *sip-external*.

The option `allow` controls the audio codecs that are allowed for the phone.

4.5.2 extensions.conf

This file defines the dialplan followed for each user. The dialplan is a form of scripting language. It contains instructions that Asterisk follows in response to some external triggers.

For our experiment, we added a macro to extensions.conf to avoid having to repeat the

same dialplan for each user/extension. The macro is listed below:

```
[macro-dialGSM]
exten => s,1,Dial(SIP/${ARG1})
exten => s,2,Goto(s-${DIALSTATUS},1)
exten => s-CANCEL,1,Hangup
exten => s-NOANSWER,1,Hangup
exten => s-BUSY,1,Busy(30)
exten => s-CONGESTION,1,Congestion(30)
exten => s-CHANUNAVAIL,1,playback(ss-noservice)
exten => s-CANCEL,1,Hangup
```

Then we configured each of our extensions under the `context` of *sip-external* as in the example that follows:

```
[sip-external]
exten => 1111,1,Macro(dialGSM,IMSI1234512345123450127.0.0.1:5062)
exten => 2222,1,Macro(dialGSM,IMSI1234512345123120127.0.0.1:5062)
exten => 9999,1,Macro(dialGSM,IMSI1234512341234120127.0.0.1:5062)
exten => 5555,1,Macro(dialGSM,IMSI1234123412341230127.0.0.1:5062)
```

4.5.3 sqlite3.db

Asterisk also has a database in *sqlite* format storing all the *caller id to device name* mapping data and the user device configuration data. The table `dialdatatable` contains the mapping from each device name to the corresponding caller id of the subscriber/user. The `sip_buddies` table on the other hand has all the user device configuration data already listed in sip.conf.

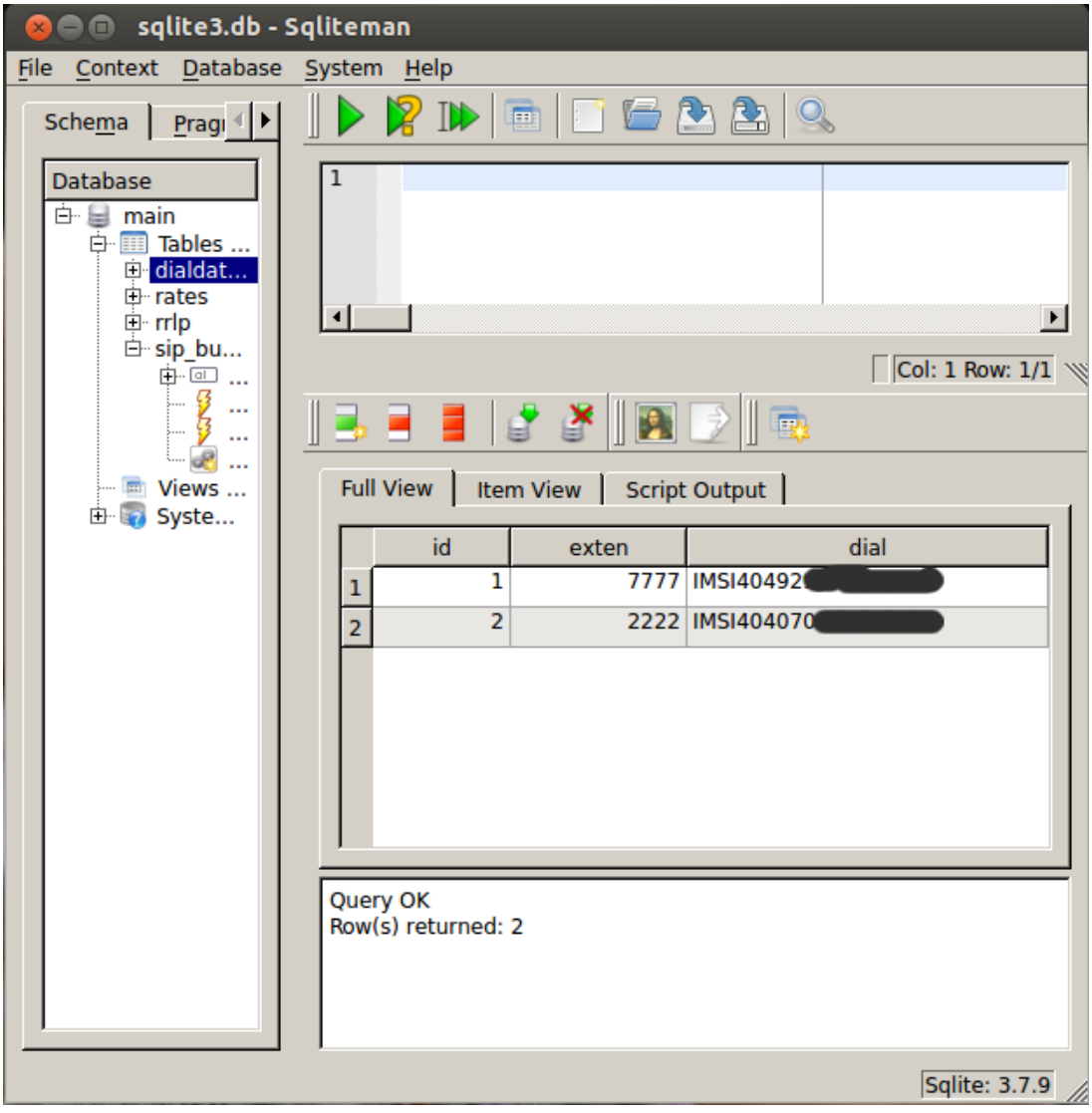

Figure 4.5: Screenshot for `dialdatatable`

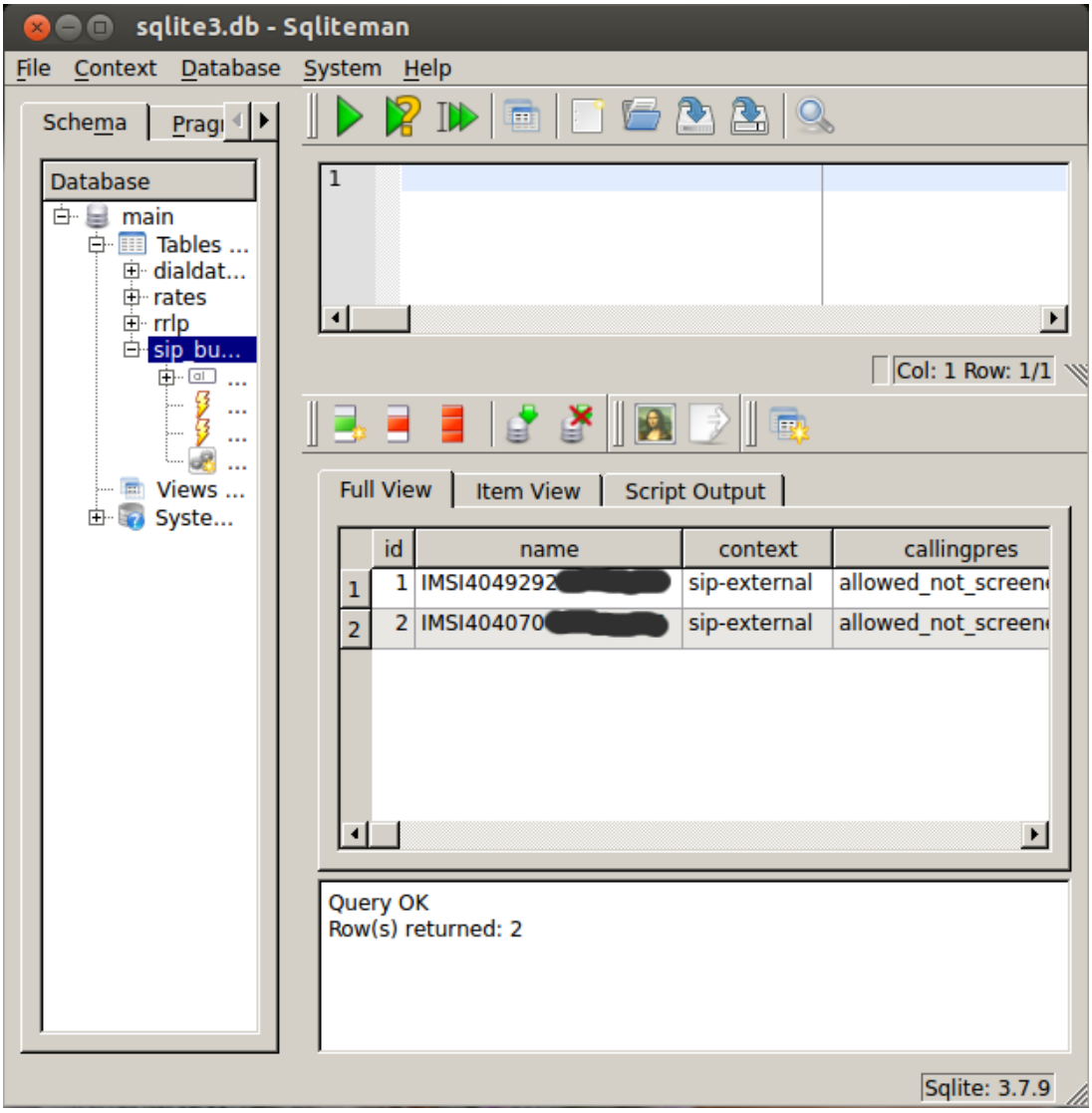

Figure 4.6: Screenshot for `sip_buddies`

Chapter 5

Spectrum sensing

Spectrum sensing is the main task in the entire operation of cognitive radio. Spectrum sensing is defined as the finding of spectrum holes in the local neighborhood of the cognitive radio receiver. Spectrum holes are the underutilized (in part or in full) subbands of spectrum at a particular time in a specific location. Moreover for cognitive radio to fulfil its potential in solving the problem of spectrum underutilization, the spectrum sensing method used should be reliable and computationally feasible in real-time [15].

There are many spectrum sensing techniques available. Three important ones of them are as follows:

- Energy detection
- Matched filter detection
- Cyclostationarity detection

5.1 Energy detection

Conventional energy detector is made up of a low pass filter, an A/D converter, a square law detector and an integrator (Figure 5.1a). This implementation is not flexible enough, especially in the case of narrowband signals and sinewaves. Also, this requires a pre-filter matched to the bandwidth B of the signal to be scanned [16].

So, an alternative implementation is generally used where we find the squared magnitude of the FFT using the Average Periodogram method (Figure 5.1b). In this architecture, we can alter the bandwidth of frequencies scanned just by taking the required number of FFT bins.

Spectrum sensing can be viewed as a binary hypothesis-testing problem [17]:

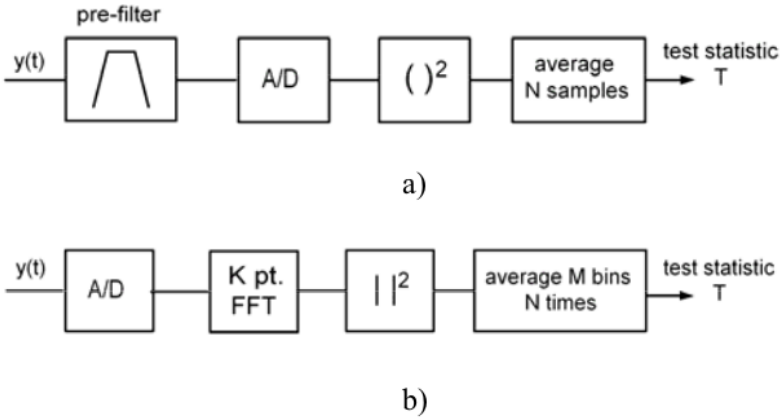

Figure 5.1: (a) Implementation using analog filter and square law device
(b) Implementation using periodogram.
Source: [16].

- H_0: primary user is absent
- H_1: primary user is present

The detection is basically to decide between the following two hypotheses,

$$\begin{aligned} H_0 : \quad & x(t) = n(t), \\ H_1 : \quad & x(t) = h(t)s(t) + n(t), \end{aligned}$$

where $x(t)$ is the received signal, $s(t)$ is the primary user signal, $h(t)$ is the complex channel gain and $n(t)$ is the additive white gaussian noise (AWGN) with zero mean and variance σ_n^2. Generally $h(t)$ is assumed to be constant h_0 for the detection period. A statistics Y is computed by taking energy samples over a time T in a bandwidth B and compared with a predefined threshold γ for making the decision.

Energy detection is one of the simplest methods of spectrum sensing. It is the optimal detection method for unknown signals. Moreover, it is widely used because its computationally less resource-intensive.

But this method is not without problems. It is always to difficult to determine a threshold that will work for all situations. This method cannot say whether an interfering signal is from a primary user or a secondary user. Low SNR (signal to noise ratio) signals cannot be detected easily.

The frequency resolution can be improved by increasing N, the number of FFT points, but then this requires more samples and thereby takes more time.

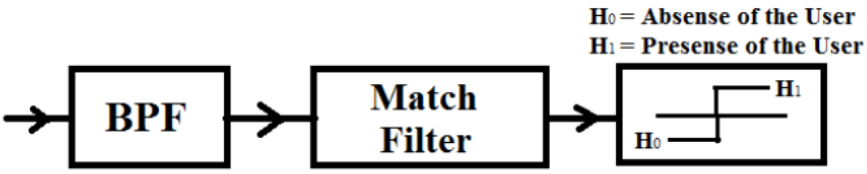

Figure 5.2: Block diagram of Matched filter implementation.

5.2 Matched filter detection

A matched filter is a linear filter to maximize the output SNR of a received signal. It is the optimum filter to detect signals that are known a priori [18].

In matched filtering, the received signal is first band pass filtered and then convolved with the impulse response. The impulse response h here is the reference signal itself [19]. Matched filtering is so called because the impulse response is matched to the reference signal.

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

Here, $x[k]$ is the received or unknown signal with additive noise. The goal of matched filtering is to enhance the component of reference signal in the received signal and to suppress the noise. This works best when the additive noise is completely orthogonal to the reference signal or when the noise is completely Gaussian. In practice though, the noise doesn't turn out to be purely Gaussian.

Matched filtering requires only $O(1/\text{SNR})$ samples to meet a given P_d, probability of detection requirement. Thus it requires less detection time.

But, matched filtering requires us to have a priori information about the received signal. This technique requires demodulating the received signal. For demodulation, we require information like bandwidth, operating frequency, modulation type, pulse shaping, packet format, etc. Demodulating the received signal correctly also requires timing synchronization, carrier synchronization, etc. It might still be possible to achive this because the received data carry preambles, synchronization data, etc.

This method requires a specific type of receiver for every primary user. Implementing this method on a receiver will increase the complexity and the power consumption greatly.

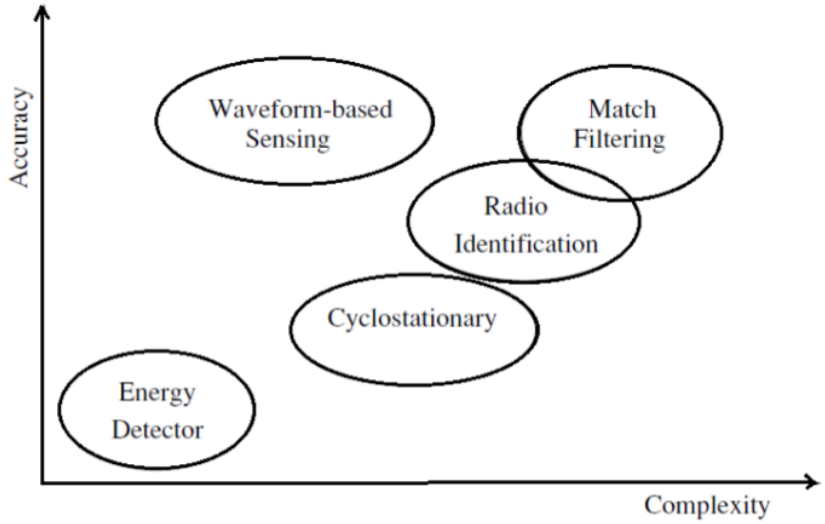

Figure 5.3: Comparison of some spectrum sensing methods

5.3 Comparison of sensing techniques

Energy detection technique is the simplest method of all but it is also the most error prone. On the other hand, matched filtering is the most complex but it is very accurate. Cyclostationarity detection is more complicated than energy detection but it is more accurate. There is no ideal detection method. There are always compromises and tradeoffs to be made. Figure 5.3 shows a comparison of some spectrum sensing methods.

There are other methods of spectrum sensing that are more involved. For example multitaper spectral estimation, wavelet based detection, waveform based detection, etc.

5.4 Implementation of energy detection technique

In this section, we give a brief mathematical overview of the Average Periodogram Analysis method, that we have used in this project to calculate the energy in various operating frequencies. We also dwell on the wideband spectrum analyzer which happens to use the Average Periodogram Analysis method for doing its job.

5.4.1 Average periodogram analysis

This method applies the FFT (fast fourier transform) algorithm to the estimation of power spectra by sectioning the input time-domain data, finding the modified periodogram for each section and then averaging them to get the final spectral estimate. Since this method transforms the input data into smaller chunks, it requires less storage memory

to implement this algorithm and also fewer computations. This is certainly an advantage and makes this method work faster. This method is also good for nonstationarity tests because it has the potential to resolve the data in the time domain [20].

While computing the modified periodogram the data is smoothened using a window because sharp transitions of data corrupt the modified periodograms. The window used in our project is the Blackman-Harris window.

Assume $X(n), n = 0, ..., N-1$ is a sample from a stationary sequence. We take segments, possibly overlapping, of length L that are D units apart from $X(n)$. Let $X_0(n)$ be the first such segment. Then $X_1(n) = X_0(n + D)$ and $X_2(n) = X_1(n + D)$ and so on, where $n = 0, ..., L-1$. We take K segments such that those K segments almost use up all the data in $X(n)$ i.e. $(K-1)D + L = N$. Supposing $W(n), n = 0, ..., L-1$ is the window, we calculate the Discrete Fourier Transform (DFT) of each segment to get

$$A_k(m) = \frac{1}{L}\sum_{n=0}^{L-1} X_k(n)W(n)e^{-j2\pi nm/L} \qquad k = 0, ..., K-1$$

The K modified periodograms are computed as

$$I_k(f_m) = \frac{L}{U}\left|A_k(m)\right|^2 \qquad k = 0, ..., K-1$$

where

$$f_m = \frac{m}{L} \qquad m = 0, ..., L/2$$

and

$$U = \frac{1}{L}\sum_{n=0}^{L-1} W^2(n)$$

The spectral estimate is the average of these periodograms,

$$P(f_m) = \frac{1}{K}\sum_{k=0}^{K-1} I_k(f_m)$$

5.4.2 Wideband Spectrum Analyzer

GNURadio comes with a default spectrum analyzing program named "usrp_spectrum_sense.py". This program can be used as a template for other programs that involve spectrum analysis.

The output of this code is the squared magnitude of the FFT spectrum. For a typical FFT bin [i] the output is $Y[i] = re[X[i]] * re[X[i]] + im[X[i]] * im[X[i]]$. The power for a particular band can be calculated by summing these values for the correct number of

bins that cover that bandwidth. The energy in a particualar frequency corresponding to a particular bin [i] is the square root of $Y[i]$.

N time samples of $x(t)$ sampled at a sampling frequency of F_s are required to use N point complex FFT $X(\omega)$ analysis. To reduce spectral leakage, an appropriate window function like Blackman-Harris window is to be chosen and applied to these time samples. The output of the FFT represents the spectrum content as follows:

- The first value of the FFT output ($bin0 == X[0]$) corresponds to the energy in the centre frequency.
- The first half of the FFT spectrum ($X[1]$ to $X[N/2-1]$) corresponds to the positive baseband frequencies i.e. from the centre frequency to $+F_s/2$.
- The second half ($X[N/2]$ to $X[N-1]$) corresponds to the negative baseband frequencies, i.e. from $-F_s/2$ to the centre frequency.

For the purposes of our project, we used to collect N = 1024 samples using a radio tuner centered at the frequency of our interest, say 900 MHz. The number of FFT points, N should be a power of 2 for the FFT algorithm to work fast, so we set N = 1024 for our work. We set the sampling frequency to be 1 MHz because all we are required to do is scan GSM bands that are of 200 KHz each. The frequency resolution thus turns out to be: 1 MHz / 1024 = 976.56 KHz. The decimation factor is defined as the dsp rate divided by the sampling rate. The dsp rate is the actual hardware sampling rate of the ADC in the USRP kit and was 100 MHz in our case. The UHD driver (driver software for the USRP kits) requires the decimation factor to be an integer value. So we set our sampling rate to 1 MHz which made the decimation factor to be 100 MHz / 1 MHz = 100, an integer value. For example if we set the sampling rate as 3 MHz then the USRP's driver would complain because the decimation factor 100 MHz / 1 MHz = 33.33 turns out to be a non-integral value.

As has been said earlier, a GSM band is of 200 KHz. Hence, to calculate the energy in a particular GSM band of interest, we have to find the average of all the bin values which lie in the 200 KHz band centered at that frequency.

Chapter 6

OpenBTS based Cognitive Radio Test-Bed

In this project we try to demonstrate the coexistence of primary and secondary users in the same GSM frequency band by making the secondary users switch to some neighboring unoccupied GSM frequency band (spectrum hole) as soon as the primary users of that frequency band makes a call. A spectrum hole is a frequency band/channel that have been licensed to the primary users but is not being used at that particular space and time. This allows secondary users to make use of already licensed frequency bands instead of having to allot them completely new frequency bands altogether.

In the first phase of the project, we implemented a 2-frequency system where the secondary system had an option of switching into one of 2-frequency channels depending on which one was free. We expanded this to a 4-frequency system with two primary systems in the second phase. The secondary would search for an unused frequency band among these four frequencies, two of which always remain used.

6.1 The 2-frequency system

6.1.1 Experimental setup of the 2-frequency system

The hardware and software components used in this experiment are the following:

- **A primary BTS** – This is a Linux laptop running OpenBTS software with 1 USRP as the OpenBTS radio interface. The USRP hardware kit has a WBX 50-2200 MHz RX/TX daughterboard in it. Two mobile phones (primary users) are connected to the OpenBTS network running in this primary BTS system.

Figure 6.1: Experimental setup of the 2-frequency system

- **A secondary BTS** – This is an Ubuntu desktop running OpenBTS and GNURadio software. Two USRP kits are connected to this machine, one as the OpenBTS radio interface and the other as the GNURadio radio interface. The GNURadio software is used for the spectrum sensing. So, here the OpenBTS software with its radio interface acts as a Base Transceiver Station (BTS) while the GNURadio software alongwith its radio interface acts as a spectrum sensor. Each of the two USRP kits has a WBX 50-2200 MHz RX/TX daughterboard. Two other mobile phones (secondary users) are connected to the OpenBTS network running in this secondary BTS.

The secondary BTS system has cognitive capabilities. It was a challenge to make OpenBTS and GNURadio run simultaneously in the same computer and make them communicate with each other. GNURadio keeps sensing the spectrum used by the secondary users continuously in the background and takes decisions whether to switch the frequency band of the secondary BTS or not, depending upon the energy level in the frequency band in which it is running.

6.1.2 Testing of the 2-frequency system

First we choose any two GSM frequency bands say 945 MHz (F_1) and 950 MHz (F_2). The primary users are made to occupy F_1. Then we let the secondary users come into F_1. This makes the energy level in F_1 go high, which gets detected by the spectrum sensor of the secondary BTS. So, the secondary BTS moves out of F_1 and switches its frequency to F_2. Similarly, now if the primary users are made to come into F_2, the secondary switches back to F_1.

In this experiment we don't have the situation where both F_1 and F_2 remain occupied because there is only one set of primary users. Therefore, the secondary also doesn't check

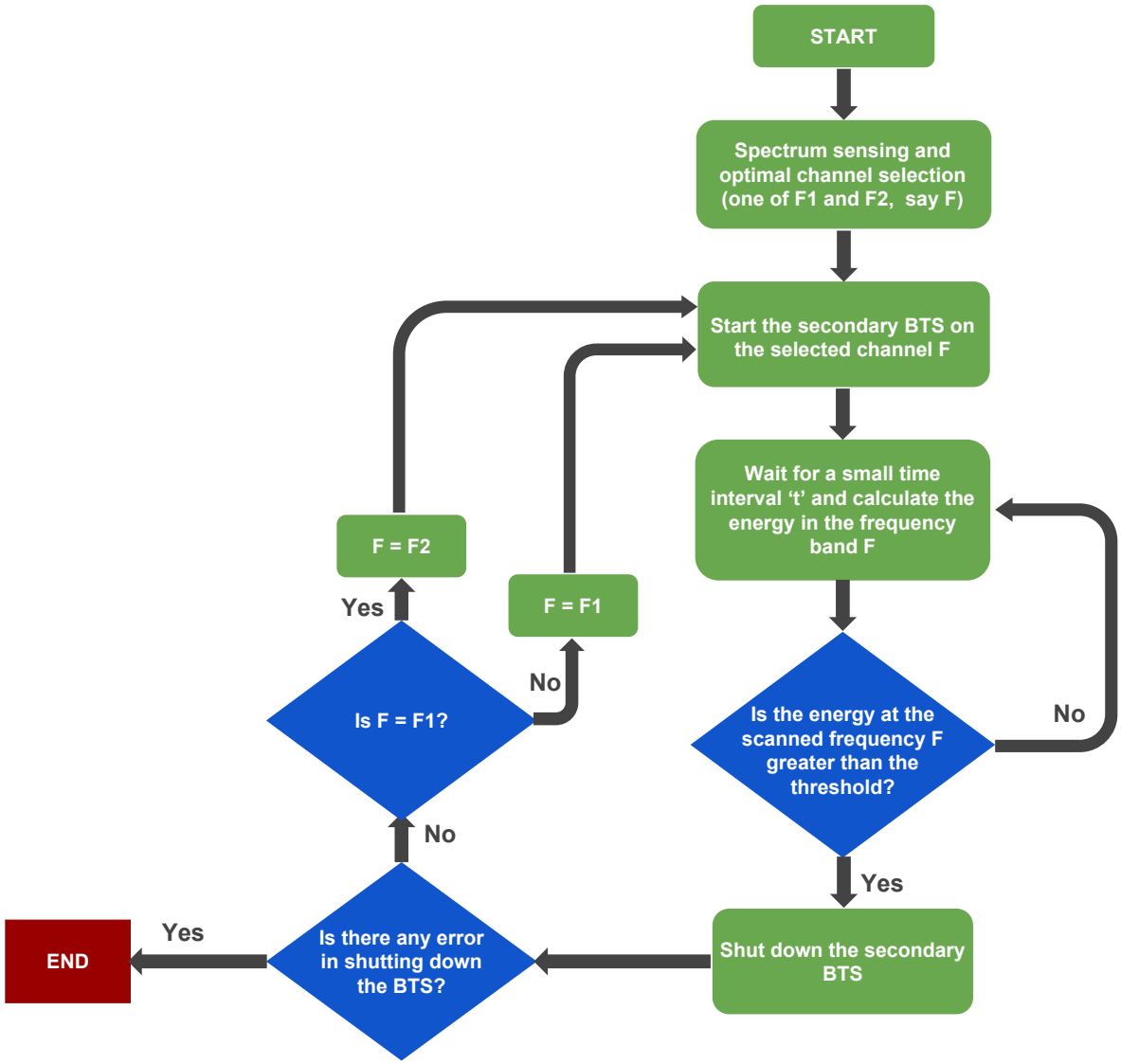

Figure 6.2: Flowchart for the 2-frequency system

the energy level in a channel before taking the decision to switch into that channel.

6.2 The 4-frequency system

As has been said earlier, in second phase, we expanded the 2-frequency system to a 4-frequency one. The frequency channels are $F_1 = 936$ MHz, $F_2 = 943$ MHz, $F_3 = 950$ MHz, $F_4 = 957$ MHz. We also had two primary systems instead of just one this time. We also used a method known as CUSUM for peak detection in this case.

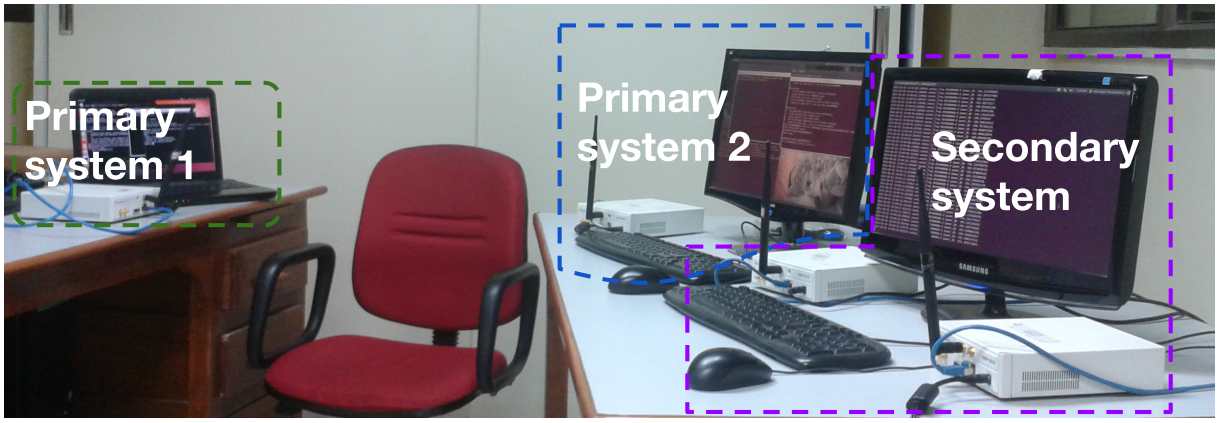

Figure 6.3: Experimental setup of the 4-frequency system

6.2.1 Experimental setup of the 4-frequency system

The tools used in this experiment are as follows:

- **Two primary BTSs** – One is a laptop and the other one is a desktop. Both of them runs Ubuntu as the Operating System. Each one of them runs OpenBTS with a USRP kit as its radio interface. A pair of mobile phones are connected to each one of them.
- **A secondary BTS** – This is the same as in the 2-frequency system. It runs OpenBTS and GNURadio on two different USRP kits. A pair of mobile phones (secondary users) are connected to its OpenBTS network.

One of the primary BTSs has a USRP with a SBX 400-4400 MHz RX/TX daughterboard, the rest of the USRPs all had a WBX daughterboard as before.

6.2.2 Testing of the 4-frequency system

Initially we make one of the primary systems operate in F_2. And the secondary is made to operate in F_1. Now we let the other primary come into F_1. The secondary senses it and attempts to switch to F_2 because the secondary is programmed to check F_1, F_2, F_3, F_4 serially in that order. After checking F_4 the secondary checks F_1, F_2, F_3, ... again and so on the cycle continues.

But the frequency F_2 happens to be occupied by the one of the primary systems. So, the secondary moves ahead to F_3 which is unoccupied and utilizes that channel.

Unlike the 2-frequency system, in this case the secondary always checks the availability of a channel before deciding to switch into it. In the 2-frequency system, it was assumed that one of the two channels is always unoccupied.

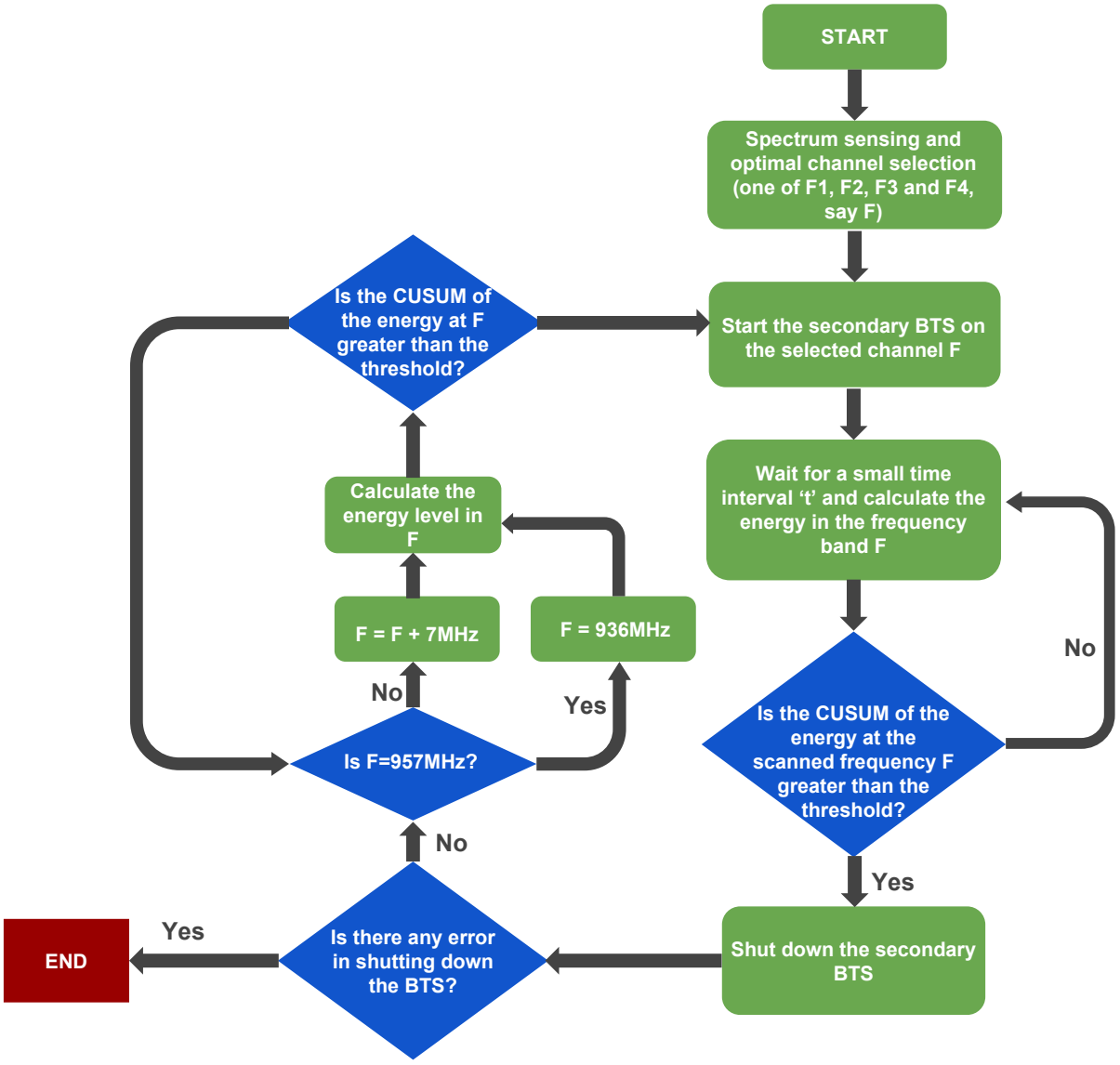

Figure 6.4: Flowchart for the 4-frequency system

The spectrum sensing is done using the energy detection based method. For making decisions whether to switch the frequency of the secondary BTS or not, a threshold of energy level is required to be set. If the energy level in a given channel is beyond the thresho ld, it implies that the primary users are also using that channel i.e. that channel is already occupied by primary users.

To choose a threshold energy level, we took various readings of the noise floor, energy when only primary users were active, energy when only secondary users were active and also when both primary users and secondary users were active simultaneously for a short duration. The energy level threshold depends on the distance of the mobile phones from the BTS. We can subdue this dependency on the distance by setting the threshold quite low so that even if the users move farther away the decision making is not affected.

6.3 CUSUM method

The CUSUM (or cumulative sum control chart) method is used here to detect the peak in energy levels [21]. This method is used to ascertain that the peak in the energy levels in a given channel are not just due to some irrelevant reasons like random fluctuations in the noise power, etc.

CUSUM is a technique used for monitoring change detection. It involves calculating the cumulative sum, which is what makes it sequential. The samples from a process x_n are assigned weights ω_n and summed in the following way:

$$S_0 = 0$$
$$S_n = max(0, S_n + x_n - \omega_n)$$

When the value of S exceeds a certain threshold value, a change in value has been found. However, this formula detects only a change in the positive direction. For the negative direction, the *max* operation has to be replaced by the *min* operation. In this case the change has been found when the (negative) value is below the threshold value.

6.4 Achievements

What follows is a brief overview of the things done and the challenges faced during the course of the project.

1. We began by trying to get acquainted with Cognitive Radio. We carried out a

literature survey on Cognitive Radio and also on the ongoing research work in the field of Cognitive Radio.

2. Since GNURadio applications are written in the Python programming language, we had to brush up our knowledge of the Python language.

3. We familiarized ourselves with the GNURadio software package. Even the installation procedure of GNURadio was a little challenging for us back then because GNURadio required a manual installation. We tweaked and played around with the already existing codes of signal processing blocks that come by default with GNURadio.

4. We acquainted ourselves with the USRP hardware kit, which happens to be the prime hardware used in our project. An outdoor experiment was carried out to estimate the range of the USRP kit. This is largest distance upto which mobile phones are able to get reasonable signal quality from the USRP.

5. Then we got familiar with the OpenBTS software. We had a hard time installing and configuring OpenBTS to work. But eventually we managed to get our SIMs registered to the OpenBTS network and also to make calls and to text messages between the phones. The USRP kit is the hardware radio interface for the OpenBTS software.

6. Spectrum sensing lies at the very heart of the entire operation of Cognitive Radio. So we surveyed literature on various methods of spectrum sensing. We made a decision to go with Energy detection based method because it is comparatively less complicated and computationally less resource-expensive. Average Periodogram Analysis is a technique used in energy detection based spectrum sensing. So we carried out a study on Average Periodogram Analysis.

7. After all these we prepared our problem statement and designed a flow graph presenting our proposed algorithm to solve the problem. Then we started developing the experimental setup that we have covered at the beginning part of this chapter.

8. A key approach in solving our problem was to make GNURadio and OpenBTS run simultaneously in the same computer hardware. This was a little tricky for us. Because we had to figure out if it was possible to run two USRP kits on the same computer. Fortunately, it is possible if the two kits do not use the same IP address. So, we had to configure the kits to use different IP addresses. This was done by burning a different IP address to one of the kits.

9. Next we built a 2-frequency system which had a pair of primary users and a pair of secondary users communicating in parallel. The primary users were to be given

higher priority. So the secondary users would have to switch to a different and unoccupied channel as soon as the primary users started making calls. A detailed description has already been given at the beginning part of this chapter.

10. Then we extended the 2-frequency system to a 4-frequency system which has two pairs of primary users instead of just one. Thus we have demonstrated the coexistence of primary users and secondary users in the same GSM frequency band.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The rapid growth in the number of cell phones and other wireless devices has brought about a scarcity of spectral resources. This is further worsened by the inefficient usage of the spectrum. Cognitive Radio has a lot of potential to solve this problem by finding the spectrum holes and enabling secondary users to utilize them. CR thereby increases the number of mobile device users possible by putting underutilized frequency bands into use. We have demonstrated the capabilities of CR by developing a 2-frequency CR Test-Bed and a 4-frequency CR Test-Bed and successfully testing them out. We have shown that the secondary users yield the frequency channel they have been utilizing for communication as soon as the presence of primary users is detected thus giving a higher priority to the primary users.

7.2 Future work

In the test-beds that we developed, we have used energy detection based spectrum sensing technique. We are measuring the energy at the uplink frequency of the primary BTS to detect the presence of active primary users. But this sensed energy level is dependent on the distance of the primary users from the sensing unit. So, this makes our decision-making criteria for the presence of primary users on a particular frequency band makes distance dependent. In future, this dependency on distance of primary users from the sensor can be removed by resorting to better spectrum sensing techniques.

Also, as soon as the primary users emerge on the scene, the secondary calls are dropped and has to be restarted after switching to a new frequency. Better algorithms can be

developed to avoid this call drop.

Appendix A

Codes

A.1 Code for the 2-frequency system

A.1.1 freq2secondaryBTS.py

This code was written to demonstrate the 2-frequency system. This code was written by modifying the already available program named "usrp_spectrum_sense.py" that comes together with the GNURadio software package. We set the default UHD device address to "192.168.20.2" because that is the IP address of the USRP device we are using as a spectrum sensor. The default samping rate was set to 1e6 i.e. 1MHz. The default FFT size is given as 'sampling rate / channel bandwidth'. We wanted an FFT size of 1024 so we set the bandwidth to 976.56 Hz since 1 MHz / 976.56 Hz $\approx$ 1024.

The class 'my_top_block' was modified by replacing the lines:

```
self.channel_bandwidth = options.channel_bandwidth

self.min_freq = eng_notation.str_to_num(args[0])
self.max_freq = eng_notation.str_to_num(args[1])

if self.min_freq > self.max_freq:
    # swap them
    self.min_freq, self.max_freq = self.max_freq, self.
        min_freq
```

with the lines

```
self.channel_bandwidth = options.channel_bandwidth
```

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

The method 'set_next_freq' of the class 'my_top_block' was modified by replacing

```
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
            self.next_freq = self.min_center_freq
```

with

```
        target_freq = self.up_freq
```

In the code listing that follows we have listed only the functions that we customized and the ones that we added.

```
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 = -59.0



#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


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()
    if downFreq <= 945e6:
        newDownFreq = downFreq + 10e6
    else:
        newDownFreq = downFreq - 10e6

    tb.up_freq = newDownFreq - 45e6
    print "new tb.up_freq: ", tb.up_freq
    startOpenBTS(newDownFreq, tb)
```

A.2 Code for the 4-frequency system

A.2.1 secondaryBTS.py

Most of the code is similar to "freq2secondaryBTS.py". The only modified functions are listed below:

```
def main_loop(tb):
    startOpenBTS(tb.down_freq,tb)


def sub_loop(tb):

    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

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

#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
```

A.3 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.4 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.5 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 Desktop Operating System (version 12.04 and greater) only. For other operating systems, please check 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 please 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 please 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] Simon Haykin. Cognitive radio: Brain-empowered wireless communications. *IEEE Journal on selected areas in communications*, 23(2), 2005.

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

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

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

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

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

[7] `http://wireless.arcada.fi/MOBWI/material/CN_1_2.html`. [Accessed on May 27, 2014].

[8] `http://gnuradio.org/redmine/attachments/download/156/fullnetwork.jpg`. [Accessed on May 27, 2014].

[9] `http://wireless.arcada.fi/MOBWI/material/CN_1_3.html`. [Accessed on May 27, 2014].

[10] `http://en.wikipedia.org/wiki/Software-defined_radio`.

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

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

[13] `http://en.wikipedia.org/wiki/OpenBTS`. [Accessed on June 12, 2014].

[14] https://wush.net/trac/rangepublic/attachment/wiki/WikiStart/OpenBTS-4.0-Manual.pdf. [Accessed on May 27, 2014].

[15] Simon Haykin, David J. Thomson, and Jeffrey H. Reed. Spectrum sensing for cognitive radio. *Proceedings of the IEEE*, 97(5), May 2009.

[16] D Cabric, A Tkachenko, and R W Brodersen. Experimental study of spectrum sensing based on energy detection and network cooperation. In *TAPAS '06 Proceedings of the first international workshop on Technology and policy for accessing spectrum*, August 2006.

[17] Wei Zhang, R.K. Mallik, and K. Letaief. Optimization of cooperative spectrum sensing with energy detection in cognitive radio networks. *IEEE Transactions on Wireless Communications*, pages 5761–5766, December 2009.

[18] http://en.wikipedia.org/wiki/Matched_filter.

[19] P P Bhattacharya et al. A survey on spectrum sensing techniques in cognitive radio. *International Journal of Computer Science & Communication Networks*, 1(2):196–206, 2011.

[20] 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.

[21] http://en.wikipedia.org/wiki/CUSUM.