Energy-Detection Based Spectrum Sensing for Cognitive Radio on a Real-Time SDR Platform

McBath John Rwodzi

University of Cape Town

This thesis is submitted in fulfilment of the academic requirements
for the degree of
Master of Science in Electrical Engineering
in the Faculty of Engineering and The Built Environment
University of Cape Town
2016

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

As the candidate’s supervisor, I have approved this dissertation for submission.

Name: Dr. Mqhele E. Dlodlo, Associate Professor

Signed: ______________________

Date: 2016-07-17

Declaration

I declare that this thesis is my own work. Where collaboration with other people has taken place, or material generated by other researchers is included, the parties and/or materials are indicated in the acknowledgements or are explicitly stated with references as appropriate.

This work is being submitted for the Master of Science in Electrical Engineering at the University of Cape Town. It has not been submitted to any other university for any other degree or examination.

McBath John Rwodzi

Name

30 July 2016

Date

Dedication

To God be all the Glory!

My Papa.

Abstract

There has been an increase in wireless applications due to the technology boom; consequently raising the level of radio spectrum demand. However, spectrum is a limited resource and cannot be infinitely subdivided to accommodate every application. At the same time, emerging wireless applications require a lot of bandwidth for operation, and have seen exponential growth in their bandwidth usage in recent years. The current spectrum allocation technique, proposed by the Federal Communications Commission (FCC) is a fixed allocation technique. This is inefficient as the spectrum is vacant during times when the primary user is not using the spectrum. This strain on the current available bandwidth has revealed signs of an upcoming spectrum crunch; hence the need to find a solution that satisfies the increasing spectrum demand, without compromising the performance of the applications.

This work leverages on cognitive radio technology as a potential solution to the spectrum usage challenge. Cognitive radios have the ability to sense the spectrum and determine the presence or absence of the primary user in a particular subcarrier band. When the spectrum is vacant, a cognitive radio (secondary user) can opportunistically occupy the radio spectrum, optimizing the radio frequency band. The effectiveness of the cognitive radio is determined by the performance of the sensing techniques. Known spectrum-sensing techniques are reviewed, which include energy detection, entropy detection, matched-filter detection, and cyclostationary detection.

In this dissertation, the energy sensing technique is examined. A real-time energy detector is developed on the Software-Defined Radio (SDR) testbed that is built with Universal Software Radio Peripheral (USRP) devices, and on the GNU Radio software platform. The noise floor of the system is first analysed to determine the detection threshold, which is obtained using the empirical cumulative distribution method. Simulations are carried out using MATrix LABoratory (MATLAB) to set a benchmark. In both simulations and the SDR development platform, an Orthogonal Frequency Division Multiplexing (OFDM) signal with Quadrature Phase Shift Keying (QPSK) modulation is generated and used as the test signal. The results obtained show that the real-time energy detector can detect signals with low Signal-to-Noise Ratios (SNRs) down to -8 dB at desired values of probability of detection ($P_d \geq 0.9$), and probability of false

alarm $\left(P_{fa} \leq 0.1 \right)$ at a sample size of 2048. This satisfies the IEEE 802.22 WRAN standard for TV band sensing.

An energy spectrum sensing SDR testbed is successfully built and tested as proof of concept. This work leaves a testbed for other reaserchers in the COMMED Research Group that can be used for wireless communication experimentation. In addition, the USRP configuration and Out-of-Tree (OOT) module implementation method using GNU Radio Companion (GRC) is clearly documented.

Acknowledgements

To God, who saw the future before I could even fathom it. Who made me in His image and likeness, His mind becoming my mind. For the grace He so lavished during the course of this work, I am forever grateful.

My sincere gratitude goes to my supervisor Professor Mqhele E. Dlodlo, for the unwavering support, encouragement and patience that saw me through the completion of my Masters degree. Thank you for believing in me.

Many thanks to Michael Dickens from Ettus Research. Thank you for the technical support and encouragement in the development of the testbed. My heartfelt gratitude goes to Dr Sesham Srinu of the University of Cape Town. Thank you for your time and assistance in the conceptualization of the work. To the COMMED Research Group of the University of Cape Town, I am sincerely grateful for each one of you. You all played a significant role in the completion of this work. I am sincerely grateful. Special thanks goes to Frederick Kumi, Lerato Mahopi, Henry Ohize and Nixon Thuo Ng'ethe, thank you for being dependable.

To Valerie Chiriseri, thank you for being God's outstretched arm. To my friends who became family Bashangwanye Magwape, Lusanda Gwayi and Theophilus Dzingai. Thank you for all the support. To mum, dad and my siblings, I would not have asked to be born in another family! Thank you a million for your relentless love and support.

Lastly, special thanks goes to my sponsors the National Research Fund (NRF). Thank you for the investment, the future of Africa is indeed bright!

Table of Contents

List of Figures

List of Tables

Abbreviations

ADC	Analogue to Digital Convertor
AM	Amplitude Modulation
API	Application Program Interface
AWGN	Additive White Gaussian Noise
BPF	Band Pass Filter
COTS	Commercial off-the-shelf
CR	Cognitive Radio
CSD	Cyclic Spectral Density
DAC	Digital to Analogue Convertor
DAB	Digital Audio Broadcasting
DDC	Digital Down Convertor
DSP	Digital Signal Processing
DVB-T	Digital Video Broadcasting for terrestrial television
DUC	Digital Up Convertor
FCC	Federal Communication Commission
FDM	Frequency Division Multiplexing
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDTV	High-Definition Television
HIPERLAN-II	High Performance Local Area Network type 2
GbitE	Gigabit Ethernet
GI	Guard Interval

GNU	GNU’s Not Unix
GPP	General Purpose Processor
GUI	Graphic User Interface
GRC	GNU Radio Companion
GSM	Global System for Mobile
IEEE	Institute of Electrical and Electronics Engineers
IDE	Integrated Development Environment
IF	Intermediate Frequency
ISM	Industrial, Scientific and Medical
LTE	Long Term Evolution
MNF	Maximum Normal Fit
MSPS	Mega Samples Per Second
OFDM	Orthogonal Frequency Division Multiplexing
OOT	Out-of-Tree Module
PC	Personal Computer
PCA	Principal Component Analysis
PN	Pseudo Noise
PU	Primary User or Licensed User
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
ROC	Receiver Operating Characteristics
SBX	S-Band Transponder
SDR	Software-Defined Radio
SNR	Signal-to-Noise Ratio

SU	Secondary User or Unlicensed User
SWIG	Simplified Wrapper and Interface Generator
TCP	Transmission Control Protocol
UHD	USRP Hardware Driver
USRP	Universal Software Radio Peripheral
USRP2	Version 2 – Universal Software Radio Peripheral
Wi-Fi	Wireless Fidelity
WiMax	Worldwide interoperability for Microwave access
WSS	Wide-Sense Stationary
WLANs	Wireless Local Area Networks
XML	Extensible Markup Language

Notations

H_0	Null hypothesis
H_1	Alternative Hypothesis
η	Noise-to-Noise Ratio
ψ	Signal-to-Noise Ratio
L	Bin Size
N	Sample Size
$x(t)$	Received Analogue Signal
$\tau(n)$	Energy Detector Test Statistic
$e(x)$	Entropy Test Statistic
$\phi(m)$	Matched Filter Detector Test Statistic
λ	Generic Threshold
λ_ϵ	Energy Detector Threshold
λ_e	Entropy Detector Threshold
λ_m	Matched Filter Detector Threshold
$s(f, \propto)$	Cyclic Spectral Density
$R_y^\propto(\tau)$	Cyclic Autocorrelation Function

Chapter 1

1 Introduction

The future of wireless systems is highly dependent on the radio spectrum [1], [2]. Reports published by the Federal Communications Commission (FCC) of the United States of America (USA) indicate the inefficiency of the current fixed spectrum allocation [3]–[5]. In fixed spectrum allocation, a portion of the spectrum is allocated to an incumbent who only uses it at selective times, frequencies or space. This implies that some portions of the spectrum are not always fully utilized. Figure 1.1 illustrates this phenomenon and shows how spectrum utilization is characterized by some portions that are heavily used, while others lie dormant [6].

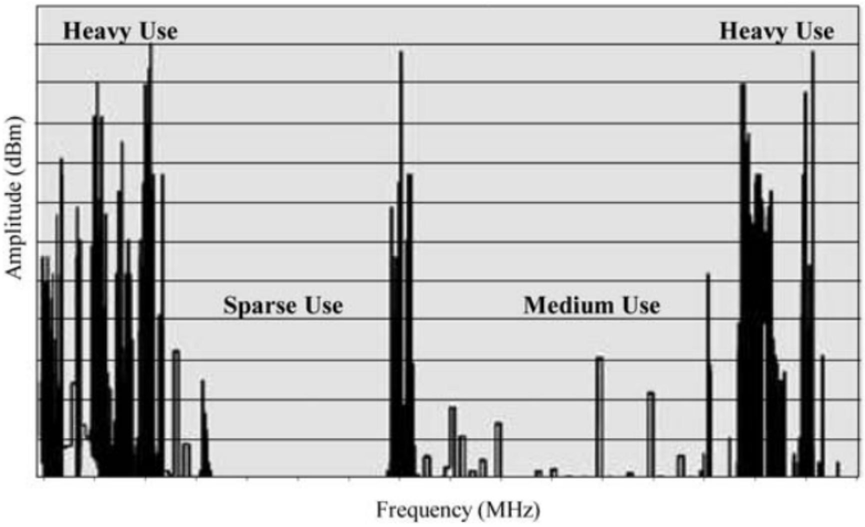


Figure 1.1: Radio Spectrum Utilization [6]

Statistics have shown that in some bands, the maximum spectrum occupancy ranges between 5.5% and 13.1% for the allocated communication applications, as shown in Figure 1.2 [7], [8]. Some of these applications include smart grid, public safety, military, surveillance radar and medical applications [9]. Their demand for wireless connectivity has been increasing; leading to the potential for spectrum scarcity [7]. However, the major radio spectrum challenge has been shown to be poor spectrum utilization, and not spectrum shortage [7]–[10].

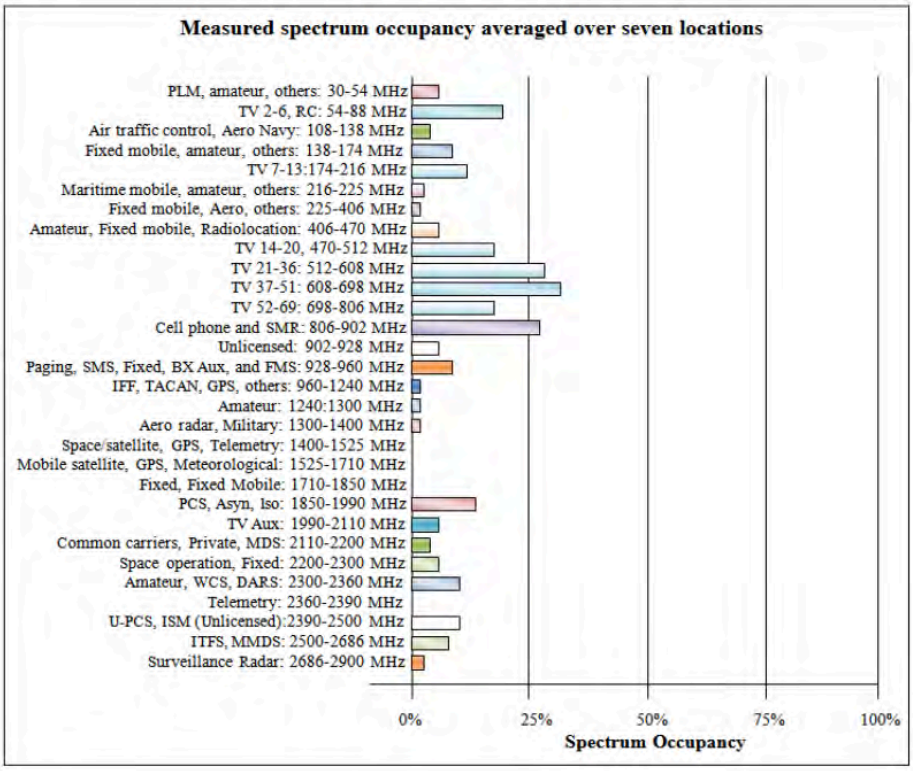


Figure 1.2: Radio Spectrum utilization profile [11]

In 1999, Mitola and Maguire proposed cognitive radio technology as a solution to inefficient spectrum utilization [12]. In essence, a cognitive radio is an intelligent Software-Defined Radio (SDR) with the ability to sense and understand its surroundings; enabling it to dynamically adjust its transmission parameters and access vacant bands in the spectrum [9]. To ensure spectrum optimization, efficient spectrum sensing algorithms have since been developed [13]–[15]. The cognitive radio technology for spectrum sensing, can be realized by GNU Radio (an open source software) and the Universal Software Radio Peripheral (USRP) developed by Ettus Research [16].

1.1 Motivation

Spectrum sensing, management, sharing and mobility are the main functions of a cognitive radio [8], [10]. Spectrum sensing is crucial, as it determines spectrum availability in the presence of licensed users [17]–[19]. In spectrum sensing, cognitive radios have the ability to scan the spectrum, sense when the primary user is not using a particular band, and opportunistically allow a secondary (unlicensed) user to occupy the vacant bands (white space); bringing about efficient spectrum utilization. Hence, most of the cognitive radio research work to date has been on spectrum sensing [12]–[14], [19]–[22]. This primarily consists of simulations, which alone are not adequate to prove the validity of the algorithms, as is the case with real-time implementation. Real-time implementation enables analysis of system performance and gives a realistic picture of the effectiveness of the spectrum sensing algorithms.

1.2 Problem Statement

Seeing the importance of the spectrum sensing function, researchers can ill-afford overlooking the low-level details required for real-time sensing system design. Furthermore, if the simulations have a high abstraction level, many assumptions made in the simulated environment may not be feasible in real-time scenarios, as this results in a disconnect between the predicted result and empirical results. Hence, there is a need to explore practical implementation of spectrum sensing algorithms on a real-time testbed, to determine correlation between simulation and implementation results.

1.3 Research Questions

This research rests on the following questions:

1. What are the limitations of the simulated energy detector?
2. How can the energy detection technique be implemented using the SDR?
3. How does the sensing performance of an implemented real-time implemented energy detector compare to its simulated equivalent?

1.4 Research Objectives

The main purpose of this research is to build a real-time spectrum sensing system that can improve radio spectrum utilization. This is broken down into the following specific objectives:

1. To simulate the spectrum sensor using MATLAB.
2. To create energy detector-based spectrum sensing modules in GNU Radio Companion.
3. To compare the performance of the real-time energy sensor, implemented on a USRP SDR platform using GNU Radio Companion to its simulation results.

1.5 Project Scope

This research focuses on the real-time implementation of a single node cognitive radio on the USRP SDR platform. The energy detector non-cooperative spectrum sensing technique is chosen for implementation. An analytical study of the performance of the real-time energy detector is completed. Results obtained for the real-time implementation are then benchmarked to those obtained by simulations, and the resulting conclusions are presented.

1.6 Publications

McBath J. Rwodzi, Sesham Srinu, Mqhele E. Dlodlo “Spectrum Sensing using USRP Software-Defined Radio Platform”, Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2015, pp. 1-2.

1.7 Thesis Outline

The rest of the work contains six primary chapters, as outlined below:

Chapter 2 describes the Cognitive Radios (CR) and the following spectrum sensing methods: energy detection, entropy detection, cyclostationary detection, and matched-filtering detection. The spectrum sensing techniques are analysed and their advantages and disadvantages compared. Reasons are then given as to why the energy detector was chosen for implementation in this research.

Chapter 3 presents an overview of the hardware platform along with the associated software used in this research: Universal Software Radio Peripheral and the GNU Radio Software. Spectrum-sensing related work is highlighted at the end of the chapter.

Chapter 4 presents the simulation setup and parameters considered for the energy detector. The results of the simulation are then given and the performance evaluated.

Chapter 5 describes the configuration of the Software-Defined Radio and the implementation of the energy detector on the GNU Radio platform. A detailed outline of GNU Radio Out-of-Tree (OOT) module construction is given.

Chapter 6 presents the results obtained from the real-time implementation of the energy based spectrum sensing technique on the USRP and GNU Radio platform. The performance of the real-time implementation is evaluated with respect to simulations and conclusions drawn.

Chapter 7 summarises the research work and provides recommendations for future work.

Chapter 2

2 Cognitive Radios and Spectrum Sensing

In this chapter the background to cognitive radio technology is given. Its functionality is discussed with special attention given to spectrum sensing. The thresholding selection techniques used in spectrum sensing are then analysed. This is followed by an overview of Orthogonal Frequency Division Multiplexing (OFDM) the selected multiplexing technique for this research. Lastly the Signal-to-Noise Ratio estimation methods are then presented.

2.1 Background

According to the Federal Communications Commission (FCC) a cognitive radio is “a radio or system that senses its operational electromagnetic environment and can dynamically and autonomously adjust its radio operating parameters to modify system operations such as maximize throughput, mitigate interference, facilitate interoperability, and access secondary markets”[10]. Thus, cognitive radios have the ability to sense the radio spectrum and adjust accordingly. Sensing algorithms can be developed to facilitate spectrum sensing and enable wireless applications to locate bands with adequate resources. Primary Users (PUs) and Secondary Users (SUs) are the two categories of spectrum user classification [8], [10], [17]. Primary users are the licensed users and have higher priority to use the spectrum. Secondary users, also known as cognitive radios, are unlicensed users. They have lower priority and they opportunistically access the spectrum in the absence of licensed users. The functions of cognitive radios involve spectrum sensing, spectrum management, spectrum sharing and spectrum mobility [8], [12], [23].

2.2 Cognitive Radio Functions

The cognitive radio cycle in Figure 2.1, shows the processes and how they relate within the radio environment [24]. Spectrum sensing is the first step in the cycle. It involves spectrum band monitoring and capturing of the electromagnetic radio transmissions for the purpose of identifying spectrum holes [8]. The cognitive radio scans the spectrum, taking into account some or all of the following spectrum dimensions: frequency, time, space, code, polarization and angle of arrival [10], [25]. Frequency, time and space are the most commonly used spectrum dimensions [12], [26]. Spectrum sensing can be viewed from two perspectives, out-band and in-band [24]. Out-band sensing involves the secondary user sensing the radio spectrum for vacant bands in order to occupy them. With in-band sensing, the secondary user is already occupying the band and on the lookout for incoming primary users [23], [24].

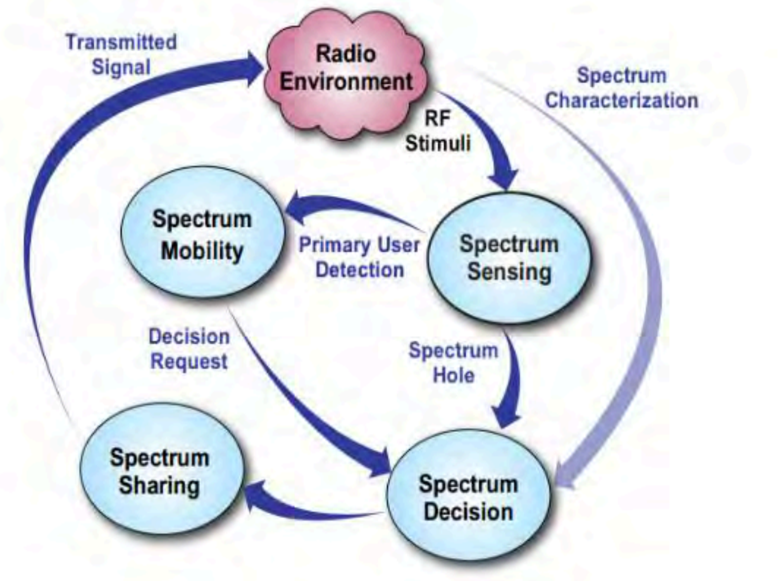


Figure 2.1: Cognitive Radio Cycle [23]

After vacant band identification, spectrum decision follows. This process ensures that the secondary user obtains the frequency band that match its Quality of Service (QoS) requirements [8]. The spectrum characteristics considered include the time analytics as well as frequency band resource features such as interference level, channel error and carrier frequency [27]; whilst the cognitive radio parameters prioritized are the bandwidth, data rate, acceptable error rate and the delay bound [24]. Spectrum decision rules are used to select the suitable bands and then the cognitive radio is configured to suit the selected band [28].

The cognitive radio occupying a spectrum band has lower priority than an incoming primary user. Once the primary user is detected, the cognitive radio reconfigures and seamlessly moves to the next best vacant spectrum band. This process is known as spectrum mobility. Spectrum mobility ensures communication performance at the CR is maintained [6]. In the event that the current occupied band’s characteristics do not meet the SU QoS requirements, handoff occurs. Handoff may also be triggered when the PU comes into the band occupied by the SU. Glitches during the handoff are avoided by spectrum decision that ensures that possible vacant bands are listed.

Spectrum sharing enables cognitive radio users to select channels and allocate power with the aid of a broker. The broker is connected to the cognitive radios via a base station and maintains the desired quality of service; this is known as inter-network spectrum sharing. When spectrum sharing occurs within a network, it is known as intra-network spectrum sharing [8].

2.3 Spectrum Sensing

Spectrum bands can be categorized into three types [12], [29], [30]: black spaces, gray spaces and white spaces. Black spaces are heavily used portions of the spectrum by high power local interferers [30]. Gray spaces are lightly used and are occupied by low power interferers [30], whereas, white spaces have no interference, except ambient noise [30]. Cognitive radios enable secondary users to occupy the gray spaces and white spaces; these are also known as spectrum holes. According to Akyildiz et al. [8] a spectrum hole can be defined as “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 2.2 illustrates the spectrum hole concept [23].

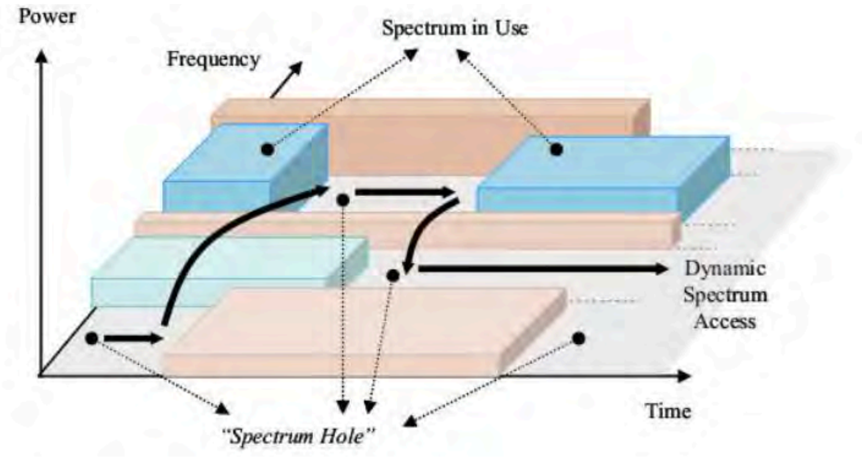


Figure 2.2: Spectrum hole [23]

The main objective of spectrum sensing is avoiding interference between the PU and SU during transmission. However, this can be rectified by altering the modulation type, power and frequency. Spectrum sensing in the presence of noise remains a major area of research [19], [25], [31].

In practice, active and passive methods can be used for spectrum awareness. Active methods involve spectrum detection and estimation, whereas, passive methods involve the use of database registry and beacon signals to obtain information of the primary user in the spectrum. Passive methods require prior knowledge about the primary user to carry out their operations. This is unlike active methods, hence making active methods more popular. An example of an active method is the use of the cognitive radio technology.

Secondary users should be able to identify spectrum holes using different sensing algorithms to maintain QoS. Detection techniques in cognitive radio are therefore critical to the performance of wireless systems. Detection quality is determined based on two primary criteria [12]: probability of detection and probability of false alarm. Probability of detection "denotes the probability of a cognitive radio user declaring that a primary user is present when the spectrum is indeed occupied by a primary user" [6]. Probability of false alarm "denotes the probability of a cognitive radio user declaring that a primary user is present when the spectrum is free" [6]. A false alarm therefore signifies the reduction of the CR throughput (missing the spectrum holes).

The spectrum sensing problem can be represented as a binary hypothesis, mathematically given by [12]:

$$\begin{aligned} H_0 &: Y[n] = W[n] && \text{if PU is absent} \\ H_1 &: Y[n] = W[n] + hS[n] && \text{if PU is present} \end{aligned} \tag{2.1}$$

where Y denotes the received signal observation for secondary user, S the signal from primary user, W the Additive White Gaussian Noise (AWGN) and h is the channel gain.

To evaluate the performance, a decision threshold (λ) is set and compared to the test statistic (τ) generated. The detection rule is determined as:

$$\begin{aligned} \tau &> \lambda && \text{primary user present,} \\ \tau &< \lambda && \text{primary user absent.} \end{aligned} \tag{2.2}$$

The performance of the spectrum sensing techniques can also be affected by multipath and fading. Consequently, in literature, several detection techniques have been investigated to improve the sensing performance. There are three basic categories for spectrum sensing, namely: transmitter-based detection, interference-based detection and receiver-based detection. These are illustrated in Figure 2.3 [24]. In the receiver-based detection method, wireless sensors placed in proximity of the PU receivers are used to measure the power leakage released by their local oscillators [23]. Whereas, interference-based detection is based on the assumption that if two signals can interfere with each other, it means they are in the same communication proximity, hence detection can be based on their interference [29]. In the transmitter-based detection, weak signals from a primary transmitter are detected by local observations of the cognitive radio users [30]. The transmitter-based spectrum sensing detection method is used in this research.

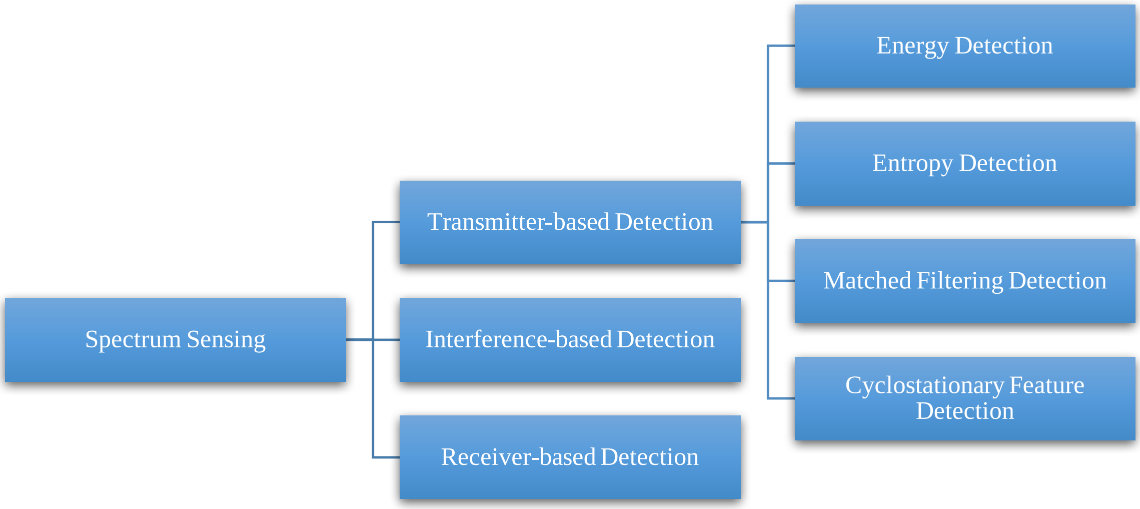

Spectrum Sensing
Transmitter-based Detection
Interference-based Detection
Receiver-based Detection
Energy Detection
Entropy Detection
Matched Filtering Detection
Cyclostationary Feature Detection

Figure 2.3: Single Node Spectrum Sensing Techniques [24]

Sensing the radio frequency environment involves the cognitive radio taking samples, and then performing digital signal processing operations that produce a test statistic. The test statistic is then compared to a predetermined threshold to determine whether the primary user is present or absent. Numerous detection techniques have been developed for use in the spectrum sensing seen in Figure 2.3. All the detection methods can be classified as either coherent or non-coherent detection techniques [6], [16]. In coherent signal detection, a priori knowledge of the primary user is required for detection by the secondary user, whereas, such a priori knowledge in non-coherent detection, is not required to determine the presence or absence of a signal. For instance, energy and entropy detection techniques are non-coherent, whilst cyclostationary and matched-filter detection techniques are coherent.

2.3.1 Energy Detection

This is the most popular detection technique and has “the least computational and implementation complexity” [32]. A priori knowledge of the primary user signal is not required for signal detection [21], [28], [31]. Calculating the received signal’s energy gives the test statistic, which is compared to a predetermined threshold. The threshold is determined from noise energy and its accuracy is key to the performance of the energy detector. If the received signal's energy at the cognitive radio is greater than the set threshold, the alternate hypothesis H_1 is validated and the primary user is concluded to be present. If the energy is lower, the null hypothesis H_0 is validated, thus signifying the presence of a spectrum hole. The binary hypothesis is presented in Equation 2.1.

Figure 2.4 shows the stages in the time domain energy detector. The signal being detected $y(t)$ is defined within a specific Bandwidth (B), with centre frequency (fc) by the cognitive radio. Once the signal is received via the Radio Frequency (RF) antennas, it is filtered by the Band Pass Filters (BPF) for channel selection [26]. Consequently, the analogue signal is sampled to obtain discrete signals using the Analog to Digital Convertors (ADCs). The digital output is squared and the average of N samples calculated to give the energy test statistic which is compared to a decision threshold λ_{ϵ}. A binary hypothesis H_0 and H_1 is established and used to determine the presence or absence of the primary user signal.

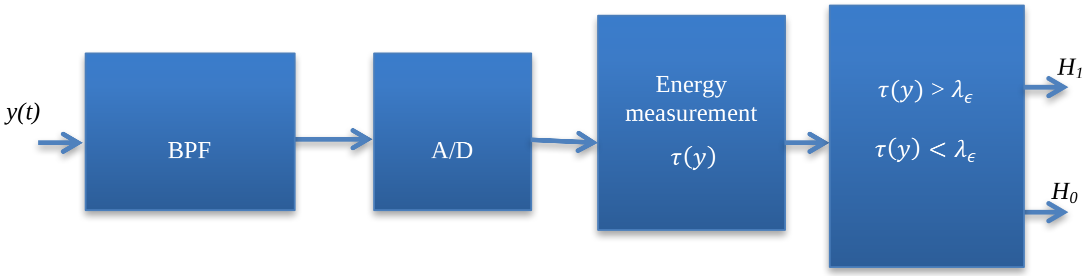

y(t)
BPF
A/D
Energy measurement
$\tau(y)$
$\tau(y) > \lambda_{\epsilon}$
$\tau(y) < \lambda_{\epsilon}$
H_1
H_0

Figure 2.4: Energy detector block diagram [12]

Noise received by the sensor is considered as Additive White Gaussian Noise (AWGN). The test statistic for the energy detector can be formulated as [31]:

$$\tau(n) = \sum_{n=0}^{N-1} (Y[n])^2, \tag{2.3}$$

where $\tau(n)$ is the energy test statistic, N is the sample size, and $Y[n]$ is the received signal sequence.

The decision rule, as shown in Equation 2.2, is applied to the test statistic. In energy detection, the determined threshold λ_ϵ is assumed to be a Gaussian random process with zero mean and variance following a chi-squared central distribution [34]. The approach possesses many limitations for narrowband signals due to its inflexibility. The periodogram method is therefore used which is defined by the square of the modulus of the Fast Fourier Transform (FFT), to estimate the spectrum in real-time. With the periodogram method, wider bandwidths can be accessed and many signals detected simultaneously. Signal detection can be improved by adjusting the frequency resolution and FFT size. Increasing the FFT size also increases sensing time, and this is analogous to changing the analogue pre-filter. Increasing the sample sizes increases the accuracy of the signal detection. The choice of sample size and frequency used is therefore determined in order to achieve low latency, minimum possible complexity and desired resolution.

The performance of the energy detector is determined by testing the Neyman-Pearson hypothesis [12]. A comparison is made between the log-likelihood ratio and the decision threshold,

$$log\left(\frac{P\left(y_{0,}y_1, \dots\dots\dots\dots . y_{(N-1)}\ |H_1\right)}{P\left(y_0, y_1, \dots\dots\dots . . y_{(N-1)}\right)|H_0}\right) \lessgtr_{H_0}^{H_1} \lambda_\epsilon, \tag{2.4}$$

where $P(y|H_1)$ and $P(y|H_0)$ represent the probability density functions of the alternative hypothesis H_1 and null hypothesis H_0 respectively, and λ_ϵ is the decision threshold.

Absence of a signal ($\tau(n) < \lambda_\epsilon$) gives a central chi-squared distribution with N degrees of freedom, whilst the presence of a signal ($\tau(n) > \lambda_\epsilon$) gives a non-chi-squared distribution with N degrees of freedom. A large sample size (N) is thus favourable for energy detection. Mathematically, the probability of detection and probability of false alarm can be evaluated as [29], [34]:

$$P_d = Q\left(\frac{\lambda_\epsilon - N(\sigma_w^2 + \sigma_x^2)}{\sqrt{2N(\sigma_w^2 + \sigma_x^2)^2}}\right), \tag{2.5}$$

$$P_{fa} = Q\left(\frac{\lambda_\epsilon - N\sigma_w^2}{\sqrt{2N\sigma_w^4}}\right), \tag{2.6}$$

where Q is the generalized Marcum Q – function, which is expressed as [12]:

$$Q(x) = \frac{1}{2}\left(\frac{2}{\pi}\int_{\frac{x}{\sqrt{2}}}^{\infty} \exp(-t^2)\, dt\right). \tag{2.7}$$

The threshold λ_ϵ can be represented in terms of the P_{fa} as shown in Equation 2.8 [26].

$$\lambda_\epsilon = F^{-1}(1 - P_{fa}|U, \sigma_w^2), \tag{2.8}$$

where F^{-1} and P_{fa} are the inverse gamma function and desired probability of false alarm respectively. U is the shaping parameter, which is highly dependent on the sample size (N), and σ_w^2 the sizing parameter determined by the noise variance.

2.3.2 Entropy Detection

Figure 2.5 shows the block diagram of the entropy detection process. The signal received *y(t)* is passed through the BPF and then forwarded to the ADC. Consequently, the N-FFT output is processed to produce the entropy test statistic *e(y)*, which is compared to a set decision threshold λ_e.

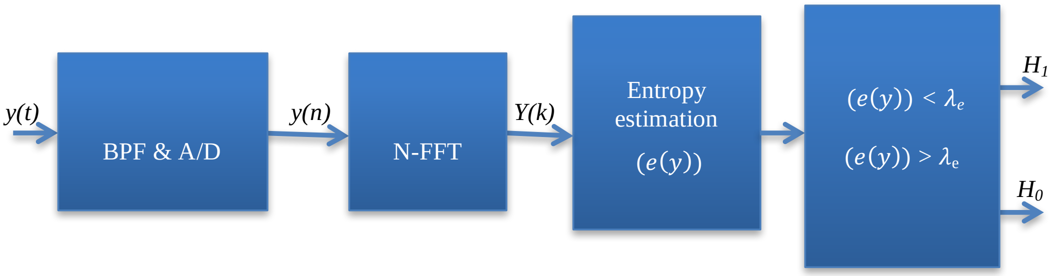


Figure 2.5: Entropy detection block diagram [12]

Entropy estimation is done using the histogram-based approximation of the received signal. The magnitude of the FFT converts the received signal in the frequency domain, forming a histogram of L bins of equal width. The bin size expression is given by [12]:

$$Bin\ size, \Delta_{fd} = \frac{(Y_{max} - Y_{min})}{L}, \tag{2.9}$$

where Y_{min} and Y_{max} are the minimum and maximum values of $Y(k)$ respectively. L satisfies the design parameter such that $\frac{N}{L} \geq 4$; where N and L are the number of received samples and number of bins respectively.

The entropy estimate is therefore given by the expression [12]:

$$e(x) = \sum_{i=1}^{L} -\left(\frac{f_i}{N}\right) log_2 \left(\frac{f_i}{N}\right), \tag{2.10}$$

where f_i represents the number of samples of Y(k) in the i^{th} bin and $p_i = \frac{f_i}{N}$ denotes the number of samples in the i^{th} bin.

This implies that to observe the presence or absence of the primary user, the following binary hypothesis holds:

$$e(x) \leq \lambda_e,\ H_1 \text{ primary user present}$$

$$e(x) > \lambda_e,\ H_0 \text{ primary user absent} \tag{2.11}$$

The decision threshold λ_e is expressed as [35]:

$$\lambda_e = H_L + Q^{-1}(1 - P_{fa})\sigma_w, \tag{2.12}$$

where H_L, σ_w and Q, are the theoretical Gaussian entropy, noise standard deviation and Gaussian distribution function respectively.

2.3.3 Cyclostationary Detection

Cyclostationary signals have periodic characteristics and spectral correlation that make them susceptible to noise and interference [14], [19]. The presence of primary users is hence determined by the periodicity of the received signals [21], [36]. The periodicity property is analysed from the sinusoidal carriers, pulse trains and spreading code using the spectral correlation function [21], [36]. These characteristics are used to detect the presence of a primary user in the scanned band, with their absence signifying a vacant band. An overview of the processes that take place during cyclostationary detection is shown Figure 2.6.

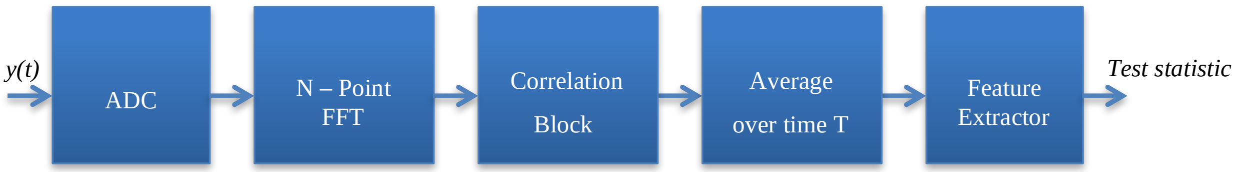


Figure 2.6: Cyclostationary Detection [24]

A binary hypothesis is established for primary user identification based on the received signal's *y(t)* periodicity properties. Knowledge of the signal to be detected is required as it is a coherent detection method. One of the major advantages of cyclostationary algorithms is that they can separate primary user signals from noise signals [19]. This is because noise is Wide-Sense Stationary (WSS) and has no correlation with the primary user signal [21]. Consequently, primary user modulated signals are cyclostationary and possess spectral correlation [21]. Cyclostationary-based sensing makes use of the cyclic correlation function, also known as Cyclic Spectral Density (CSD), instead of a Power Spectral Density (PSD) for signal detection [10].

The cyclostationary property of the received signal *y(t)* is determined based on mean and autocorrelation function given by Equations 2.13 and 2.14 [21], [24].

$$E_x(t) = \mu(t + mT_0), \tag{2.13}$$

$$R_x(t,\tau) = R_x(t + mT_0, \tau + mT_0), \tag{2.14}$$

where T_0 is the time period, τ is the autocorrelation function lag, t is the time index, and m is the integer.

Considering the Fourier series, the CSD function is given by the expression [12], [21], [24]:

$$S(f,\alpha) = \sum_{\tau=-\infty}^{\infty} R_y^{\alpha}(\tau) e^{-j2\pi f\tau}, \tag{2.15}$$

where $R_y^{\alpha}(\tau)$ is the Cyclic Autocorrelation Function (CAF) given by:

$$R_y^{\alpha}(\tau) = E\left[y(n+\tau)y^*(n-\tau)e^{-j2\pi\alpha n}\right], \tag{2.16}$$

where α is the cyclic frequency.

2.3.4 Matched Filtering Detection

This is a coherent sensing technique, and requires accurate knowledge of the transmitted signal by the primary user to optimally detect the signal at the cognitive user. Such information includes the bandwidth, operating frequency, modulation and pulse shaping. The information is used for demodulating the received signal [15]. The a priori knowledge at the cognitive user increases the sensor's detection speed and accuracy. Matched filtering is therefore considered as the most efficient method for primary user detection as it takes the least time to determine the probability of false alarm [10]. Figure 2.8 shows the block diagram of the matched filtering detector.

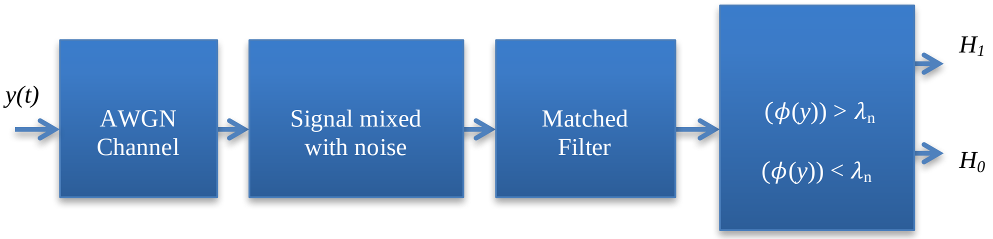


Figure 2.7: Matched Filtering Detection [12]

Matched filtering detectors are difficult to implement because of the cumbersome algorithms, which also draw large quantities of power during computation. This is a major setback to their use [10]. The test statistic $\phi(y)$ for a matched filtering detector is determined by the inner product principle as expressed in Equation 2.17 [24]:

$$\phi(y) = \sum_{n=0}^{N-1} y(n)s(n), \tag{2.17}$$

where $y(n)$ and $s(n)$ represent the received signal and correlation coefficient respectively.

P_d and P_{fa} in accordance to the Neyman-Pearson hypothesis are expressed as [15]:

$$P_d = Q\left(\frac{\lambda_m - E}{\sqrt{E\sigma_W^2}}\right), \tag{2.18}$$

$$P_{fa} = Q\left(\frac{\lambda_m}{E\sigma_W^2}\right), \tag{2.19}$$

where Q and E are the Q-function, received signal energy respectively. λ_m is the threshold expressed as [15]:

$$\lambda_m = Q^{-1}(P_{fa})\sqrt{E\sigma_W^2.} \tag{2.20}$$

The decision is determined by the binary hypothesis and is represented by:

$$\phi(m) > \lambda_{\mathrm{m}},\ H_1 \text{ primary user present}$$
$$\phi(m) < \lambda_{\mathrm{m}},\ H_0 \text{ primary user absent} \tag{2.21}$$

2.4 Spectrum Sensing Techniques Analysis

The energy detection technique has many drawbacks discussed as follows. Increasing the noise power reduces the energy detector performance and this makes the received signal from primary users and CR users indistinguishable. Noise in energy detection techniques is assumed to be AWGN, with mean zero and known variance [37]. The noise emanates from various sources: thermal noise, circuits and interference caused by other signals. Noise variance may therefore vary beyond what is estimated due to changes in temperature, interference and filtering [38], which has an effect on the threshold. The energy detection technique is also prone to baseband filter effects and spurious tones [10]. Data collection time, required to maintain reliability, may be lengthy. Threshold selection is the major challenge when using the energy detector, therefore the right procedures have to be followed during implementation. Despite the disadvantages the energy detector poses, its simplicity and low computational complexity makes it suitable for use in this research.

The matched filtering detection technique quickly determines probability of detection and probability of false alarm. The availability of a priori knowledge enables the quick identification of the PU signal, as well as averaging of the noise signal. This makes it possible to accurately detect signals with low Signal-to-Noise Ratio (SNR) [39]. The required knowledge includes the bandwidth, modulation type, order, operating frequency, pulse shaping and frame format [24]. The primary and secondary user systems do not actively communicate and thus this information may not be obtainable. The received primary user signal also requires demodulation before being aligned with the reference signal. Receiving different signal types would mean a different receiver for each, which increases the implementation complexity. Moreover, this increases the sensing time and requires a large power expenditure [10]. Hence, the matched filtering technique was not considered for implementation in this work.

The major advantage the cyclostationary feature detector has over the energy detector is its ability to separate noise signals from PU signals using the periodicity property in low SNRs [40]. This technique is ideal for low SNRs as it can combat interference. The cyclostationary detector functions with a large FFT size, which increases the power requirement for computation and is not cost effective. When noise is stationary, energy detectors perform better than cyclostationary detectors. However as noise increases due to co-channel interference, the performance of the energy detector degrades, whilst that of the cyclostationary detector improves [10].

The entropy detector detects signals with very low SNR with high accuracy, although a priori knowledge of the PU is required. The algorithms required for this detector are therefore more complex. This detection technique was not chosen for implementation because of this high computational complexity. Table 2.1 gives a comparative analysis of the spectrum sensing techniques presented in this work.

Table 2.1: Spectrum sensing technique comparison [15], [41]

Metric	**Energy Detector**	**Entropy**	**Cyclostationary**	**Matched Filter**
Detection Accuracy	+ Good performance at high SNRs - Poor performance at low SNRs - High noise floor could be a false detection	+ Good performance at all SNRs	+ The best performance at all SNRs	+ Very good performance at all SNRs (if RX has sufficient knowledge of TX) - Poor performance if little is known about TX
Complexity	+ Low computation/implementation complexity	- The moderate complex	- Most complex	- High complexity (requires a dedicated RX for each primary signal class)
Robustness	+ Does not require any prior information of TX signal - Requires knowledge of noise power - Does not work for spread spectrum signals	+ Does not require any TX information at RX	- RX must know TX signal fundamental frequency	- Requires near perfect TX information at RX
Design Choices	- Difficult to choose decision threshold	Selecting optimum L value is difficult	+ Cyclostationarity can be intentionally induced to improve accuracy	+ TX characteristics can be chosen to improve accuracy

2.5 Threshold Selection Technique

The basic theory of the energy detector states that the energy of a defined band is measured to determine the test statistic [42]. As discussed, the test statistic is then compared to a set threshold. When the value of the test statistic is greater than the threshold, a primary user is said to be occupying the sensed band. When the test statistic is less than the set threshold value, the spectrum band is considered vacant. Hence, threshold accuracy has a critical role in determining the occupancy of the sensed spectrum. There are two basic categories of techniques that are used to set the threshold: adaptive threshold and fixed threshold [13], [24]. These are discussed in section 2.5.1 and 2.5.2 respectively.

2.5.1 Adaptive Threshold Technique

The adaptive threshold technique provides a threshold that adjusts to the noise variation in the environment over the time interval in which the spectrum is monitored. A priori knowledge of the primary user and noise floor level is not required. This technique enhances the performance of the energy detector as the variable threshold lowers false alarms and misdetections [24]. The received signal standard deviation and mean statistical properties are used to determine the threshold. The standard deviation of a point on signal determines the variation rate of the threshold. Increasing deviation implies that the threshold variation also increases and vice-versa. Some of the adaptive threshold techniques that have been developed are: Otsu’s algorithm, Maximum Normal Fit (MNF), Principal Component Analysis (PCA) and the Recursive One-Sided Hypothesis (ROHT) [24]

Originally applied in image processing, Otsu’s algorithm is now also used in spectrum sensing. In image processing, an assumption is made that an image has two types of pixels that form a bi-modal histogram. These two classes are the foreground pixels and background pixels, which represent the signal and noise component respectively. For spectrum sensing, the threshold is determined by distinguishing the signal from the noise components and reducing the intra-class variance [43]–[45].

The MNF technique is based on the assumption that the noise component has the lowest samples in the received signal. In addition, the noise component is assumed to follow a Gaussian distribution [7], whereas the signal component distribution does not necessarily have to be defined. Using the best-fit approach, the noise threshold can be determined from the two peaks formed from the noise and signal components [7], [24].

PCA is a mathematical method that uses eigenvalue analysis. This entails variables with mutual relation being converted into uncorrelated variables; these are also referred to as principal components. Each principal component plays a significant role in determining the variability in the data. The eigenvalues obtained produce a covariant matrix that is used to separate the signal from the noise. Large covariance is assumed to denote signal samples and low covariance the noise samples. The threshold is estimated from the midpoint of these two signals [46]. The main objective in this technique is to minimize data dimensionality and the recognition of implicit variables [24].

The ROHT is a mathematical technique where the signal components are separated from the noise components, and is also referred to as the one-sided hypothesis testing [47]. It is assumed that a measured sample has more noise components compared to signal components, and the noise components are assumed to follow a normal Gaussian distribution. Another assumption is that the sample is large enough, such that the mean and standard deviation obtained are assumed the same for the portion of spectrum being sensed. The algorithm performs iterations and the Gaussian curve is used to remove components to its right-hand side. The threshold at the i^{th} iteration is determined from the p-value, mean and standard deviation [24].

2.5.2 Fixed Threshold Technique

Unlike the adaptive threshold detection technique, the fixed threshold technique is set based on a fixed noise reading above the noise floor. Hence, knowledge of the noise floor level is required for the threshold to be set. Some fixed threshold techniques that have been developed are as follows: histogram analysis, cumulative density function analysis, empirical analysis, receiver noise characteristic thresholding, and the p-tile based technique [24], [48].

In the histogram analysis or mode method, the signal and noise bimodal histogram plots are drawn based on frequency of the measured statistics. In a densely populated band, the threshold is derived from the centre point of the two plots [48]. When the band is thinly populated and characterized by noise with no incumbent, a bimodal histogram may not be possible. Thus applying the Laplacian operator on the measured spectrum samples is used to estimate the slopes. The Laplacian operator defines the rate at which the slope changes for *U(x,y)* and can be expressed mathematically as [48]:

$$H(x, y) = -\nabla^2 \{ U(x, y) \}, \tag{2.22}$$

$$\text{where} \quad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}.$$

When the signal has constant noise around it leading to ramps at the edges, the areas of constant slope have a Laplacian magnitude of zero, whereas on the points of transition the Laplacian magnitude is high. Plotting the points that coincide with the high Laplacian magnitude gives a bimodal histogram, which is used to determine the threshold from the centre point of the plots [48]. The corresponding Laplacian values can be deduced from the threshold *L(n)*, expressed as [48]:

$$L(n) = \mu_L + (n\sigma_L), \tag{2.23}$$

where μ_L represents the mean, σ_L represents the Laplacian standard deviation and *n* is a defined integer.

In the cumulative density function analysis method, an unoccupied radio spectrum band is measured and the data used to evaluate the Inverse Cumulative Density Function (ICDF). The ICDF gives a plot of probability of false alarm versus the threshold. Hence the threshold can be determined for a specific P_{fa} [48]. For a noise power random variable G, the ICDF can be represented as:

$$ICDF(G, T) = P(S \geq T), \tag{2.24}$$

where T is the threshold and $P(S \geq T)$ is the correlative probability of false alarm [48].

When the radio spectrum data is captured from an occupied band, the sample can be denoted by R and Z representing the random variable power. The cumulative density function can be given by:

$$CDF(Z,R) = P\left(Z \leq R(f_i,t_j)\right), \; R(f_i,t_j) \in R, \tag{2.25}$$

where $R(f_i,t_j)$ is the power sample measured at a frequency f_i and time t_j

Empirical analysis is a basic approach to determining the threshold. In this method, radio spectrum measurements are obtained and the threshold determined by observation. In the receiver noise characteristic thresholding technique, the threshold is determined by the receiver's ability to respond to changes in the radio spectrum and the measure of signal-to-noise ratio.

In the p-tile based technique, knowledge of the spectrum usage is required to determine the threshold. Given that the portion of the radio spectrum being utilized is known to be p, then the threshold is satisfied by the fraction of measurements of p that are higher than T.

$$1 - p = CDF(Z = T, R). \tag{2.26}$$

However the fixed threshold techniques have a number of drawbacks that include [48]:

- The techniques are exclusive to a single receiver and cannot detect signals below the noise floor level.
- In order to determine the threshold, a priori knowledge of the bandwidth noise is required.
- There is high level of inaccuracy, as the threshold is determined manually.

Hence, in this work, an adaptive ICDF method was used to determine the threshold. The adaptive ICDF method possesses both characteristics of the adaptive and fixed thresholding techniques. In the adaptive ICDF method, as in the ICDF method, the threshold is a function of probability of false alarm. The experimental approach of this method is explained in section 6.4.

2.6 Orthogonal Frequency Division Multiplexing

The chosen multi-carrier modulation technique for implementation in this research is Orthogonal Frequency Division Multiplexing (OFDM). It's background is therefore presented in this section.

OFDM divides a single wideband data stream into parallel narrowband data streams for transmission [49]. The basic principle behind OFDM is the separation of high rate data stream into

N low rate data streams [16], [50], [51]. Consequently, the streams are modulated to form *N* orthogonal sub-streams [49]. This is achieved by taking an *N*-point Inverse Fast Fourier Transform (IFFT) in the discrete time domain [16], [50].

Considering a 2048-point FFT, the transmitted data is organized into frames as shown in Figure 4.9. Each frame has 8 symbols that has 1 preamble and 7 OFDM symbols. The preamble and OFDM symbols have an identical construct. The dissimilarity is on the 1680 used subcarriers [52]. The preamble’s 1680 subcarriers are Pseudo Noise (PN) and the OFDM symbols are the data carriers [37]. Each OFDM frame has symbols that are mapped to complex symbols during modulation. Modulation takes place at different frequencies that are multiples of each other. Performing the inverse FFT of the mapped signal results in OFDM symbols in the discrete time domain. The 368 guard subcarriers on the OFDM symbol are used to ensure the data subcarriers remain synchronized [52], whilst the 512 cyclic prefix samples for guard intervals are placed between OFDM symbols to facilitate the reduction of Inter Symbol Interference (ISI) caused by multipathing. Thereafter, the OFDM frames are transmitted over the channel. At the receiver, the guard intervals are removed and the complex signals converted to vectors [16]. Consequently, the FFT is performed, the result passed through a one-tap equalizer and the data retrieved [16].

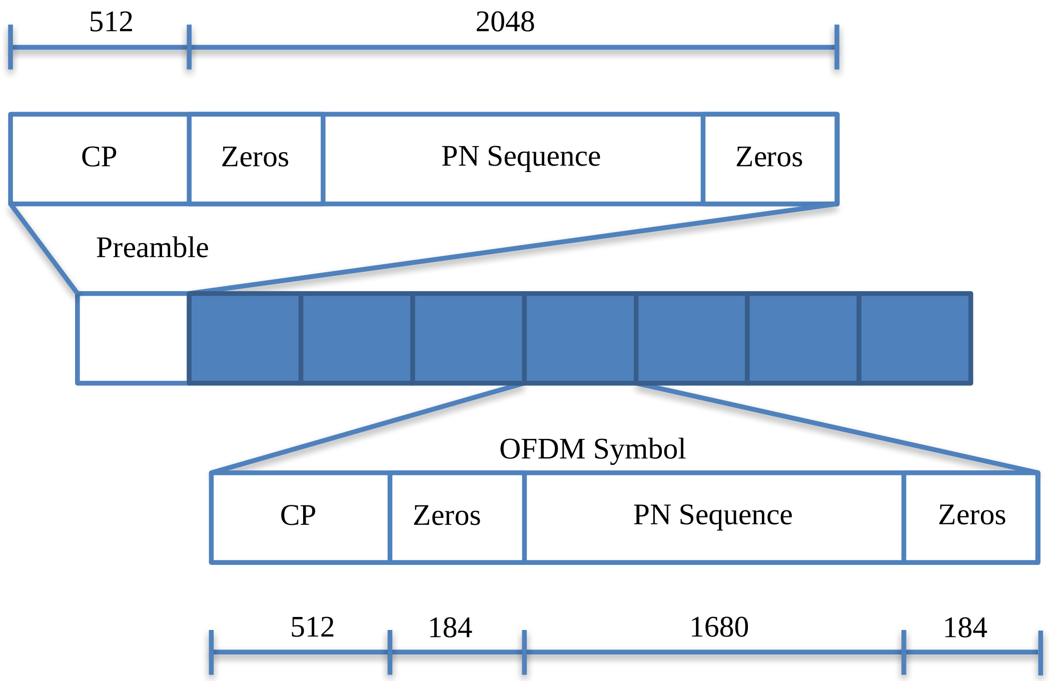


Figure 2.8: 2048-FFT OFDM Frame Structure [37], [52]

Orthogonality of subcarriers is key to the attainment of high spectral efficiency in OFDM. This implies that on every OFDM transmission, all the frequencies are multiples of the inverse of the symbol duration [24]. As compared to Frequency Division Multiplexing (FDM), the sender and receiver have less complexity. OFDM is also resistant to frequency selective fading, offers high transmission bit rates and is less prone to timing offsets [53]–[55].

OFDM multi-carrier modulation method was chosen for implementation in this research, as it is currently deployed in some of the major wireless technologies such as Worldwide Interoperability for Microwave Access (WiMAX) and Long Term Evolution (LTE) [24]. Moreover, OFDM is also widely used in other technologies such as Asymmetric Digital Subscriber Line (ADSL) [56], Digital Audio Broadcasting (DAB) [57], Digital Video Broadcasting – Handheld (DVB-H) [58] and Digital Video Broadcasting – Terrestrial (DVB - T) [59].

2.7 Signal-to-Noise Ratio Estimation

Orthogonal Frequency Division Multiplexing (OFDM) has the capability of transmitting many data streams orthogonal to each other. Each of these streams has at least one subcarrier, which has a certain defined carrier frequency. The carriers are evenly spaced in the frequency domain, and data is transmitted by either phase or amplitude modulation. Only a small segment of the sub-carriers in the transmission band are used. The unused sub-carriers are defined by the midsection and the transmission band-end regions [60]. OFDM initiates channel band optimization at the transmitter. At the receiver, SNR estimation is essential in order to measure the system performance. Some of the other advantages for SNR estimation include “clear channel assessment, soft-decision channel decoding, transmitter power control, adaptive coding and modulation bit loading and hand-off ” [61]. SNR estimation algorithms can be developed for operation in the frequency domain or the time domain [16]. These SNR estimation algorithms can be classified into two basic types: Data-Aided (DA) and Non-Data-Aided (NDA). NDA are also referred to as blind SNR algorithms [60].

Data-aided schemes require knowledge from selected pilot subcarriers for SNR estimation in the payload field. Conversely, for blind SNR estimation in the payload field, no prior knowledge from the pilot sub-carriers is required. The Cyclic Prefix (CP) and OFDM symbols exhibit interdependence which results in interference in the CP interval with outgoing OFDM symbols [60]. This is a major drawback as it calls for a large number of samples to do SNR estimation with minimum error. Baumgartner et al. [62] and Xu et al. [63] proposed a Maximum-Likelihood Estimation (MLE) which is based on the assumption that the noise and signal samples follow a normal distribution with unknown mean and variance. The Expectation Maximization Algorithm (EMA) may also be used as stated by Scaglione et al. [64], which is highly dependent on inferred variables such as channel estimation.

One of the methods for SNR estimation is the blind SNR estimation technique as presented by Y. Li [60]. In this there is no correlation with the channel and signal attributes, and the technique is applicable to single carrier and OFDM signals. The signal received by the cognitive radio in the frequency domain $R_j(k)$ can be represented by the expression below:

$$R_j(k) = C_j(k)H_j(k) + W_j(k), \tag{2.27}$$

where j denotes the time instant and k the subcarrier. $W_j(k)$ denotes the Gaussian noise, $C_j(k)$ the OFDM symbols and $H_j(k)$ the channel frequency response.

By definition the SNR ψ is given as the ratio of the signal power to the noise power, over a defined time period J with subcarriers K, and represented as:

$$\psi = \frac{\sum_{j=J}\sum_{k=K}\left|C_j(k)H_j(k)\right|^2}{\sum_{j=J}\sum_{k=K}\left|W_j(k)\right|^2}. \tag{2.28}$$

Taking into consideration Equation 2.27, this may be written as:

$$\psi = \frac{\sum_{j=J}\sum_{k=K}\left|C_j(k)H_j(k)\right|^2}{\sum_{j=J}\sum_{k=K}\left|R_j(k) - C_j(k)H_j(k)\right|^2}. \tag{2.29}$$

Channel estimation is not applied for SNR estimation in this research as it leads to infinite SNR estimation. The Least Squares channel estimate algorithm below illustrates this.

$$H_j(k) = \frac{R_j(k)}{C_j(k)}. \tag{2.30}$$

In Blind SNR estimation, the estimated SNR at the receiver is calculated by firstly evaluating the Noise-to-Noise Ratio (NNR). A resemblance of a frequency domain packet of bandwidth 202Hz, used for SNR estimation is shown in Figure 2.9.

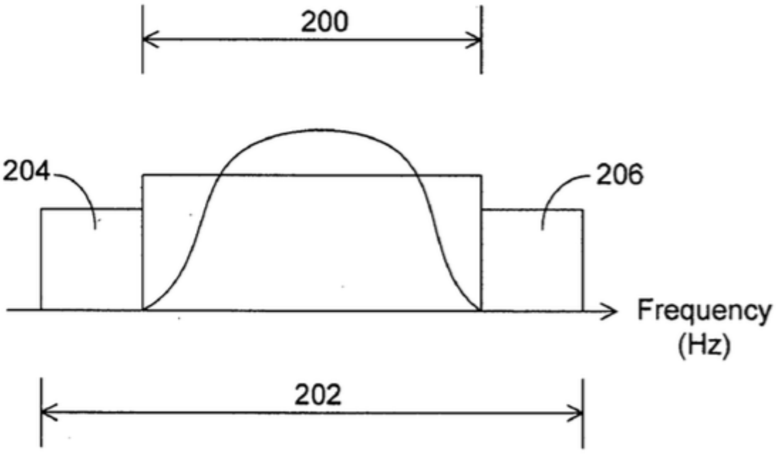


Figure 2.9: Frequency domain data packet [61]

The diagram outlines the different band segments considered for the NNR and SNR calculation. The NNR calculation is explained first. The sum of amplitude squares of the noise in the signal band (200Hz) is calculated first, as shown by the expression below:

$$\sum_{k \in K} |N(j,k)|^2 \tag{2.31}$$

This is then divided by the sum of amplitude squares of the noise in the noise only bands, highlighted in Figure 2.9 (as 204Hz and 206Hz), and expressed as:

$$\sum_{q \in Q} |N(j,q)|^2 . \tag{2.32}$$

Equation 2.33 therefore gives the noise-to-noise ratio over a defined time interval J.

$$\eta = \frac{\sum_{j \epsilon J} \sum_{q \epsilon Q} |N(j,q)|^2}{\sum_{j \epsilon J} \sum_{k \epsilon K} |N(j,k)|^2} . \tag{2.33}$$

After obtaining the noise-to-noise ratio, calculating the SNR ratio follows. To do so, the sum of amplitude squares of the signal in the defined signal band is calculated as shown by Equation 2.34.

$$\sum_{k \in K} |Y(j,k)|^2 \tag{2.34}$$

Calculating the sum of the amplitude squares of the noise in the noise only band follows as shown by Equation 2.35.

$$\sum_{q \in Q} |Y(j,q)|^2 \tag{2.35}$$

Summing the amplitude squares at an instant j over the time interval J for both the signal with noise Equation 2.36 and noise only Equation 2.37 respectively gives the following expressions.

$$\sum_{j \epsilon J} \sum_{k \epsilon K} |Y(j,k)|^2 . \tag{2.36}$$

$$\sum_{j \epsilon J} \sum_{q \epsilon Q} |Y(j,q)|^2 . \tag{2.37}$$

SNR estimation is then calculated by dividing the signal power by the noise power, multiplying by the NNR constant and subtracting 1 as given by the expression below.

$$\psi = \frac{\sum_{j\epsilon J}\sum_{k\epsilon K}|Y(j,k)|^2}{\sum_{j\epsilon J}\sum_{q\epsilon Q}|Y(j,q)|^2}\eta - 1. \tag{2.38}$$

However, in this research the following method was used to estimate the linear SNR [65]:

$$SNR_{linear} = \frac{P_{signal+noise} - P_{noise}}{P_{noise}}, \tag{2.39}$$

where $P_{signal+noise}$ is the received signal power with noise, and P_{noise} is the noise power calculated in the absence of a signal. The method was chosen because of its simplicity. This method is further explained in section 6.2.

2.8 Chapter Summary

In this chapter, spectrum-sensing techniques were introduced and their functionality outlined. A comparison of the detection techniques was also presented and it was noted that the energy detector was chosen for implementation because it is the least complex and therefore can be easily implemented. Another finding from literature was that the performance of the energy sensor is highly dependent on the threshold, therefore the thresholding techniques were analysed and the adaptive ICDF noted to be the most suitable for this research. OFDM was also presented which was used to model the PU. The choice was based on OFDM’s sensing and spectrum shaping capabilities in addition to its flexibility and adaptability, making it the favourable transmission technology for the cognitive radio system. Furthermore, the SNR estimation methods at the reciever were discussed. In the following chapter, the hardware used to implement the real-time sensing system is described. An in depth analysis of the operation of the Universal Software Radio Peripheral is given. The GNU Radio open source software that works with the USRP is also discussed in detail.

Chapter 3

3 Software-Defined Radios

This chapter starts by giving an overview of conventional radio systems. Afterwards the SDR's architecture and operation are explained in detail. GNU Radio and the GNU Radio Companion (the user interface) are then presented, with special attention given to the GNU Radio Companion operation, and creation of Out-of-Tree modules. The USRP platform is then described and the operation of the UHD outlined.

3.1 Conventional Radio Systems

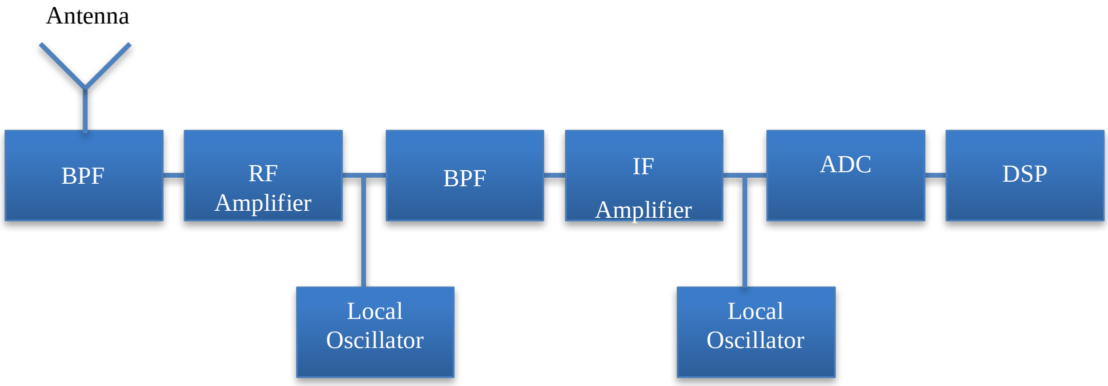


Figure 3.1: Conventional Radio Receiver [66]

Conventional radio systems as shown in Figure 3.1 have hardware modules with dedicated functions that are responsible for receiving and processing the Radio Frequency (RF) signals. In conventional super-heterodyne radio systems, the RF signal is received via the antenna and unwanted frequencies are filtered. The signal is downconverted by mixing it with a new signal from the oscillator, giving the output as an Intermediate Frequency (IF) signal. The intermediate frequency signal is then passed to the Digital Signal Processor (DSP) for decoding. The high frequency effect in conventional radios is resolved by the use of filters and oscillators.

Some of the shortcomings of traditional radio systems are as follows [67], [68]:

- Analog devices have a limited range of signals that they can accommodate due to their narrow band nature.
- As components wear out, the challenge of data integrity arises.
- Analog device outputs are likely to be prone to harmonic distortions.
- Nonlinear components replicate signals at the output at frequencies near the frequency of the original signal.

3.2 Background to Software-Defined Radios

J.Mitola first described the Software-Defined Radio concept in 1992 [69]. This combines the digital radio and software on computers into one technology. The digital radio does the digital signal processing in the digital domain, with the analogue section dedicated to operations such as ADC/DAC and amplification [70]. The FCC defines an SDR as a “Generation of radio equipment that can be reprogrammed quickly to transmit and receive on any frequency within a wide range of frequencies, using virtually any transmission format and any set of standards” [71]. The SDR Forum offers a more generic definition: “Software Defined Radio is a collection of hardware and software technologies that enable reconfigurable systems for wireless networks and user terminals” [72]. However, the main principle behind SDRs is reducing the hardware required for digital signal processing by replacing it with software algorithms that can be run on general-purpose computers, while simultaneously creating a radio that can operate at multiple radio frequencies with the same hardware [16]. In SDRs, software prototypes of varying complexity can be implemented cost effectively, contrary to hardware implementation in conventional radio systems. The increased affordability of ADCs and DACs has contributed greatly to the feasibility of SDR systems, as they convert the RF signal to digital. Another advantage of SDRs is flexibility due to the reconfigurable hardware that is used. This promotes interoperability of wireless systems, making it possible for diverse air interface protocols to be used with the same hardware [73]. A summary of the advantages of SDRs is outlined in Table 3.1.

Table 3.1: Advantages of Software-Defined Radios [66]

Interoperability	Support of multiple standards through multimode and multiband radio capabilities.
Flexibility	Efficient shift of technology and resources.
Adaptability	Faster migration towards new standards and technology through programmability and reconfiguration.
Sustainability	Increased utilization through generic hardware platforms.
Reduced Ownership Costs	Fewer infrastructure, less maintenance and easier deployment.

SDRs may be implemented on three main platforms: Field Programmable Gate Arrays (FPGAs), General Purpose Processors (GPPs) and Digital Signal Processors (DSPs) [74]. FPGAs are suitable for stringent real-time processing requirements for computationally intensive SDR applications. These computationally intensive applications may include Digital Up Conversion (DUC) and Digital Down Conversion (DDC). Their advantage is that simultaneous processing and dedicated hardware multipliers experience low latency [74]. However, the major drawback with FPGAs is their longer implementation time. GPPs do sequential processing, making them relatively slow [74]. Techniques such as multithreading and multi-core enable GPPs to do real-time complex processing, whereas DSPs use the Harvard Bus architecture and Multiplier Accumulator units which make them faster than GPPs. Options for the SDR hardware platform include: The Universal Software Radio Peripheral (USRP), Berkeley Emulation Engine 4 (BEE4) [75], Reconfigurable Open Architecture Computing Hardware (ROACH) [76] and Reconfigurable Hardware Interface for computiNg radiO (RHINO) [76]–[78]

USRP was developed by Mat Ettus from the Ettus Research Group [59], [62]. The USRP is tailor-made for SDRs. The device is made up of the Xilinx Spartan FPGA, which has a relatively low cost. On average, a single USRP costs USD$1700 [76]. The USRP consists of motherboards and daughterboards, which allow a frequency range of up to 6GHz. The board also has a soft processor, which creates an interoperable interface. However having a soft processor poses a disadvantage, as this takes up memory and reduces the digital signal processing on the FPGA. The BEE4 produced by BEEcube is a modified version of the BEE originated by the University of California. This is an FPGA based platform with an existing “repository of low-level component designs” [75]. The BEE4 platform enables researchers to prototype a diverse range of work in minimal time [80], [81]. In addition, it supports high performance computing, MIMO and SDR applications that require high-performance. ROACH [82] was designed by the Square Kilometre Array South Africa Group (SK-SA). This is a reconfigurable platform that consists of digital signal processing FPGAs and supports very good processing speed. The major drawback is the cost; a single board costs USD$5300 and runs on software costing USD$5250, giving a total cost of about USD$10550 [76]. RHINO was developed by the University of Cape Town as a tool to support research for SDRs [77]. The RHINO board spurs innovation in the area of SDR development and radio prototyping. However the RHINO board tools are still under development [77]. The USRP was chosen as the hardware platform for this research primarily because of the availability of the open source GNU Radio with a readily available reference library that can be used to implement the spectrum sensing energy detector. In addition, the USRP falls amongst the cheapest SDR platforms in terms of price and development cost.

3.2.1 Architecture

The architecture of SDRs consists of three main sections: RF section, IF section and the Baseband section. A block diagram highlighting these is shown in Figure 3.2. The RF signals are received via antennas and converted to IF signals. The IF signals are then converted to baseband and the digital signal processing carried out in software [27], [83], [84]. The conversion of RF to IF signals accommodates the hardware and software speed limit of Commercial Off-The-Shelf (COTS) components. Digitalization by the ADC also prepares the signal for further digital signal processing.

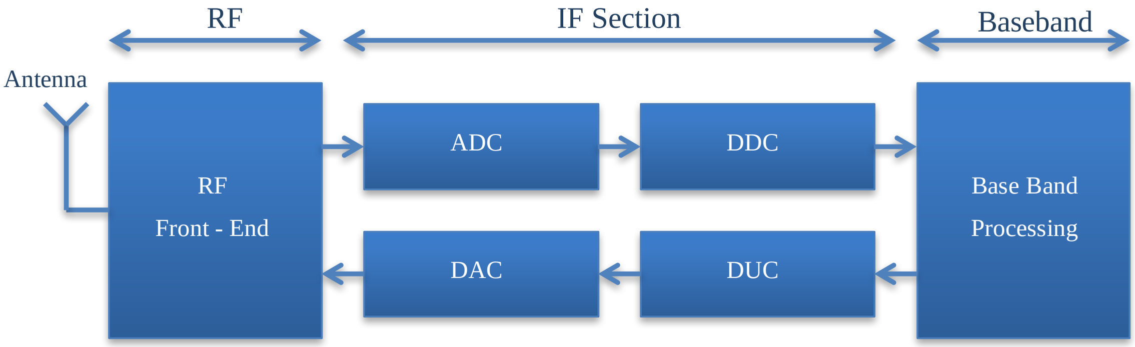


Figure 3.2: Software-Defined Radio Architecture [16], [29], [85]

3.2.1.1 Radio Frequency Section

Antennas receive and transmit RF multi-band signals. RF modules attached to the antennas contain commands for this operation and are controlled by the baseband digital signal processors [16]. Moreover, antennas are capable of space division multiplexing and adaptive beam generation [16].

3.2.1.2 Intermediate Frequency Section

The received RF signal is converted to intermediate frequency signal [16]. During reception, the baseband signal is sampled by the ADCs. The output from the ADC is periodically stored in an FPGA, which has DDCs that do fine-frequency tuning and contain filters that are used for decimation [86]. Thereafter, the digital samples are sent for baseband processing.

3.2.1.3 Baseband Section

The DSP carries out the SDR computations. This includes baseband signal processing, modulation and demodulation, synchronization, timing, encoding and decoding [85]. RF signals are processed in the analogue domain while baseband, and IF signals are processed in the digital domain. Implementing the RF signal in the digital domain forms the ideal SDR. However, this requires very high sampling rates [74]. Therefore the use of hardware poses some limitations on the SDR.

3.2.2 Limitations of Software-Defined Radios

The ADC resolution limits the dynamic range of the USRP-based receivers. This results from the fact that the maximum possible sampling rate is 100 Mega Samples Per Second (MS/s) [86]. According to the Nyquist theorem, the sampling rate should be twice the bandwidth. This therefore stamps the bandwidth to 50 MHz. The dynamic range is defined by the following expression [16], [37]:

$$DR = 6db * n, \tag{3.1}$$

where DR is the ADCs dynamic range, and n is the number of bits. The bus speed has a small range of Mbps - Gbps, limiting the data that can be transmitted from the ADC to the workstation. The Central Processing Unit (CPU) speed affects the rate at which operations are executed per sample. CPUs with less power are used for narrowband algorithms that are less computationally intensive. Widebands are more computationally intensive and require more power as compared to narrow bands. The OFDM used in this research is wideband and therefore computationally intensive. The power consumption can be reduced, but the size of the SDR’s hardware (the ADC, DAC and the RF module) required for signal processing cannot be altered.

3.3 GNU Radio Concept

One of the frameworks for SDRs is the open source toolkit GNU Radio. Its active community support, extensive processing tools library and versatility are the major advantages that draw academic researchers, commercial companies and students to use the framework [70]. GNU Radio is compatible with all personal computer operating systems (Linux, Windows and Mac OS) and can be used with the USRP for SDR development [87]. Figure 3.3 shows the coding languages that constitute GNU Radio.

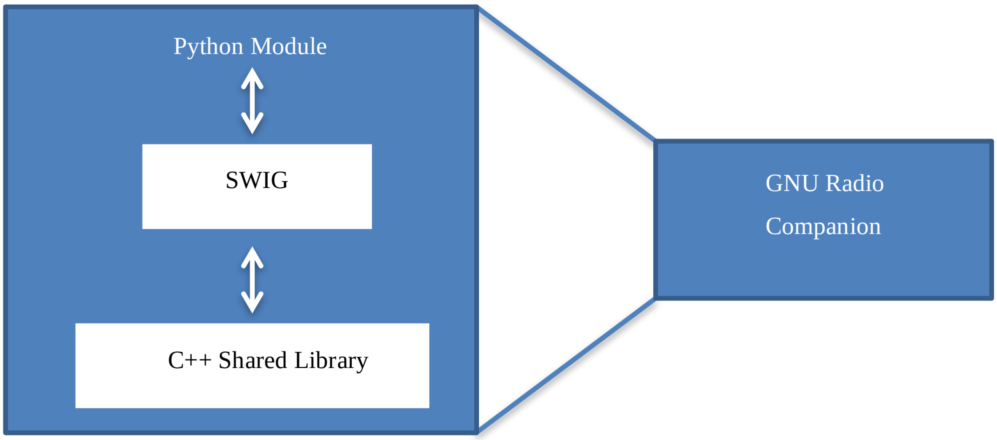


Figure 3.3: GNU Radio software architecture [74]

C++ and Python are the main coding languages used with GNU Radio. C++ is used to realize modules that are used in signal processing such as FFT modules, modulation, demodulation, signal filters, equalizers, coding and decoding. These modules are connected by data streams to form flowgraphs that indicate the data flow direction. A typical flowgraph is presented in Figure 3.4, which shows the sequential flow of data.

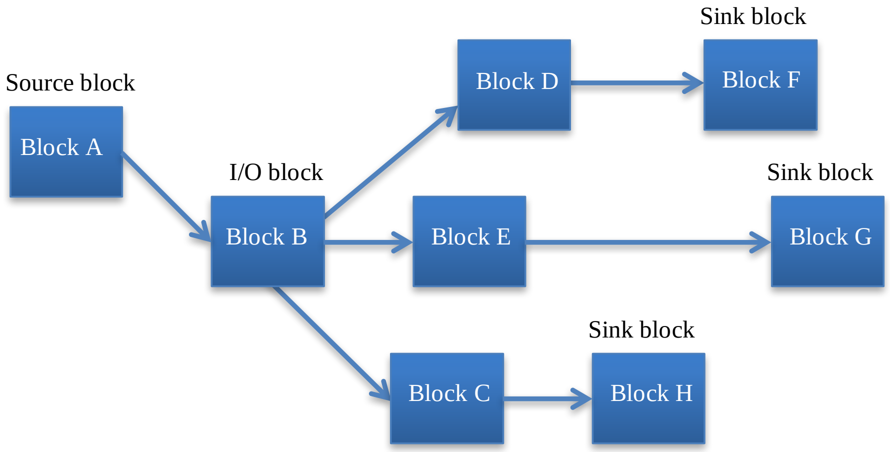


Figure 3.4: Generic Flowgraph of GRC [16]

This is likened to connecting RF hardware blocks in the production of a conventional radio [16]. The major advantage of Python is that it allows reconfiguration of blocks without having to compile each time a change is made [37]. The C++ and Python modules are glued together by the Simplified Wrapper and Interface Generator (SWIG). Moreover, SWIG enables users to develop their own modules and add them to the library. These modules in GNU Radio Companion can be grouped into three classes:

- Sources: These are blocks with output ports only and they inject data into the flow graph. An example is the random source block.
- Input or Output: These receive streams of data from the input port, convert it and inject back into the flow graph. The output is passed to the next block through the output port.
- Sinks: These have input ports only, which receive streams of data or signals. For example a graphical analyzer that converts received data into graphical form. Another example is null sink that stores received data into a file.

A flowgraph therefore starts from a source block and ends with a sink block. Four distinct data types travel through flowgraphs, namely complex (8 bytes), floats (4 bytes), short integers (2 bytes) and bytes (1 byte) [16]. Input streams can have different data rates, but output streams always have the same data rate [16]. GNU Radios have default First In First Out (FIFO) buffers, with size dependent on the operating system installed (LINUX 32kB, Windows 64 kB and Mac OS 64kB). The input and output streams all have buffers linked to them. These introduce additional latency in the GNU Radio Companion implementation, as latency is directly proportional to buffer size [16]. Blocks read buffer data before processing, and afterwards write data to the succeeding buffer as illustrated in Figure 3.5. The blocks are designed such that a single output stream can be connected to multiple input data streams, and several blocks can be combined to form one hierarchical block [16].

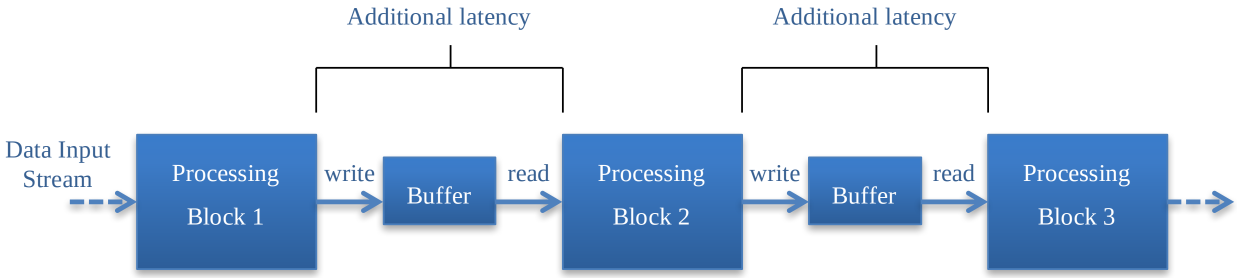
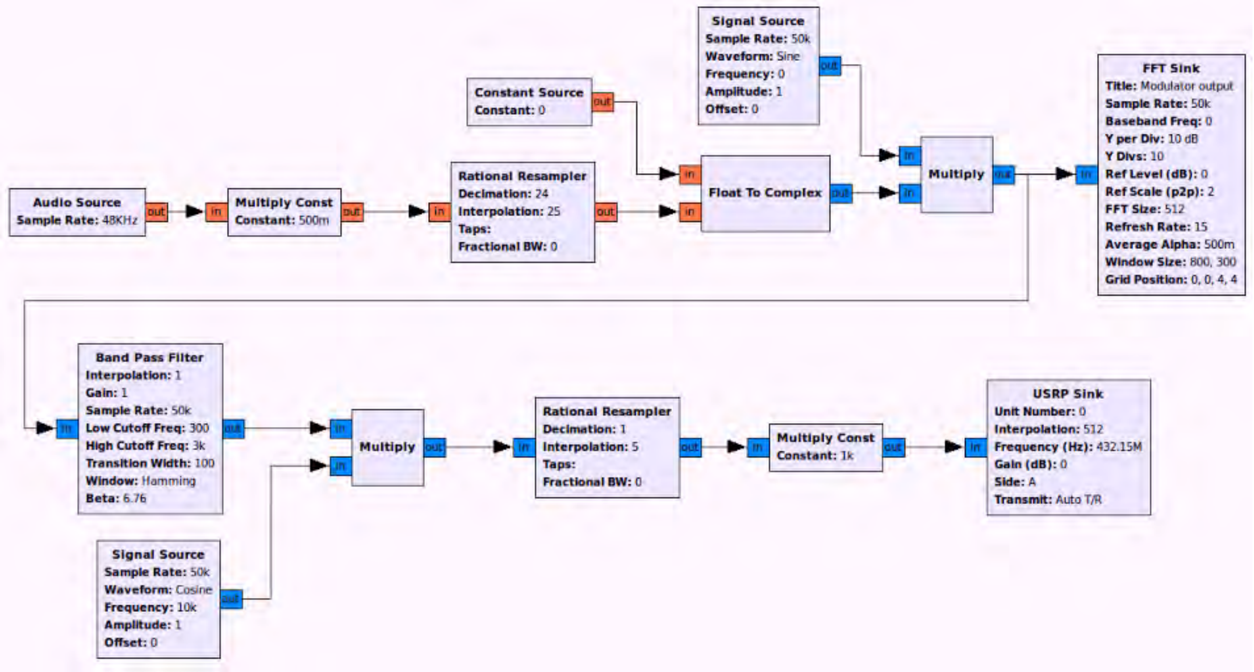


Figure 3.5: GNU Radio Companion processing blocks [37]

GNU Radio Companion (GRC) is the graphical user interface for GNU Radio. GRC offers a visual type of programming by drag-and-drop of modules/blocks. The blocks are made using C++, and the connections done using Python [16]. The GRC blocks can be easily configured by adjusting parameters in the top block class. When the GRC application is running, the code in the blocks is executed sequentially [54]. However, GRC is still under development and only a limited range of blocks is available in the library. In addition, the visual programming offered by the GRC blocks is limited in versatility compared to C++ and Python textual programming. Therefore researchers recommended the use of Python or C++ to create new blocks with additional functionality. A popular application for GRC is in the learning environment where flowgraphs can be quickly implemented for demonstration purposes [55], [88]. Figure 3.6 shows a GRC example: A single sideband transmitter that uses a bandpass filter.



Audio Source
Sample Rate: 48KHz

Multiply Const
Constant: 500m

Rational Resampler
Decimation: 24
Interpolation: 25
Taps:
Fractional BW: 0

Constant Source
Constant: 0

Float To Complex

Signal Source
Sample Rate: 50k
Waveform: Sine
Frequency: 0
Amplitude: 1
Offset: 0

Multiply

FFT Sink
Title: Modulator output
Sample Rate: 50k
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 512
Refresh Rate: 15
Average Alpha: 500m
Window Size: 800, 300
Grid Position: 0, 0, 4, 4

Band Pass Filter
Interpolation: 1
Gain: 1
Sample Rate: 50k
Low Cutoff Freq: 300
High Cutoff Freq: 3k
Transition Width: 100
Window: Hamming
Beta: 6.76

Signal Source
Sample Rate: 50k
Waveform: Cosine
Frequency: 10k
Amplitude: 1
Offset: 0

Multiply

Rational Resampler
Decimation: 1
Interpolation: 5
Taps:
Fractional BW: 0

Multiply Const
Constant: 1k

USRP Sink
Unit Number: 0
Interpolation: 512
Frequency (Hz): 432.15M
Gain (dB): 0
Side: A
Transmit: Auto T/R

Figure 3.6: GRC single sideband transmitter [89]

3.4 Structure of GNU Radio Module

GNU Radio modules come with the following subdirectories: *apps*, *cmake*, *docs*, *lib*, *swig*, *python*, *include* and *grc* as shown in Table 3.2. These subdirectories have *autoconf*, *libtool* and *automake* tools that make them compile independently. Moreover, they facilitate connecting code written in C++ to Python [16].

Table 3.2: GNU Radio Module Subdirectories

Subdirectories	**Description**
apps/	Consists of complete applications installed in the system as well as GRC blocks.
cmake/	Has instruction files for autotools and cmake command used to locate GNU Radio libraries.
docs/	Consists of instructions to extract documents from the C++ and Python files.
lib/	Provides storage for source (.cc and .h) files. These are specifically meant for all other languages except python. C++ header files are stored in *include/* for them to be exported, and *lib/* for use during compiling.
swig/	Consists of SWIG tool instructions that are followed to build python interfaces used by C++ classes.
python/	Used to store all python files.

3.4.1 Creating Out-of-Tree Modules

GNU Radio can be extended to increase its functionalities and blocks. This is done by building new modules, referred to as Out-of-Tree (OOT) modules. The OOT modules are not part of the GNU Radio main source tree, but can be added by submitting the blocks to the GNU Radio repository (CGRAN). Module development in GNU Radio has been made easy by the use of a *gr_modtool* script, used to edit *makefiles*, for the desired requirements. This enables the developers to focus only on the digital signal processing programming. After modifying the python and C++ code, the blocks are rebuilt and reinstalled by executing the following commands in the terminal:

```
$ cd build/   # Access build directory
$ cmake ../             # Tell Cmake that all its config files are one directory up
$ make                  # Start building
$ make test             # Build test again if the change affects tests
$ sudo make install     # Install block in GNU Radio
```

Whilst creating new blocks in GRC, errors are prone to occur. However, troubleshooting can be done using a number of tools [16]:

- **Quality assurance (QA) codes:** QA codes can be added for modified or written blocks. These consist of test codes that are used to check for errors in the individual blocks. The *ctest -V* command may also be used in place of the *make test* command to give a more narrative description. Print statements may also be used to test each segment of code.
- **GRC and graphical sinks:** Graphical sinks can be easily attached to blocks to observe the graphical output of each block. These sinks can be classified into two; WX GUI Widgets and QT GUI Widgets. Examples of these include oscilloscopes and FFT blocks. The suitable sink widget is selected to display the output data of a block.
- **Saving output data into files:** This is an all-inclusive method that takes advantage of external tools to analyze the output data from blocks. A *file_sink* is attached to the output port of a block and when the flowgraph runs, data is saved into *.dat* files. These files are then opened in Octave, MATLAB or SciPy for off-line analysis.

3.5 Universal Software Radio Peripheral

The Universal Software Radio Peripheral is a hardware platform developed by Matt Ettus, that is used to implement SDR systems [79], [90]. This platform is made up of three main sections: the radio frequency antennas, motherboard and daughterboards as highlighted in Figure 3.7. The main purpose of the USRP is to facilitate RF – IF conversion and vice-versa, thus providing the link between the baseband signal and intermediate frequency [50], [90], [91].

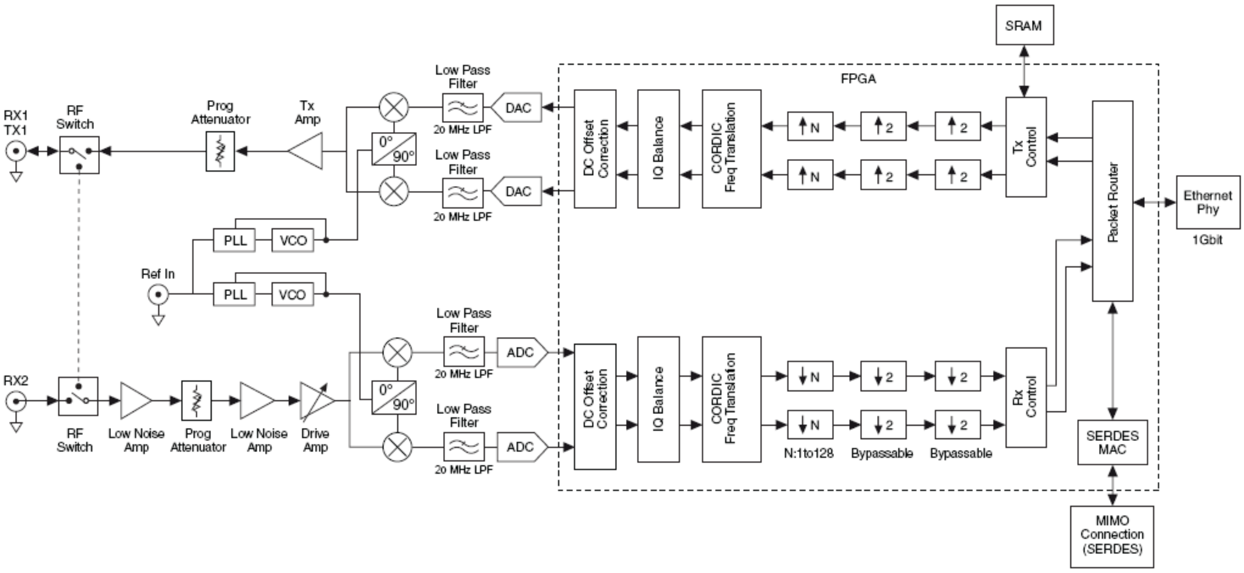


Figure 3.7: Schematic Diagram of a N210 Universal Software Radio Peripheral [90]

3.5.1 USRP Families

The USRP family of products consist of the N-series, E-series, USRP1 and USRP2 [90], [92]. Table 3.3 outlines the difference in the Ettus Research family of products. Depending on the daughterboard, the USRP2 can operate at a frequency range of 0 to 5.9 GHz [16]. The latest model of the USRP N-series is the N210. Not only can these USRPs be used for research, they have applications in industry as well.

Table 3.3: USRP family specifications [16], [70]

	USRP1	**USRP2**	**N210**
Interface	USB 2.0	Gigabit Ethernet	Gigabit Ethernet
FPGA	Altera EPIC12	Xilinx Spartan 3 2000	Xilinx Spartan 3A-DSP 3400
RF bandwidth to/from host	8MHz @ 16 bits	25 MHz @ 16 bits	25 MHz @16 bits
Cost	$700	$1400	$1400
ADC samples	12bit, 64MS/s	14bit, 100MS/s	14bit, 100MS/s
DAC samples	14bit, 128MS/s	16bit, 400MS/s	16bit, 400MS/s
Daughterboard capacity	2 TX, 2 RX	1 TX, 1 RX	1 TX, 1 RX
SRAM	None	1 Megabyte	1 Megabyte
Power	6V, 3A	6V, 3A	6V, 3A

The N210 USRP used in this research has different bandwidths at the different points during the signal processing chain [86]. These are the analogue bandwidth, FPGA bandwidth and host bandwidth [86]. 40 MHz is allocated as the analogue bandwidth for the purpose of hindering aliasing. In reality, of the 40MHz defined between the RF port and the baseband interface, only a small segment is used. The USRP N210 has an FPGA processing bandwidth of 100MS/s for both the ADC and DAC convertors. This is defined as the rate at which samples are sent and received [86].

3.5.1.1 Motherboards

Motherboards facilitate the baseband signal to IF and IF to baseband conversion during reception and transmission respectively. This is done by the aid of an onboard FPGA and memory [37]. During reception the motherboard converts the received analog signal to digital using two

14-bit high speed 100 MS/s analog-to-digital convertors, and the output IF signal processed by the FPGA. The FPGA chip is loaded with channelization firmware and FPGA images during the initial operation. This enables the USRP to communicate with any UHD [16]. FPGAs handle digital down and up conversion using the onboard decimating low pass filters. The digitized signal from the decimating low pass filter has a high data rate. To lower the rate the signal is sampled, thus making it compatible with the Gigabit Ethernet (GbE) cable and computer speeds [16], [37]. It is important to note however, that FPGAs do not handle any complex digital signal processing computations; rather the computer does these. During transmission, the DACs have a sampling rate of 400MS/s, which converts the digital signal to analog. Consequently, the analog signal is converted to IF and transmitted over the channel via the RF antenna as an RF signal [16].

3.5.1.2 Daughterboards

Daughterboards on the other hand specify the bandwidth processed by the motherboard [16], [92], [93]. Figure 3.8 shows the variety of daughterboards that enable the USRPs to operate at a range of frequencies. These are classified into three categories, TX/RX, RX and TX. The TX/RX can be used for both transmit and receive path, such boards include the RFX900, RFX1200, RFX1800, RFX2400, WBX, SBX and XCR2450. Among these is the SBX daughterboard with frequency range 400 MHz to 4.4 GHz used in this research [5]. The TX daughterboards are specific to the transmission path, for example the BasicTX and LFTX. Likewise, the RX daughterboards are specific to the receive path and these include: BasicRX, LFRX, DBSRX and TVRX2.

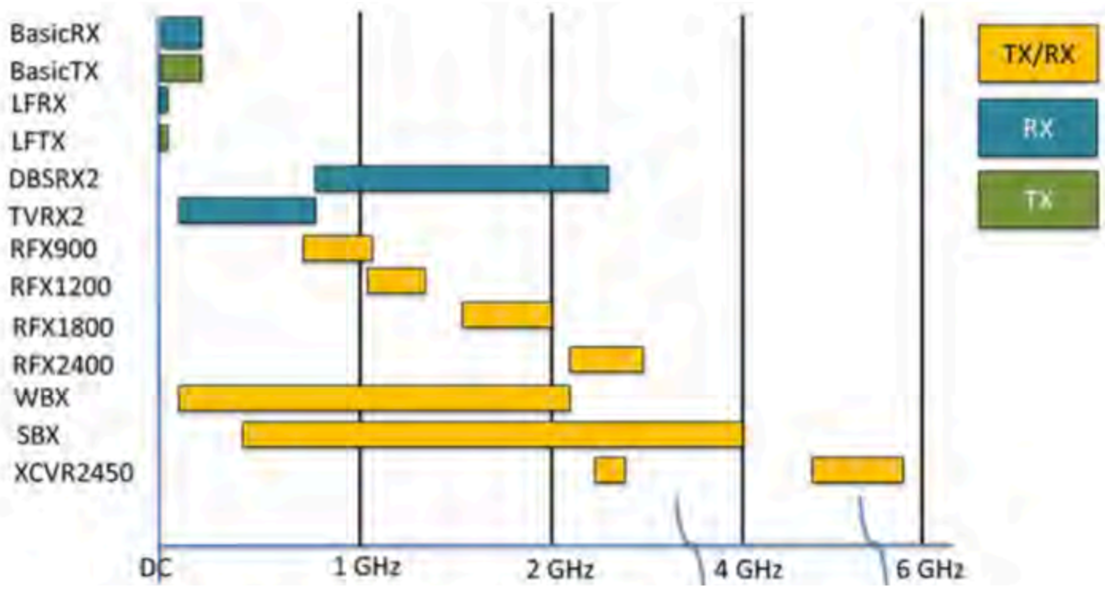


Figure 3.8: RF Daughterboard Frequency Coverage [92]

Ettus Research recommends that the choice of the daughterboard used be specific to the application frequency coverage requirement [92]. For instance, WiMAX applications require a coverage of 2.5GHz with an RX capability, therefore SBX is the recommended daughterboard [92]. High Frequency (HF) communication requires a frequency coverage of 3-30MHz, with Tx and Rx capabilities, hence the recommended daughterboards are the LFRX and LFTX [92].

3.5.2 USRP Hardware Driver

This is a software Application Programmimg Interface (API) that works in conjuction with GNU Radio to realize real-time SDR systems. UHDs foster development of new applications and control all the operations of the USRP [70]. The standard software combination supports interoperability with different frameworks which include LabVIEW, MATLAB/Simulink, and RFNoC [94]. In addition, this also reduces the development time and effort. This software can be acquired open-source or with an Ettus research license [94].

3.6 Chapter Summary

This chapter introduced the SDR as a solution to the limitations of conventional radio systems. The SDR system architecture and system operation were presented. The GNU Radio concept was also discussed and the structure of GNU Radio software modules outlined, to give an overview of how the SU and PU would be implemented on the GRC platform. The hardware component of the SDR, the N210 USRP kits was then analysed. It was noted the most suitable daughterboard to use for this research was the SBX daughterboard, as it enabled the USRP to operate within the 400MHz to 4.4GHz range, which covers the 2.2 GHz frequency set for the transmitting and receiving signals. The next chapter presents the simulation results for the analysis done using MATLAB.

Chapter 4

4 Simulation Results and Analysis

In this chapter, the performance of the energy-based spectrum sensing is evaluated. The efficiency of the sensing algorithm is measured based on the following key performance metrics: Probability of detection, probability of false alarm, probability of misdetection, and the signal-to-noise ratio. The performance of the energy detector is evaluated over an AWGN channel using the MATLAB simulation tool. The simulation results obtained are used as a benchmark, and later compared to the energy detector's real-time implementation.

4.1 Simulation Specifications

MATLAB software (R2013a), a high-level language tool, is used to model the spectrum sensing algorithm. The PU (modelled in MATLAB) generates an OFDM test signal with Quadrature Phase Shift Keying (QPSK) modulation. AWGN is added to the modulated OFDM signal, and the result is the SU's received signal. The energy test statistic is then computed from the received signal and compared to a predetermined threshold, at a defined probability of false alarm. To improve the accuracy of the detector 10 000 Monte-Carlo iterations are performed. Table 4.1 (overleaf) shows a summary of the parameters used for the simulated cognitive radio sensor system.

Table 4.1: Simulation Parameters

PU signal	OFDM
Channel noise	AWGN
Number of samples/ FFT size	128, 256, 512, 1024, 2048
Signal-to-noise ratio	-25 dB to 0 dB
Probability of false alarm	0 to 1
Desired probability of detection	0.9
Desired probability of false alarm	0.1
Number of iterations	10 000

4.2 Channel Noise Effects

The performance metrics considered in evaluating the energy detector are the probability of detection, probability of false alarm, and the probability of misdetection.

Probability of detection ($\boldsymbol{P_d}$): Refers to the probability that the SU correctly declares that a PU is present, when the PU is present. High detection performance requires that the probability of detection is maximized.

Probability of false alarm ($\boldsymbol{P_{fa}}$): Refers to the probability that the SU incorrectly declares that a PU is present, when the PU is actually absent. Too many false alarms signify that spectrum opportunities are being missed. Hence, to maximise spectrum efficiency the probabilty of false alarm must be minimized.

Probability of misdetection ($\boldsymbol{P_m}$): Refers to the probability that the SU declares that the PU is absent, when the PU is actually present. Occurrence of misdetection may result in interference of the PU by the SU, therefore probability of misdetection has to be monitored to avoid too many collisions.

Signal-to-Noise Ratio (SNR): Refers to the primary users signal strength, with respect to the noise. The factors that affect the SNR at the cognitive radio receiver include the distance between the primary user and the secondary user (path loss), transmission power of the primary user and the medium of propagation [12].

Signal-to-Noise Ratio Wall (SNR_{wall}): Refers to the minimum SNR that the detector can achieve with the desired probability of detection ($P_d \geq 0.9$), and probability of false alarm ($P_{fa} \leq 0.9$), according to the IEEE 802.22 Wireless Regional Area Network (WRAN) standard [95].

4.3 Effect of Sample Size on Probability of Detection

Figure 4.1 shows the plot of signal strength (SNR = -25 dB to 0 dB) versus the probability of detection with varying sample size (N = 128, 256, 512, 1028, and 2048). The probability of false alarm is set to 0.1.

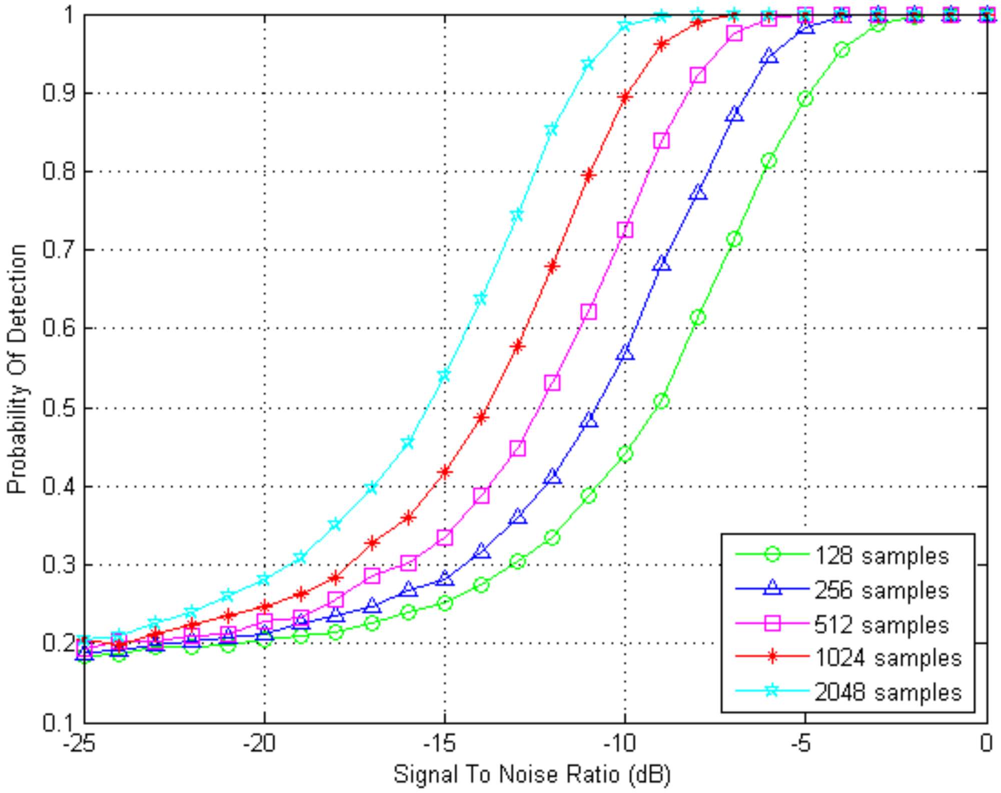


Figure 4.1: Signal-to-Noise Ratio versus Probability of Detection with varying Sample Size ($P_{fa} = 0.1$)

The plot shows an increase in probability of detection with increasing signal-to-noise ratio. This implies that as the signal strength increases (with increasing signal-to-noise ratio) the probability of detecting a PU also increases, which is in consonance with the findings in Chapter 2. The probability of detection can also be increased, even at lower signal-to-noise ratios, if the sample size is increased. For instance, with a sample size of 2048 the SNR_{wall} is -12 dB, and when the sample size is 128, the SNR_{wall} becomes -5 dB. The plot also shows that as the signal-to-noise ratio reduces, the probability of detection for all stated sample sizes approaches 0.2, as seen with an SNR of -25 dB. This implies that at very low SNRs, the ability of the energy detector to correctly recognise the primary user is poor, regardless of sample size.

4.4 Effect of Signal-to-Noise Ratio on Probability of Detection with varying Probability of False Alarm

Figure 4.2 shows a plot of signal strength (SNR = -20 dB to 0 dB) versus probability of detection, with varying probabilities of false alarms (P_{fa} = 0.01, 0.1 and 0.2). An analysis of the plot below shows that increasing the probability of false alarm increases the probability of detection. For instance at an SNR of -13 dB, the probability of false alarm of 0.01 has a probability of detection of 0.48. When the probability of false alarm is 0.1, the probability of detection is 0.71. Doubling the probability of false alarm to 0.2, further increases the probability of detection to 0.84. It is important to note that at SNR = -8 dB regardless of the choice of probability of false alarm, the energy detector performs optimally and can easily distinguish a primary user in the spectrum from the noise. Also, a reduction in the signal strength greatly affects the performance of the detector, which is observed from the reduction in probability of detection, for all false alarms. Below -20 dB, the performance continues to deteriorate significantly, and it becomes challenging for the detector to distinguish the determinant (PU) signal from the noise signal.

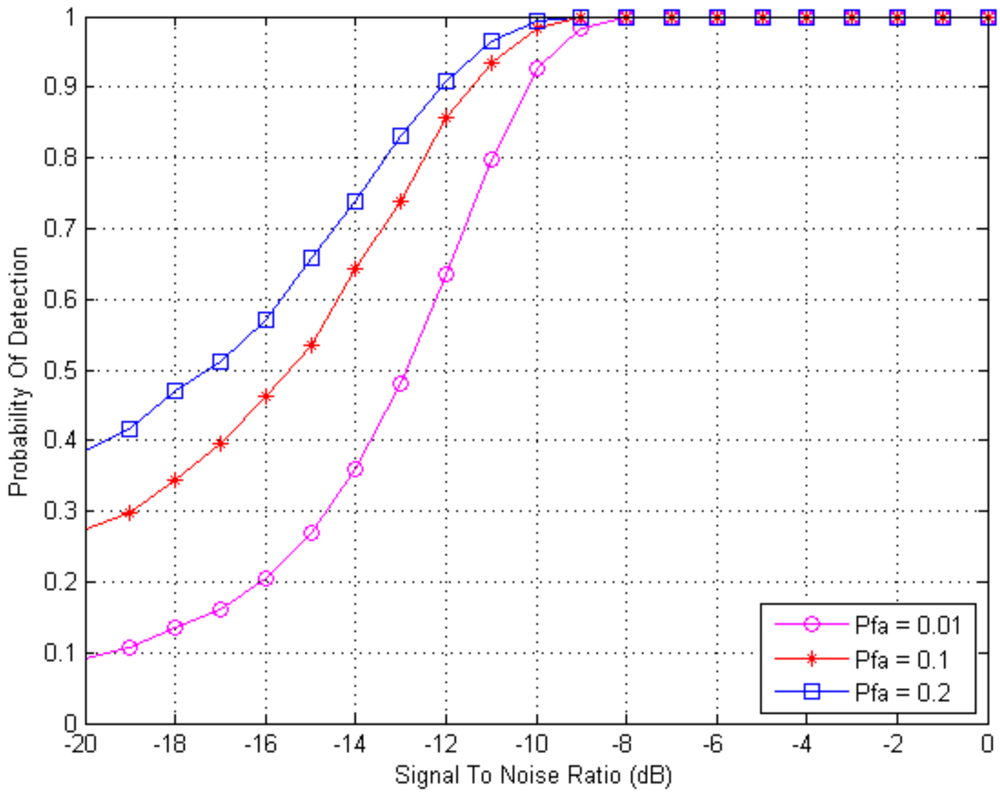


Figure 4.2: Signal-to-Noise Ratio versus Probability of Detection with varying Probability of False Alarm

4.5 Receiver Operating Characteristics

The Receiver Operating Characteristics (ROC) analysis provides a mathematical framework that computes and compares the probability of detection and the probability of false alarm, thus providing a precise, quantitative way of assessing accuracy [96]. Each point on the curve resembles the different possible combinations of probability of detection and probability of false alarm at different thresholds during the sensors operation. The accuracy is a measure of the Area Under the Curve (AUC) [97]. The closer the area is to 1 unit square, the greater the performance of the detector. Random guessing is signified by an area of 0.5 unit squares. Hence, a satisfactory test should have an area between 0.5 and 1 unit square [97], [98]. Figure 4.3 shows the ROC curves for the energy detector with varying SNRs.

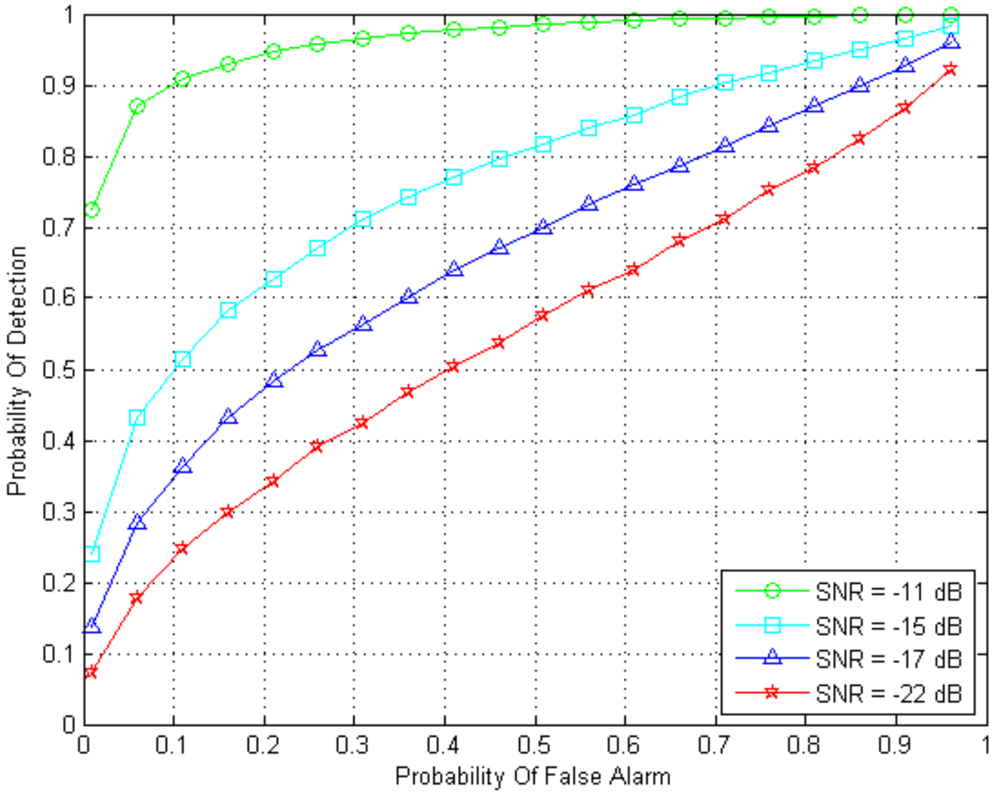


Figure 4.3: ROC curves at different Signal-to-Noise Ratios with Energy Detector

The area under the curve decreases as SNRs reduce from -11 dB to -22 dB, as expected from literature [12]. This means that increasing the SNR increases the probability of detection, thus increasing the performance of the energy detector. The detector shows high performance at SNR = -11 dB, as it has a high probability of detection. Decreasing the SNR reduces the detector's performance. The plot shows that only SNR = -11 dB achieves the desired values for probability of detection and probability of false alarm values according to the IEEE 802.22 WRAN standard.

4.6 Complementary Receiver Operating Characteristics

Figure 4.4 shows a plot of probability of false alarm versus probability of misdetection, with varying signal strength. As observed from the plot, the probability of misdetection decreases as the SNR increases. In addition, as the probability of false alarm increases, the probability of misdetection decreases. Unlike the ROC curves, as the area under the CROC curves reduces, the performance of the energy detector increases. The plot shows that the AUC decreases as the SNR increases, implying that the probability of misdetection is reduced. Hence, the likelihood of PU and SU collisions reduces with decreasing AUC.

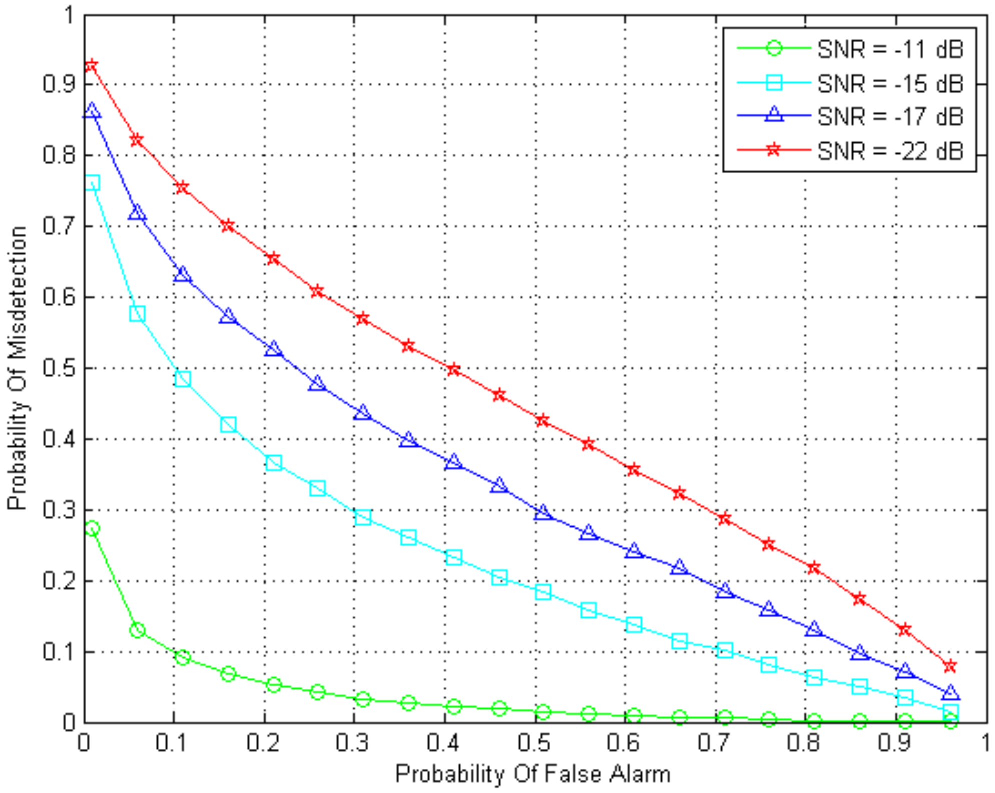


Figure 4.4: Probability of False Alarm versus Probability of Misdetection with varying Signal-to-Noise Ratio

For instance at SNR = -11 dB, at probability of false alarm of 0.1, the misdetection probability is 0.1; whereas, for SNR = -15 dB, at probability of false alarm of 0.1, the misdetection probability is 0.5.

4.7 Chapter Summary

In this chapter the simulation results of the energy detector were presented and discussed. The significance of this is that it provides a benchmark for the real-time implementation. The simulation results showed that the probability of detection is a function of the sample size, SNR and probability of false alarm. In particular, beyond a certain minimum SNR, the effect of sample size ceases to be significant. On analysing the effect of probability of false alarm and probability of detection, it was noted that detectors with higher probability of false alarm performed better than those with lower probability of false alarm. Also, signals with high SNR have high probability of detection. Moreover, as the probability of false alarm increased, the probability of misdetection decreased. The next chapter gives a concise description of the energy detector in SDR. The implementation procedure of the energy detector on the GNU Radio platform is given.

Chapter 5

5 Implementation of Energy Based Detector on GNU Radio and USRP Platform

This chapter describes the procedures carried out during the implementation of the spectrum sensor. Firstly, the system design is outlined that consists of the testbed layout, GNU Radio installation instructions, USRP configuration and the testing procedures. Detailed implementation steps for the spectrum sensing on USRP using GRC are then presented, and the new modules created explained in detail. Lastly, a summary is given at the end of the chapter.

5.1 System Design

Figure 5.1 shows the laboratory testbed setup of the real-time energy sensor. Two N210 USRPs (USRP Tx and USRP Rx) with Vert900 antennas that are connected to a single Personal Computer (PC) via three CAT6 Gigabit Ethernet (GbE) cables with RJ-45 connectors, and a GbE switch. The PC used in this research runs on MaC OS, with GNU Radio software version 3.7.7 installed on it.

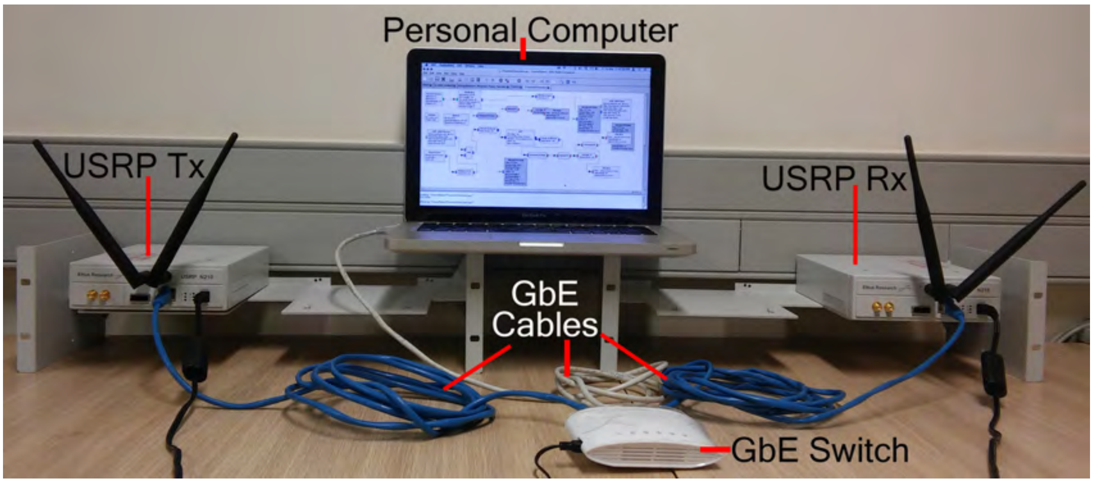


Figure 5.1: Experimental Setup for Spectrum Sensing using GNU Radio and USRP

The datapath from Tx/Rx modules to the host Central Processing Unit (CPU) is shown in Figure 5.2. The blue and red arrows in the block diagram represent the transmission and reception paths respectively. The GRC modules generate an OFDM test signal that is sent to the USRP transmitter (Tx) which acts as the primary user. The USRP Rx acts as the secondary user (cognitive radio), that continuously scans the radio spectrum in anticipation of a primary user. The secondary user sends the received signal to the energy detector implemented using GRC on the PC and a decision is made based on the energy test statistic.

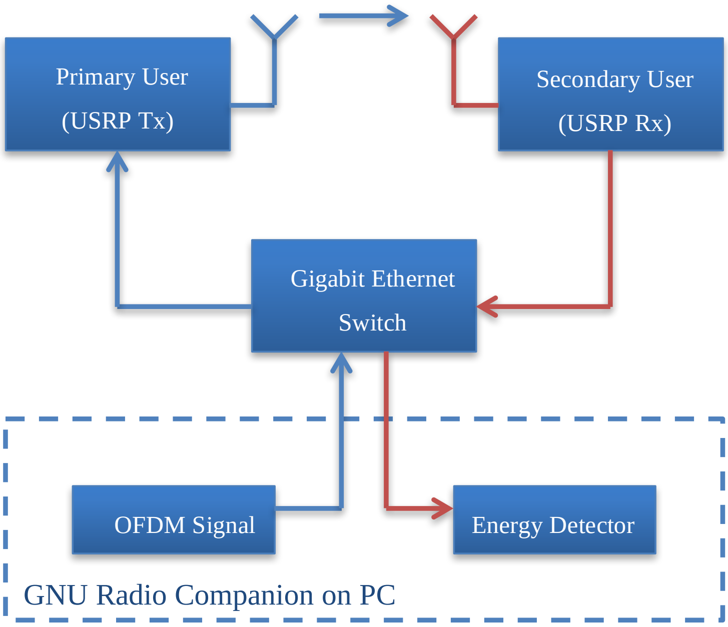


Figure 5.2: Testbed Architecture Block Diagram

The USRP N210 devices have a full duplex stream interface which can be used to connect the PC as the host interface. The interface permits both transmission and reception of 16-bit I/Q data at the same time [86]. It is important to note that connecting a single host to two USRP devices halves the sample rate to 12.5 MS/s. The distance between the transmitter and receiver is set to be 1m in direct line-of-sight (LOS).

5.1.1 GNU Radio Installation

GNU Radio can be installed on any version of Mac OS X (starting from Mac OS X 10.4) with compatible Integrated Development Environment (IDE) Xcode software. The IDE is free and downloadable from the Apple Store [99]. GNU Radio installation can be done using one of the following two methods: installation from source or via Macports. The instructions for installing from source are covered in detail on the GNU Radio website [99], [100]. In this research, GNU

Radio is installed on Mac OS X 10.10.1 via MacPorts. MacPorts is installed first, the installation instructions can be followed from [101]. Consequently, GNU Radio is then installed by running the command below in the terminal. In addition, this also installs all the directories required.

```
$ sudo port install gnuradio
```

After GNU Radio installation, the USRP kits are configured and tested to enable communication with GRC.

5.1.2 USRP Configuration and Testing

The first step is to configure the Ethernet port on the PC. Thereafter the Ethernet connection is tested by running the following commands:

```
$ sudo ifconfig en0 192.168.10.1

$ sudo ifconfig en0
```

The USRPs are configured next. To do this, the devices are switched on in safe mode. The Ethernet cable is then connected to a single USRP N210 and the other end to the PC. Whilst in safe mode, the blue S2 switch located at the central position of the daughterboard is pressed down for 5 seconds and then powered. The light-emitting diodes (A, C, D, E and F) on the USRP front light up and in a sequential manner: F lights up and remains on, followed by light D that shines rapidly and stays on as well. E, C and A are then activated in sequence, each flashing three times and then switching off. The procedure is carried out for both USRP devices. To test the USRPs connection to the PC, the command *uhd_find_devices* is executed in the command line and the output returned as shown in Figure 5.3.

```
$ uhd_find_devices

Mac OS; Clang version 5.1 (clang-503.0.40); Boost_105700;
UHD_003.008.002-MacPorts-git-8c20712d(20150327)
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    type: usrp2
    addr: 192.168.10.2
    name: Right
    serial: FFFFFF
```

Figure 5.3: Single USRPs feedback to uhd_find_devices

The command returns the specifications of UHD Device 0 connected to the PC showing that the USRP has been detected. It is important to note that the Internet Protocol (IP) address of the personal computer (192.168.10.1) has to be in the same network as the USRP’s default IP of 192.168.10.2. This allows communication between the two. To avoid conflicting IP addresses, as both USRPs come with the same default IP address, the second USRP IP address is changed to 192.168.10.3 by running the command in Figure 5.4 on the terminal, to return the output in Figure 5.5.

```
$ /opt/local/share/uhd/utils/usrp_burn_mb_eeprom
--args="addr=192.168.10.2" --values="subnet=255.255.255.0,
gateway=255.255.255.255, ip-addr=192.168.10.3"
```

Figure 5.4: Command to alter USRP device IP address

```
Mac OS; Clang version 5.1 (clang-503.0.40); Boost_105700;
UHD_003.008.002-MacPorts-git-8c20712d(20150327)
Creating USRP device from address: addr=192.168.10.2
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
Fetching current settings from EEPROM...
    EEPROM ["subnet"] is "255.255.255.255"
    EEPROM ["gateway"] is "255.255.255.255"
    EEPROM ["ip-addr"] is "192.168.10.2"
Setting EEPROM ["subnet"] to "255.255.255.0"...
Setting EEPROM ["gateway"] to "255.255.255.255"...
Setting EEPROM ["ip-addr"] to "192.168.10.3"...
```

Figure 5.5: Terminal output after altering USRP device IP address

The device is power cycled to save the new IP address and a test done to confirm the IP addresses. Figure 5.6 shows the command executed in the terminal and the output observed thereafter.

```
$ uhd_find_devices

Mac OS; Clang version 5.1 (clang-503.0.40); Boost_105700;
UHD_003.008.002-MacPorts-git-8c20712d(20150327)
----------------------------------------------------
-- UHD Device 0
----------------------------------------------------
Device Address:
    type: usrp2
    addr: 192.168.10.2
    name: LEFT
    serial: FFFFFF

----------------------------------------------------
-- UHD Device 1
----------------------------------------------------
Device Address:
    type: usrp2
    addr: 192.168.10.3
    name: RIGHT
    serial: FFFFFF
```

Figure 5.6: IP address test output

Running the *uhd_usrp_probe* command with both devices connected via the Gigabit Ethernet switch shows the overall test result for all USRP configurations. The results obtained show that the host and USRPs are configured correctly and are ready to be used for the GRC experiments. The overall network design of the system is shown in Figure 5.7. This setup allows operation with a single personal computer.

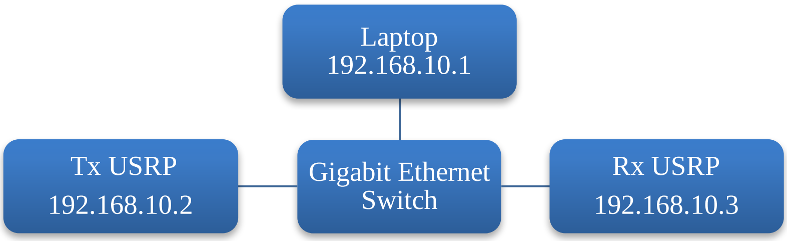


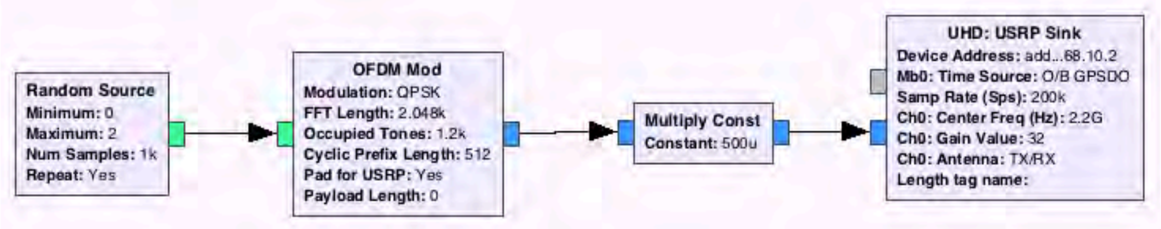
Figure 5.7: Cognitive Radio Spectrum Sensor Network Design

5.2 Spectrum Sensing in GNU Radio Companion

As discussed in section 5.1.2, the Tx USRP transmits the primary user OFDM test signal. To do this, OFDM blocks are implemented in GRC. In this section, the procedures for implementing the primary user and secondary user in GRC are outlined.

5.2.1 Primary User Implementation in GRC

Pre-existing blocks in the GRC library are used to give a quick start to the experimental design of the primary user. The random source block *(Random Source)* generates a recurring set of 1000 samples of integers that are tagged at every 96 bits. Tagged bits are passed to the configurable OFDM modulation *(OFDM Mod)* hierarchical block as shown in Figure 5.8.



Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

OFDM Mod
Modulation: QPSK
FFT Length: 2.048k
Occupied Tones: 1.2k
Cyclic Prefix Length: 512
Pad for USRP: Yes
Payload Length: 0

Multiply Const
Constant: 500u

UHD: USRP Sink
Device Address: add...68.10.2
Mb0: Time Source: O/B GPSDO
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 32
Ch0: Antenna: TX/RX
Length tag name:

Figure 5.8: OFDM Signal Generation Flowgraph in GRC

The input stream to the modulation block has 4 more bits added to each tag. Afterwards the stream is split into two: Bits that correspond to the header and bits that correspond to the payload. Bits that correspond to the header are passed to the packet header generator that evaluates the payload metadata to calculate the header. Consequently, the output from both streams is converted to symbols and mapped according to QPSK modulation at both the header and payload channels. The output of the two streams is combined and passed through the carrier allocator where symbols are distributed in time and frequency. OFDM symbols are formulated in the FFT block, set to length 2048, and cyclic prefixes of length 512 are added after every symbol. Figure 5.9 shows the generated power spectrum of an OFDM signal with 2048-point FFT produced during transmission. This plot is in consonance with the theoretical spectrum plots for OFDM discussed in Chapter 2. The output complex values are forwarded to the USRP block *(USRP Sink)* that channels them to the USRP Tx. Accurate configuration of the MAC address on the USRP sink and source blocks establishes communication between the USRP devices. The centre frequency is set to 2.2 GHz on both devices to avoid interference with 2.4 GHz ambient Wireless Fidelity (Wi-Fi) signals. Thereafter the complex valued data is sent to the SU Rx.

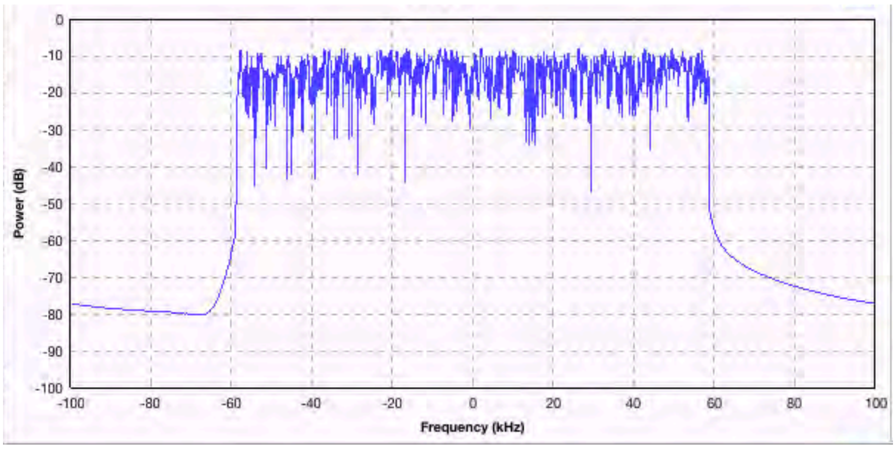


Figure 5.9: Power Spectrum of OFDM Transmitted Signal

5.2.2 Secondary User Implementation in GRC

The effect of channel noise on the transmitted signal is shown by the received signal in Figure 5.10. This signal is then passed through the real-time cognitive sensor modelled in GRC as shown in Figure 5.11. As seen from the flowgraph the USRP source block receives a stream of complex numbers that are grouped into 2048 values by the stream to vector block. These are then sent to the FFT module for IFFT computation. Parameters configured in the FFT block include the FFT size and the windowing technique selection. The Blackman-Harris technique outperforms Hanning, Hamming, and Bartlett windowing techniques, hence it is selected for implementation. This is due to its high side lobe rejection and moderate wide main lobe [102], [103]. The output complex numbers from the FFT block are then converted to a stream of real floating points, by the complex to float block for further computation. The *gr_modtool* script is modified to produce *square* and *average blocks*, OOTs that square the input stream of floating points and compute the mean of the 2048 float samples respectively. The resultant test statistic is forwarded to the decision *threshold block* (that is also modified from the *gr_modtool script*), where it is compared to a set threshold. Consequently, the file sink block stores the output from the threshold block to a binary *.dat* file.

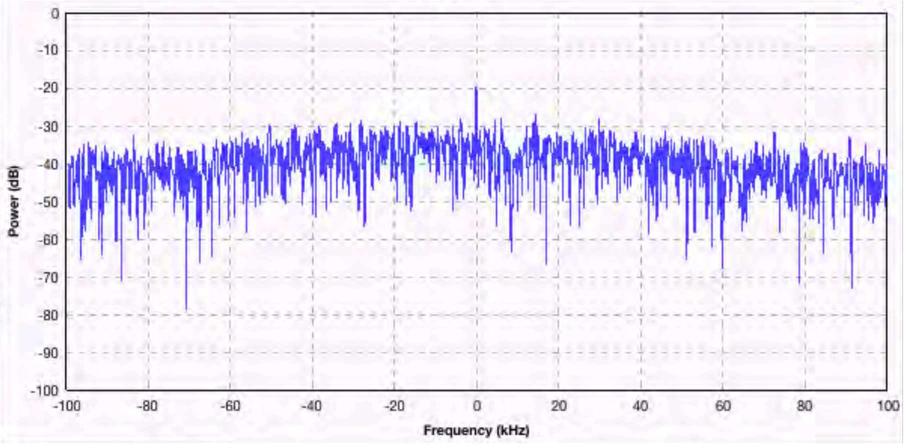
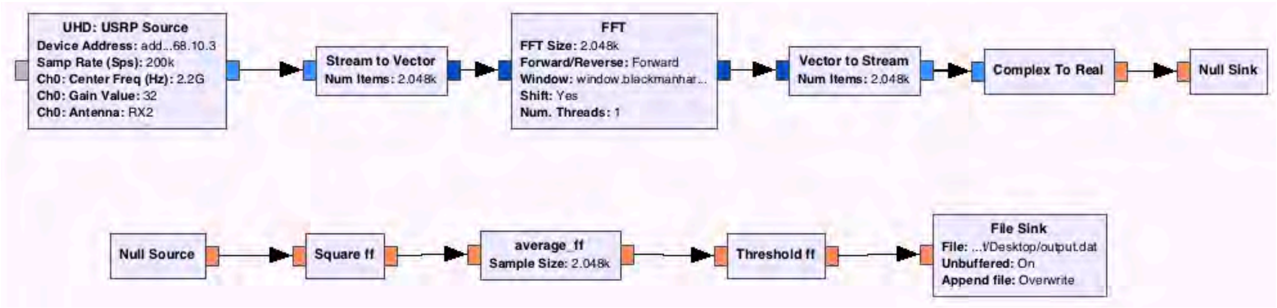


Figure 5.10: Power Spectrum of Received OFDM Signal



Device Address: add...68.10.3
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 32
Ch0: Antenna: RX2

Num Items: 2.048k

FFT Size: 2.048k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Num Items: 2.048k

Sample Size: 2.048k

File: ...t/Desktop/output.dat
Unbuffered: On
Append file: Overwrite

Figure 5.11: Spectrum Sensing Flowgraph at Secondary User Receiver

5.2.2.1 Average Block Implementation

Three main files are used to implement the average block in GRC, these are: *average_ff_impl.cc*, *average_ff_impl.h* and the *howto_ave_swig.i* files. The swig file contains the instructions that are required to convert the C++ code to python, whilst *average_ff_impl.h* is the header file that stores all the declared private and public variables [16]. The *Average_ff_impl.cc* is the source file, which consists of the private constructor, void and int.

The private constructor shown in Figure 5.12 has the *gr::sync_decimator* function obtained from the *gr::sync_block*. The *gr::sync_decimator* constructs the *N*:1 block. Where *N* is the number of input items that are used to produce one output item. The average block has one input that receives *N* floating points and one output that sends out floating points.

```
/*
    * The private constructor,
    */
   average_ff_impl::average_ff_impl(unsigned N)
     : gr::sync_decimator("average_ff",
            gr::io_signature::make(1, 1, sizeof(float)), //input
            gr::io_signature::make(1, 1, sizeof(float)),
             N) //output
   {}
```

Figure 5.12: Average block private constructor

The *work()* function in Figure 5.13 processes the *N* items and outputs a single item. It does this with the aid of the scheduler that continuously calls the work function as long as there is input data, and the output buffer space that matches *N*. Hence, the block does effective decimation by *N*; the input number of samples is a multiple of *N* and the block decimates it by *N*. *N* is dynamically changed to enable several tests to be done, and hence, it is configured as '*set_N*'. Consequently, the for-loop sends the output from the digital signal processing to the output buffer.

```
int
    average_ff_impl::work(int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items)
    {
     const float* in = (const float*) input_items[0];
     float* out = (float*) output_items[0];

     //declare variables
     double sum = 0.0;
     unsigned set_N = decimation();

     //digital signal processing
     //begin for loop from 1 to set_N

     for (unsigned i = 0; i < set_N; ++i) {
       sum += (*in++);
     }
     //calculate average
     out[0] = sum / ((double) set_N);
     //consume_each (noutput_items);
     return 1;
    }
```

Figure 5.13: Average block work function

To ensure that the GRC module configuration parameters are set correctly, the XML file in the *grc* directory is edited as shown in Appendix E.1. Thereafter running the *cmake* script (GnuradioConfig.cmake) in the build directory locates the libraries, header files, and dependencies used by GNU Radio [100]. The *Cmake* command shown in Appendix E.2 also connects the GNU Radio version with adequate requirements for operation [100]. After the *cmake* command, the *make*, *make test* and *sudo make install* commands are executed. These start the building process, test the code and install the module into GRC.

5.2.2.2 Threshold Block Implementation

The *threshold_ff_impl.h*, *threshold_ff_impl.cc* and *howto_thres_swig.i* are the three main files used to implement the threshold block. The threshold block has 1 input float item that produces 1 output float item. This is shown Figure 5.14 of the *gr::block* private constructor for *threshold_ff_impl.cc*.

```
/*
    * The private constructor
    */
   threshold_ff_impl::threshold_ff_impl()
     : gr::block("threshold_ff",
             gr::io_signature::make(1, 1, sizeof(float)),
             gr::io_signature::make(1, 1, sizeof(float)))
   {}
```

Figure 5.14: Threshold block private constructor

The use of the *forecast()* and *consume()* functions enable blocks with variable rates to be constructed. The defined forecast function informs the scheduler of the ratio of inputs to the output. The output rate is therefore a function of the input rate N, hence the relationship N:1. As a result, blocks that consume data at different rates can be designed. The threshold block algorithm as shown in *threshold_impl.cc* has one input port, denoted by ‘[0]’, as seen from the void function in Figure 5.15.

```
void
   threshold_ff_impl::forecast (int noutput_items, gr_vector_int &ninput_items_required)
   {
      ninput_items_required[0] = noutput_items;
   }
```

Figure 5.15: Threshold block void function

The *general_work()* function as shown in Appendix F is derived from the *gr::block* and implements the default 1:1 block that does the digital signal processing. To execute the *general_work()* function, the buffer is first filled and the operation initiated by the scheduler. In the threshold block, the input energy test statistic is compared to a predetermined threshold with set parameters. These are the number of items N, and the probability of false alarm P_{fa}. The decision made is passed out as either a 1 or 0. This is followed by running the *cmake, make, make test* and *sudo make* install commands as stated for the average block, to install the threshold block in the GRC library.

5.3 Chapter Summary

The experimental setup for spectrum sensing using GNU Radio and USRP was clearly presented. The equipment and software used were outlined in the system design, and the testbed modelled using the same parameters as the simulations. Software modifications done to implement the square, addition and threshold blocks were discussed, and explanations given as to how the blocks satisfy the energy sensor implementation in real-time. The next chapter shows the simulations that were done using MATLAB. The results obtained serve as the benchmark for the real-time implementation.

Chapter 6

6 Experimental Results and Analysis

This chapter validates the experimental performance of the spectrum sensing algorithms. To do this, the methodological overview and the experimental determination procedures for threshold and SNR are presented. The experiments carried out are then explained in detail, and the results obtained compared to the MATLAB simulation results. Probability of detection, probability of false alarm, and the probability of misdetection are the metrics for evaluation. The same parameters presented in Table 4.1 for simulations are repeated for the real-time implementation.

6.1 Overview of Methodology

The primary user and secondary user are modelled in GNU Radio Companion as shown in Figure 6.1. An OFDM signal with QPSK modulation is considered as a PU transmitted signal. The energy detection algorithms on the SU detect the transmitted PU signal. To generate the test signal the flowgraph is executed for 30 seconds. During the first 15 seconds, the GRC initialises and in the following 15 seconds, samples are generated.

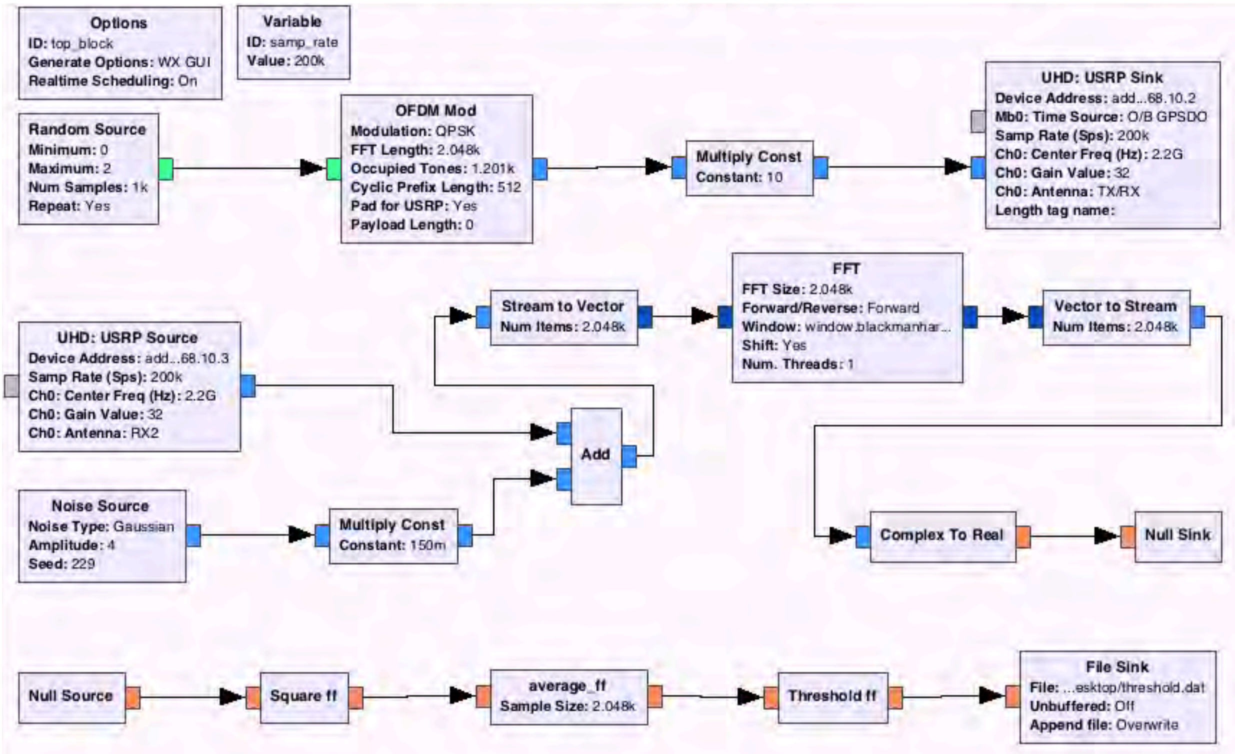
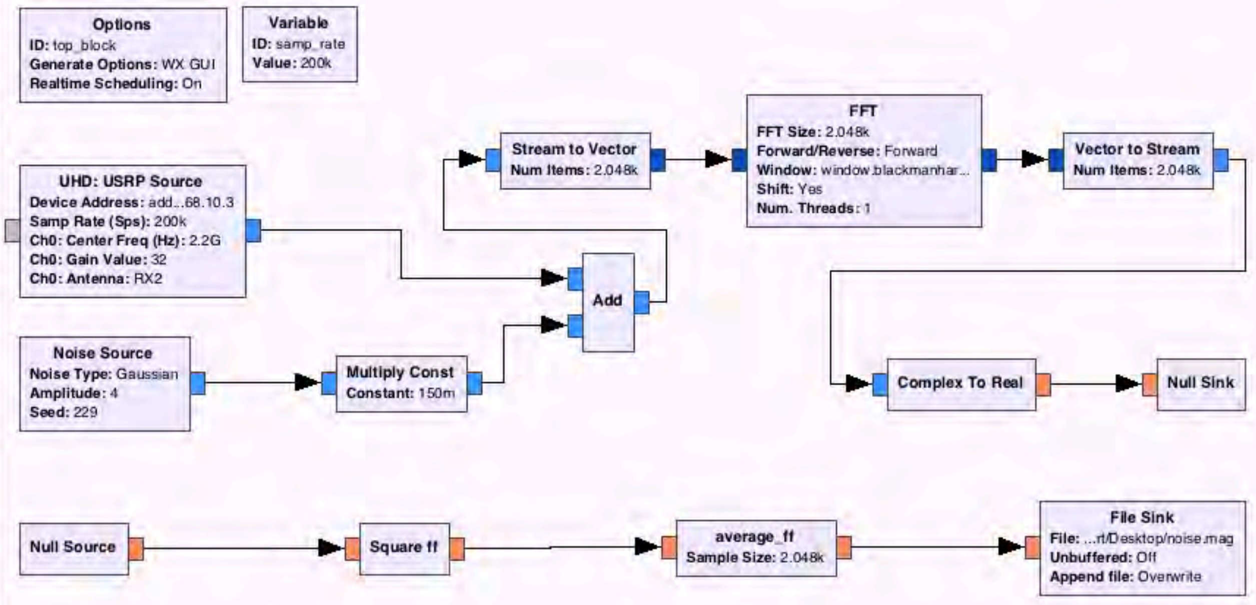


Figure 6.1: Primary User and Secondary User Flowgraph

6.2 Experimental Determination of Signal-to-Noise Ratio

The USRP is not a calibrated device, therefore the SNR is estimated. To determine the signal-to-noise ratio, the receiving USRP is connected to the PC and the flowgraph in Figure 6.2, executed and allowed to run for 30 seconds.


Options
ID: top_block
Generate Options: WX GUI
Realtime Scheduling: On

Variable
ID: samp_rate
Value: 200k

UHD: USRP Source
Device Address: add...68.10.3
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 32
Ch0: Antenna: RX2

Noise Source
Noise Type: Gaussian
Amplitude: 4
Seed: 229

Multiply Const
Constant: 150m

Stream to Vector
Num Items: 2.048k

FFT
FFT Size: 2.048k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

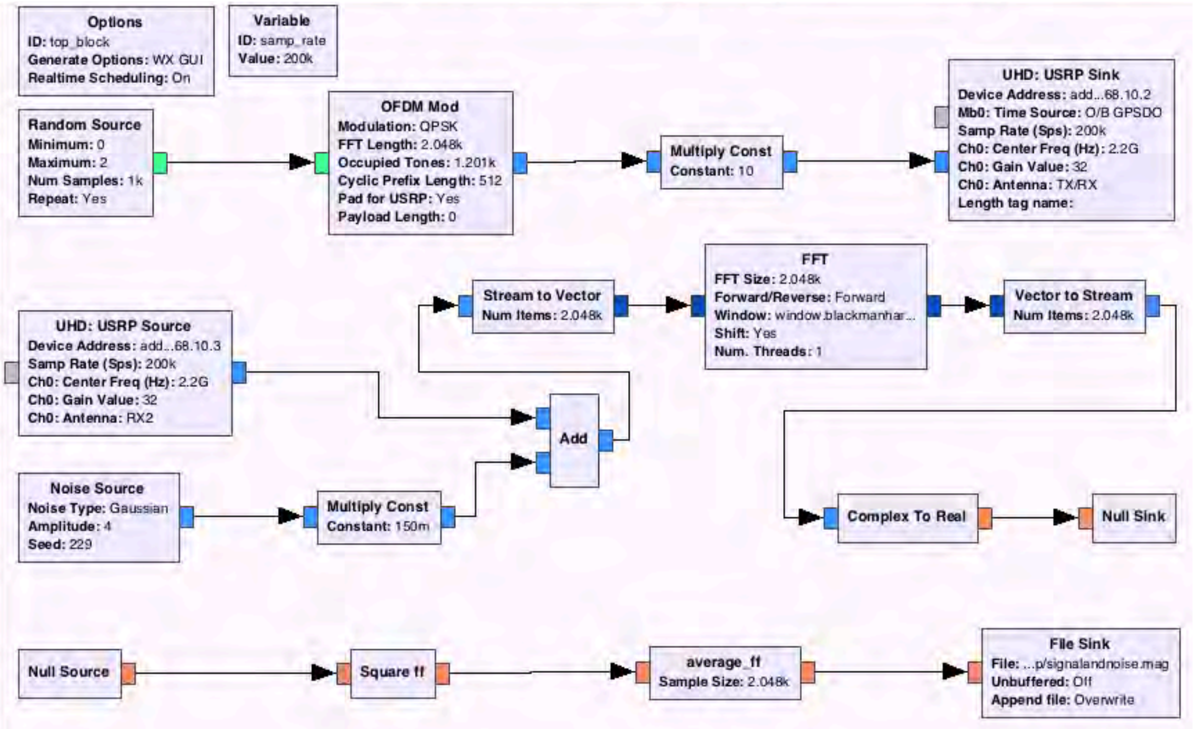
Vector to Stream
Num Items: 2.048k

average_ff
Sample Size: 2.048k

File Sink
File: ...rt/Desktop/noise.mag
Unbuffered: Off
Append file: Overwrite

Figure 6.2: Block Diagram in GRC used for SNR Estimation

The flowgraph generates energy samples from the received noise signal, which are then saved to a file (noise.mag) by the file sink block. The noise.mag file is opened in Octave, and the average of 1000 noise energy samples calculated and recorded. The same procedure is repeated adding the PU transmitter signal to calculate the test statistic. The GRC flowgraph is shown in Figure 6.3.



Options
ID: top_block
Generate Options: WX GUI
Realtime Scheduling: On

Variable
ID: samp_rate
Value: 200k

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

OFDM Mod
Modulation: QPSK
FFT Length: 2.048k
Occupied Tones: 1.201k
Cyclic Prefix Length: 512
Pad for USRP: Yes
Payload Length: 0

Multiply Const
Constant: 10

UHD: USRP Sink
Device Address: add...68.10.2
Mb0: Time Source: O/B GPSDO
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 32
Ch0: Antenna: TX/RX
Length tag name:

UHD: USRP Source
Device Address: add...68.10.3
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 32
Ch0: Antenna: RX2

Stream to Vector
Num Items: 2.048k

FFT
FFT Size: 2.048k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Vector to Stream
Num Items: 2.048k

Noise Source
Noise Type: Gaussian
Amplitude: 4
Seed: 229

Multiply Const
Constant: 150m

average_ff
Sample Size: 2.048k

File Sink
File: ...p/signalandnoise.mag
Unbuffered: Off
Append file: Overwrite

Figure 6.3: GRC Modules used to Generate the Received Signal

To estimate SNR the average noise power (P_{noise}), and average signal were calculated with noise power ($P_{signal+noise}$). To obtain various signal strengths the amplitude of the noise source block is varied. The linear SNR is calculated and converted to the logarithmic scale as shown below:

$$SNR_{dB} = 10\ log_{10}(\frac{P_{signal+noise}-P_{noise}}{P_{noise}}) \qquad (6.1)$$

6.3 Experimental Determination of Threshold

The GNU Radio flowgraph shown in Figure 6.2 is used to determine the noise threshold. The flowgraph is run for 30 seconds and noise energy samples saved to a file (output.mag). Using Octave, 1000 samples are read and saved onto a text file for further processing in Microsoft Excel. The Empirical Cumulative Distribution Function (CDF) of the noise power is plotted as shown in Figure 6.4. The plot is a function of threshold versus probability (1 - P_{fa}). The threshold is determined by extrapolating at fixed desired probability of false alarm.

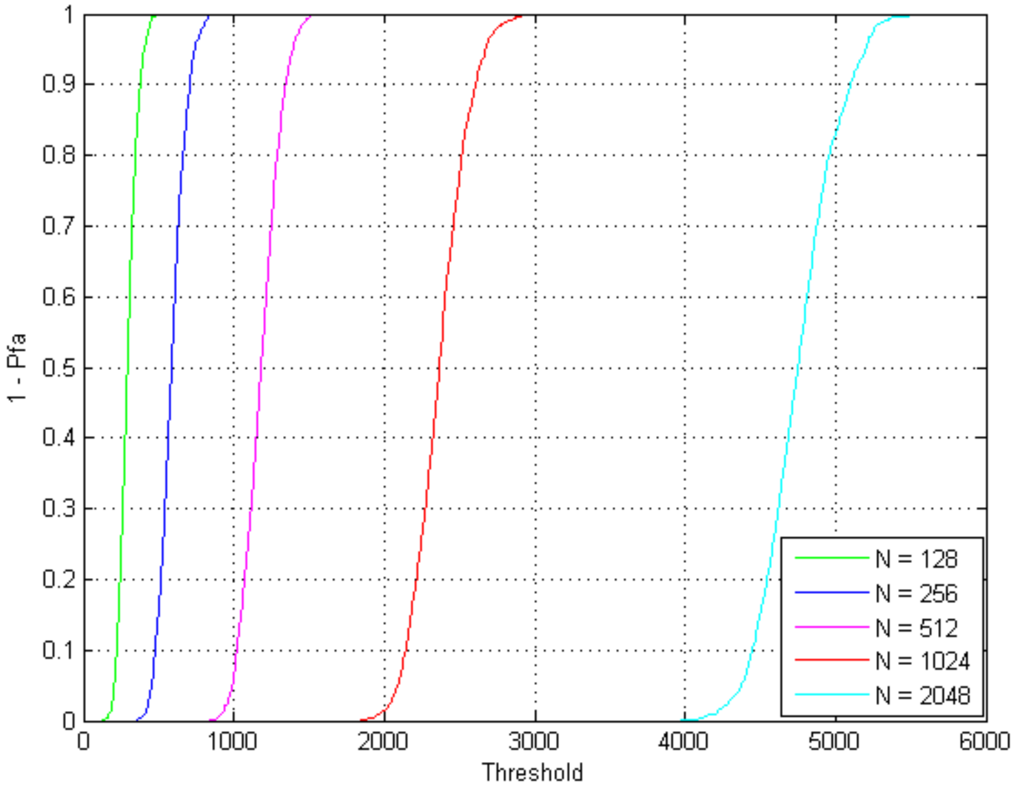


Figure 6.4: Noise Signal Empirical CDF for varying Sample Sizes (N = 128, 256, 512, 1024 and 2048).

Figure 6.5 shows a plot of noise with threshold deployed in this work. The deployed constant threshold has numerous drawbacks when in an environment where noise fluctuates rapidly, as discussed in Section 2.5. In this work, experiments were done in a laboratory environment with minimum interference.

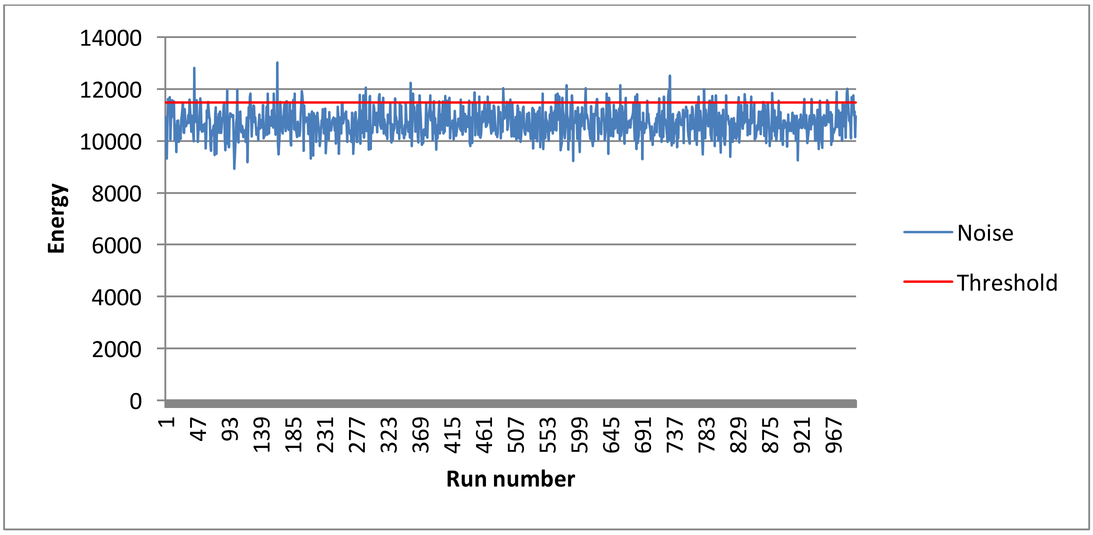


Figure 6.5: Plot of Noise Variance with Threshold for N=2048 at -15 dB

6.4 Effect of Sample Size on Probability of Detection

In this experiment, the SNR and threshold are estimated as discussed in Section 6.3 and continued in Section 6.4. The amplitude of the noise module on the GRC is varied from 4 to 50, producing a range of SNRs from -22 dB to 0 dB. The threshold is set at a probability of false alarm of 0.1, in accordance with the IEEE 802.22 WRAN standard. The probability of detection of a PU in a single test is determined from 1000 decisions that are produced on a single run of the flowgraph. For better estimation, the experiment is repeated 10 times and the average probability of detection determined. The same procedure is repeated for the specified range of SNRs.

6.4.1 Results

Figure 6.6 shows a plot of the SNR versus the probability of detection, with varying sample sizes (N = 128, 256, 512, 1024, and 2048). The plotted results show a direct proportionality of the SNR and probability of detection, which is a similar trend to the simulation results discussed in section 4.3. Therefore, increasing the sample size increases the probability of detection.

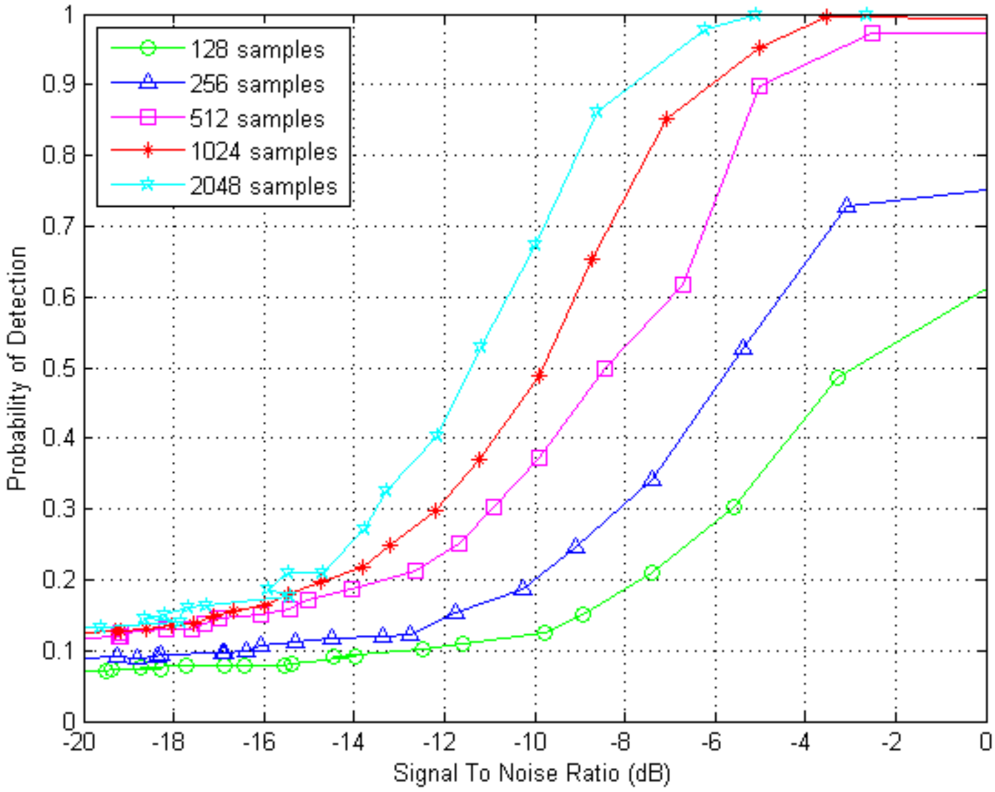


Figure 6.6: Signal-to-Noise Ratio versus Probability of Detection with varying Sample Size ($P_{fa} = 0.1$)

6.4.2 Validation

Table 6.1 shows the comparison of probability of detection with various sample sizes between the simulated and real-time experiment at -8 dB. From the table, the simulation attains the desired probability of detection ($P_d \geq 0.9$) when N = 512, whereas the real-time sensor requires a sample size of at least N = 2048 to achieve the required probability of detection.

Table 6.1: Performance comparison of simulation with implementation at -8 dB

Sample Size ($\boldsymbol{N}$)	128	256	512	1024	2048
Simulated ($\boldsymbol{P_d}$)	0.62	0.80	0.93	0.99	1.00
Real-Time ($\boldsymbol{P_d}$)	0.18	0.30	0.53	0.63	0.90

This shows that the real-time system cannot attain the IEEE 802.22 WRAN standard's favourable probability of detection at sample sizes lower than 2048. Lower sample sizes attain lower probability of detection as compared to the simulation results. This is accounted to the channel noise and thermal noise in the USRP.

6.5 Effect of Signal-to-Noise Ratio on Probability of Detection with varying Probability of False Alarm

The SNR estimation is carried out by varying the level of noise with even numbered amplitudes from 4 to 50, producing an SNR range of -22 dB to 0 dB. The output from the flowgraph is saved to a file (threshold.dat) as a stream of 1's and 0's, which are used to predict the presence or absence of the primary user respectively. The flowgraph in Figure 6.2 is run 1000 times to determine the probability of detection. The experiment is repeated 10 times for each SNR to remove the effect of noise variation. The whole procedure is then repeated for each probability of false alarm value under investigation.

6.5.1 Results

Figure 6.7 shows a plot of SNR versus probability of detection with varying probabilty of false alarm (P_{fa} = 0.01, 0.1 and 0.2) at N = 2048. MATLAB is used to make a graphical presentation of the obtained data as seen in the plot. The experimental result shows that as the probability of false alarm increases from 0.01 to 0.2, the probability of detection also increases, resembling a similar trend to the simulation results in Figure 4.2.

At high SNRs the effect of probability of false alarm is minimal as the sensor can achieve a very high probability of detection. This is observed at SNR = -6 dB, where the probability of detection for the probability of false alarm values under consideration is at least 0.9. As the SNR decreases the effect of the various probability of false alarm values becomes easily distinguished. For instance at SNR = -12 dB, the probability of detection is 0.16 for the probability of false alarm of 0.01 and 0.63, for probability of false alarm of 0.2.

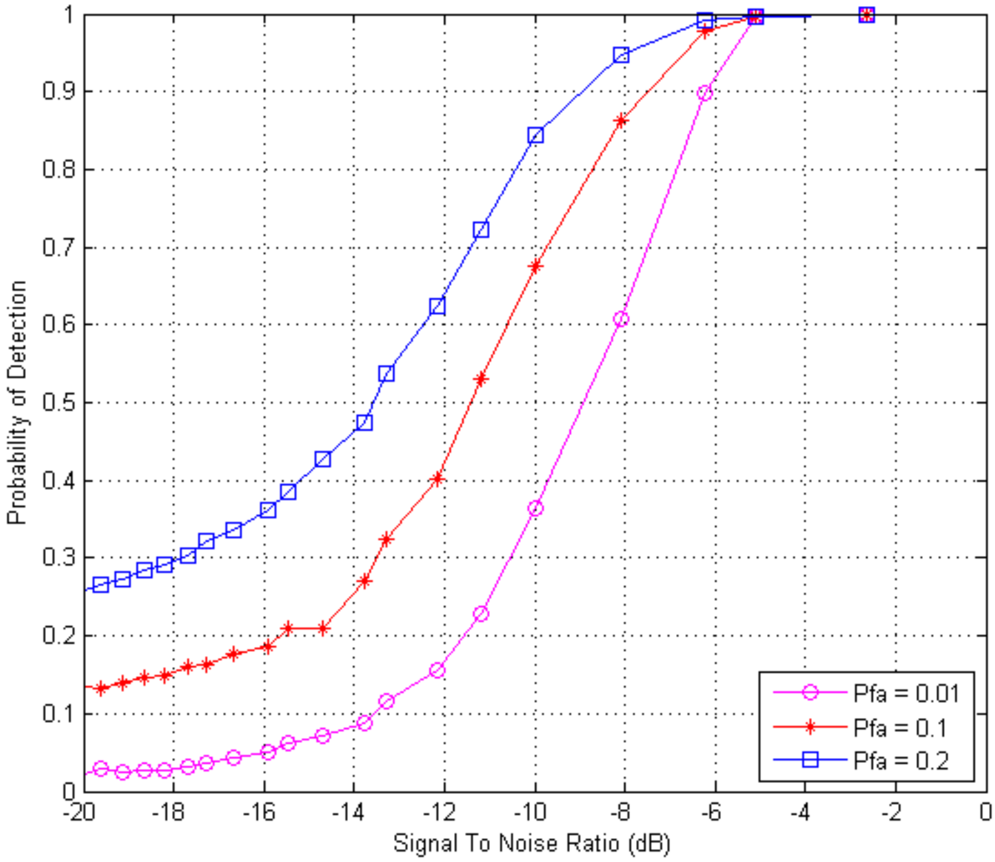


Figure 6.7: Signal-to-Noise Ratio versus Probability of Detection with varying Probability of False Alarm

6.5.2 Validation

Table 6.2 shows a comparison of the simulated sensor with the real-time sensor at -13 dB. From the results, it is evident that the real-time performance is lower compared to the simulated performance. However for both scenarios, as the probability of false alarm increases, the probability of detection also increases at a constant SNR. The differences between the two can also be accounted to the imperfect AWGN channel used on the testbed, and thermal noise in the receiver.

Table 6.2: Performance comparison of simulation versus implementation at SNR = -13 dB

Probability of False Alarm (P_{fa})	0.01	0.1	0.2
Simulated (P_d)	0.48	0.71	0.81
Real-Time (P_d)	0.13	0.34	0.56

6.6 Receiver Operating Characteristics

The performance of the energy detector is analysed using ROC curves in this experiment. This is a plot of probability of false alarm versus probability of detection at varying signal-to-noise ratios as discussed in Section 4.5. The same parameters used for simulations are used on the testbed, thus creating a benchmark for the real-time implementation. Each probability of false alarm has a respective threshold that is determined using the empirical ICDF method, with noise samples generated by the GNU Radio flowgraph shown in Figure 6.3. As in the previous experiments, the SNR estimation procedure used is explained in Section 6.2. The experiment is carried out for a range of signal-to-noise ratios (SNR = -11 dB, -15 dB, -17 dB and -22 dB). The flowgraph given in Figure 6.1 generates 1000 decisions that are used to determine the probability of detection. To eliminate noise uncertainty this is repeated 10 times and the average probability of detection is calculated. The same experimental procedure is repeated for each of the selected SNRs and the results discussed as follows.

6.6.1 Results

Figure 6.8 shows the ROC curves for the energy detector on the real-time testbed, for varying SNRs. The area under the ROC curves has a direct proportionality with the performance of the energy detector and this is used to derive the effectiveness of the real-time sensor. From the obtained curves it is observed that as the SNR increases the area under the curve (AUC) also increases. This therefore implies that the real-time energy sensor performs more efficiently as the SNR increases from -22 dB to -11 dB. Curves with an area close to 1 square unit, have a higher probability of detecting the PU signal, whereas, curves with an area close to 0.5 square units have an inferior probability of detecting the PU, hence lower performance.

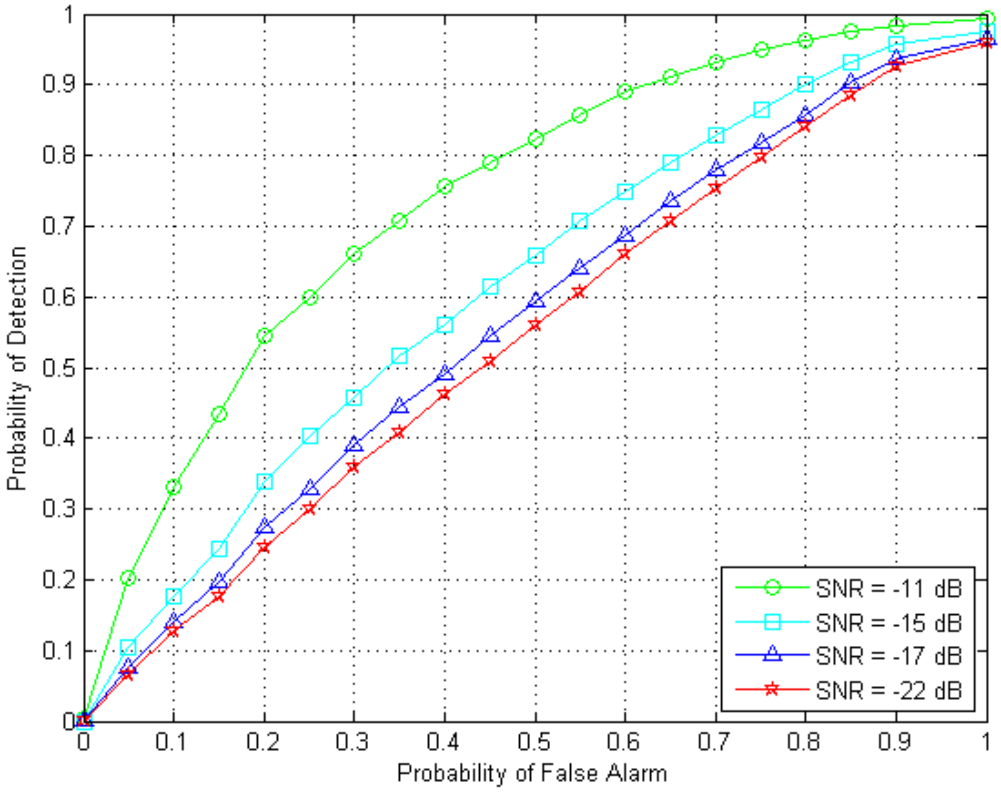


Figure 6.8: Energy Detector ROC curves at different Signal-to-Noise Ratios

6.6.2 Validation

A comparison of the real-time ROC curves with the simulated ROC curves presented in Figure 4.3 shows a similar trend between the two. Both results show a correlation between SNR and the area under the curve. As the SNR increases so does the area under the curve. Thus implying that the real-time sensor performs as expected. However, there is a degradation in performance of the real-time sensor compared to the simulated sensor.

6.7 Effect of Complementary ROC Curves with Signal-to-Noise Ratio

The same methodology used in Section 6.6 is repeated for this experiment. The only difference is in determining the probability of misdetection (P_{md}), which is defined as:

$$P_{md} = 1 - P_d, \tag{6.2}$$

6.7.1 Results

Figure 6.9 shows a plot of probability of false alarm against the probability of misdetection for varying SNRs. For the Complementary Receiver Operating Characteristics (CROC) curves, as the AUC decreases, the performance of the detector increases, which is the opposite to that of the ROC curves. The AUC represents the probability of misdetection. The plot shows that increasing the SNR decreases the probability of misdetection and vice versa. Hence, the performance of the energy detector degrades as the SNR decreases and as the probability of false alarm increases.

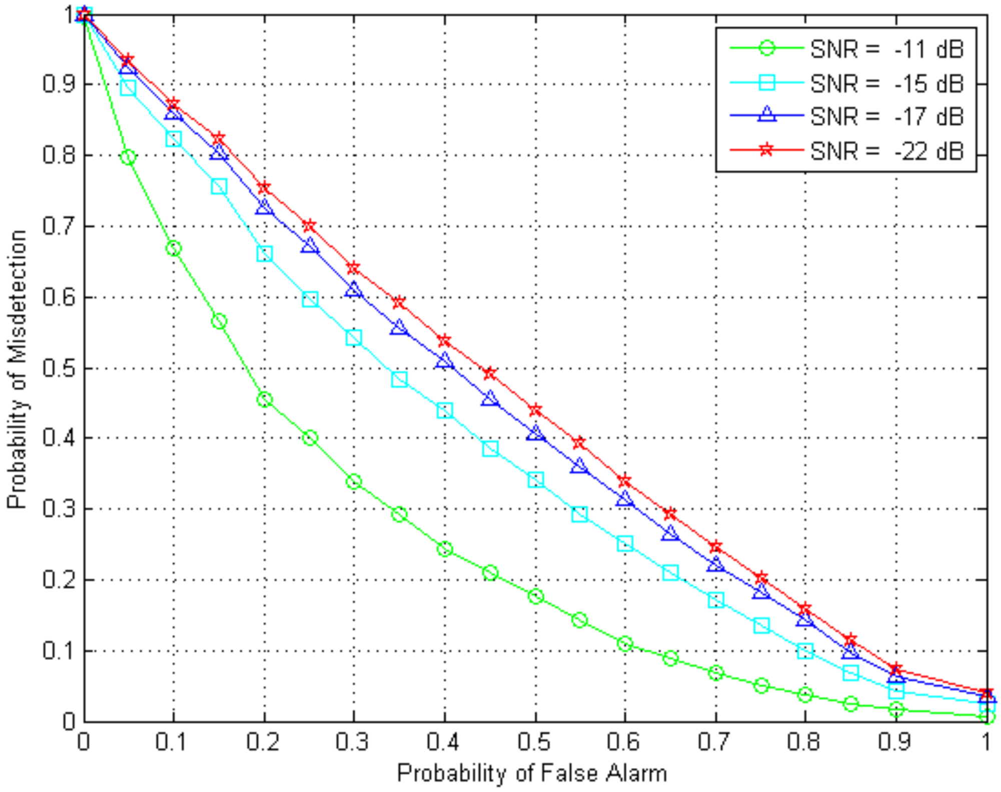


Figure 6.9: The CROC with varying Signal-to-Noise Ratio

6.7.2 Validation

The simulations in Figure 4.4 show a similar pattern to the real-time results shown in Figure 6.9. As the probability of false alarm increases, the probability of misdetection decreases, implying that the real-time sensor performs as expected. However, the real-time implementation plots have a greater area under the curves, which implies a higher probability of misdetection.

6.8 Chapter Summary

The results obtained from the real-time implementation of the energy detector using USRP and GNU Radio platforms are presented. The curves obtained via real-time implementation have been validated, as they show a similar trend with those obtained from the Monte-Carlo simulations. It is noted that the performance of the sensor can be improved by increasing the number of samples. This has the effect of eliminating the channel noise when the test statistic is being calculated, which enhances the SNR. In addition, the results show that the probability of detection has to be maximized inorder to achieve high performance; conversely, the probability of false alarm has to be minimised.

Comparisons of the real-time energy sensor to simulations show that the implemented cognitive sensor can detect signals with low SNRs up to about -8 dB. However, the performance can be improved by using a more accurate threshold selection technique. Generally the performance of the implemented real-time energy detector compared to simulations was lower, this can be accounted to the distribution of the noise being not perfectly AWGN.

Chapter 7

7 Conclusion and Recommendations

In this Chapter a summary of the dissertation is given. The main findings are highlighted and conclusions drawn. Afterwards the real-time spectrum-sensor implementation issues are discussed. Lastly further works are given.

7.1 Conclusion

This dissertation highlighted the spectrum shortage challenge brought about by the increasing number of wireless devices. Spectrum underutilization was noted to be the primary cause of this dilemma. The proposed solution to spectrum inefficiency, the cognitive radio, was therefore investigated. It was noted that spectrum sensing algorithms are key to the performance of cognitive radios, however, most of the algorithms developed so far have not been tested in real-time; hence the need to evaluate their feasibility.

This work therefore explored the practical approach to spectrum sensing. The energy based detector spectrum sensing technique was chosen for implementation. To establish a benchmark, the energy detector's algorithm performance was first evaluated using simulations with the MATLAB tool. The real-time system was realized on a SDR platform, built using USRP N210 kits from Ettus Research, with the energy sensor implemented on a personal computer using open source GNU Radio Companion. Existing modules in GRC were analysed and modified to that effect. The created OOT modules include the square, average and threshold blocks. Furthermore, the real-time testbed was modelled using the same parameters as the simulations, with a QPSK-modulated OFDM test signal. Comparisons of the simulations and real-time results were used to validate the real-time energy detector.

From the simulation results, it was observed that the energy detector can detect signals as low as -12 dB at the desired probability of detection of 0.9, and probability of false alarm at 0.1, at a sample size of 2048. With the same parameters, the lowest experimental real-time SNR that could be detected was observed to be -8 dB. This showed that the real-time sensor was less efficient by 4 dB. In addition, the analysis of the real-time and simulated sensor proved that the sample size affected the performance of the detector, thus implying a correlation between sample size and the energy detector's performance. The real-time results proved that the lowest sample size that could be used optimally at the desired P_d and P_{fa} values was 2048. In addition, the results obtained showed that for the energy detector to achieve high probability of detection, the probability of false alarm should be minimised.

The performance of the real-time system was inferior to that predicted by the simulations. This was attributed to the fact that the channel used for the real-time system was not perfectly AWGN. In addition, thermal noise from the USRP hardware also contributed to this disparity. Also, the threshold setting accuracy and SNR estimation accuracy were affected at low SNR, due to the high noise variance.

In conclusion, the SDR is one of the wireless communications emerging technologies and can be used to implement cognitive radio detection using USRP and the GNU Radio platform. The real-time results obtained are proof-of-concept for the testbed that was built. In addition, the difference between the energy sensor simulations and testbed results show the disparity in their performances, which highlights the limitation of hardware used.

7.2 Implementation Issues

The major complicating factors encountered during the implementation of the real-time testbed are listed as follows:

- Threshold estimation was affected greatly by the increased noise variance in low SNRs. To mitigate this, the adaptive threshold technique was adopted in the implemention of the real-time sensor as it is robust and can withstand the rapid noise fluctuation.
- The performance of the energy detector degraded rapidly at SNRs below the SNR_{wall}. This was attributed to the high noise power in the channel. No mitigation technique was found for this over the duration of the project.

- Real-time cognitive radios require high sampling rates, high resolution analog-to-digital convertors with large dynamic range, and high speed signal processors.

7.3 Further Work

It would be interesting to find out the performance of other detection techniques on the implemented real-time platform. In addition, the developed testbed however is not limited to sensing algorithms; Researchers can use the platform to sense multiple bands instead of a single band.

References

[1] S. Mangold, Z. Z. Z. Zhong, K. Challapali, and C.-T. C. C.-T. Chou, “Spectrum agile radio: radio resource measurements for opportunistic spectrum usage,” in *IEEE Global Telecommunications Conference, 2004. GLOBECOM ’04.*, 2004, vol. 6, pp. 3467–3471.

[2] D. T. Otermat, C. E. Otero, and I. Kostanic, “Analysis of the FM Radio Spectrum for Internet of Things Opportunistic Access Via Cognitive Radio,” 2015.

[3] A. S. Saini, H. Zhen, and R. Qiu, “Spectrum sensing and reconstruction for cognitive radio,” in *System Theory, 2009. SSST 2009. 41st Southeastern Symposium on*, 2009, pp. 13–18.

[4] L. Yang, W. Hou, L. Cao, B. Y. Zhao, and H. Zheng, “Supporting Demanding Wireless Applications with Frequency-agile Radios,” in *7th USENIX conference on Networked systems design and implementation*, 2010, pp. 1–15.

[5] Federal Communication Council, “Cognitive Radios.” [Online]. Available: www.eecs.berkeley.edu/~dtse/3r_Cognitive_radio_FCC-03-322A1.doc. [Accessed: 26-Nov-2015].

[6] I. F. Akyildiz, B. F. Lo, and R. Balakrishnan, “Cooperative spectrum sensing in cognitive radio networks: A survey,” *Phys. Commun.*, vol. 4, no. 1, pp. 40–62, 2011.

[7] S. D. Barnes, P. A. Jansen Van Vuuren, and B. T. Maharaj, “Spectrum occupancy investigation: Measurements in South Africa,” *Meas. J. Int. Meas. Confed.*, vol. 46, no. 9, pp. 3098–3112, 2013.

[8] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, “NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey,” *Comput. Networks*, vol. 50, no. 13, pp. 2127–2159, 2006.

[9] J. Wang, M. Ghosh, and K. Challapali, “Emerging cognitive radio applications: A survey,” *IEEE Communications Magazine*, vol. 49, no. 3, pp. 74–81, 2011.

[10] T. Yucek and H. Arslam, “A Survey of Spectrum Sensing Algorithms for Congnitive Radio Applications,” *IEEE Commun. Surv. Tutorials*, vol. 97, no. 5, pp. 805–823, 2009.

[11] A. Ghasemi and E. Sousa, “Spectrum sensing in cognitive radio networks: requirements,

challenges and design trade-offs," *Communications Magazine, IEEE*, vol. 46, no. 4, pp. 32–39, 2008.

[12] S. Srinu, "Entropy based Reliable Cooperative Spectrum Sensing for Cognitive Radio Networks," Doctoral dissertation, University of Hyderabad, India, 2013.

[13] F. Salahdine, W. F. Fihri, H. El Ghazi, and N. Kaabouch, "Matched Filter Detection with Dynamic Threshold for Cognitive Radio Networks," in *Wireless Networks and Mobile Communications (WINCOM)*, 2015, no. January 2016, pp. 1–6.

[14] H. Liu, Y. Liu, and Y. Huo, "Cyclic Stepping Spectrum Sensing based on Energy Detection," in *International Conference on Mechatronic Science, Electric Engineering and Computer*, 2011, vol. 1, no. 1, pp. 486–489.

[15] F. M. Salem, M. H. Ibrahim, I. A. Ali, and I. I. Ibrahim, "Matched-Filter-based Spectrum Sensing for Secure Cognitive Radio Network Communications," *Int. J. Comput. …*, vol. 87, no. 18, pp. 41–46, 2014.

[16] D. Nguyen, "Implementation of OFDM systems using GNU Radio and USRP," Masters dissertation, University of Wollongong, Australia, 2013.

[17] D. D. Ariananda, M. K. Lakshmanan, and H. Nikoo, "A survey on spectrum sensing techniques for Cognitive Radio," in *2009 Second International Workshop on Cognitive Radio and Advanced Spectrum Management*, 2009, pp. 74–79.

[18] S. V. Nagaraj, "Entropy-based spectrum sensing in cognitive radio," *Signal Processing*, vol. 89, no. 2, pp. 174–180, 2009.

[19] A. Nafkha, M. Naoues, K. Cichon, and A. Kliks, "Experimental Spectrum Sensing Measurements using USRP Software Radio Platform and GNU-Radio," in *9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM)*, 2014, pp. 1–6.

[20] R. Rashid, M. Sarijari, and N. Fisal, "Spectrum sensing measurement using GNU Radio and USRP software radio platform," in *ICWMC 2011, The …*, 2011, pp. 237–242.

[21] T. Zhang, G. Yu, and C. Sun, "Performance of cyclostationary features based spectrum sensing method in a multiple antenna cognitive radio system," *IEEE Wirel. Commun. Netw.*

Conf. 2009. WCNC 2009, pp. 1 – 5, 2009.

[22] M. Verma and R. Dua, “Performance Analysis Of Energy Detection, Matched Filter Detection & Cyclostationary Feature Detection Spectrum Sensing Techniques,” *Indian Streams Res. J.*, vol. 2, no. 5, pp. 1296–1301, 2012.

[23] I. F. Akyildiz, W.-Y. Lee, and K. R. Chowdhury, “CRAHNs: Cognitive radio ad hoc networks,” *Ad Hoc Networks*, vol. 7, no. 5, pp. 810–836, 2009.

[24] N. Thuo, “An Adaptive Threshold Energy Detection Technique with Noise Variance Estimation for Cognitive Radio Sensor Networks,” Masters dissertation, University of Cape Town, South Africa, 2015.

[25] S. Srinu and S. L. Sabat, “Spectrum Sensing for Cognitive Radio Networks.,” in *White Space Communication*, Springer International Publishing, 2014, pp. 117–151.

[26] S. Srinu and S. L. Sabat, “FPGA implementation and performance study of spectrum sensing based on entropy estimation using cyclic features,” *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1658–1669, 2012.

[27] R. K. Ganti, Z. Gong, M. Haenggi, C. Lee, S. Srinivasa, D. Tisza, S. Vanka, and P. Vizi, “Implementation and Experimental Results of Superposition Coding on Software Radio,” in *Communications (ICC), 2010 IEEE International Conference on*, 2010, pp. 1–5.

[28] M. T. Masonta, M. Mzyece, and N. Ntlatlapa, “Spectrum Decision in Cognitive Radio Networks: A Survey,” *Commun. Surv. Tutorials, IEEE*, vol. 15, no. 3, pp. 1088–1107, 2013.

[29] M. Abdulsattar and Z. Hussein, “Energy detection technique for spectrum sensing in cognitive radio: a survey,” *Int. J. Comput. …*, vol. 4, no. September, pp. 223–242, 2012.

[30] S. Chaudari, “Spectrum Sensing for Cognitive Radios: Algorithms, Performance, and Limitations,” Doctoral dissertation, Aalto University, Finland, 2012.

[31] S. Maharjan, K. Po, and J. Takada, “Energy Detector Prototype for Cognitive Radio System,” Japan, 2016.

[32] M. A. Matin, *Developments in Wireless Network Prototyping, Design, and Deployment: Future Generations*, 1st Editio. Hershey, PA: IGI Global, 2012.

[33] M. A. Sarijari, A. Marwanto, N. Fisal, S. K. S. Yusof, R. a. Rashid, and M. H. Satria,

"Energy detection sensing based on GNU radio and USRP: An analysis study," in *Proceedings - MICC 2009: 2009 IEEE 9th Malaysia International Conference on Communications with a Special Workshop on Digital TV Contents*, 2009, no. December, pp. 338–342.

[34] A. Rahman, S. Khadka, M. Gahadza, A. Haniz, M. Kim, and J. Takada, "Development of Spectrum Sensing System with GNU Radio and USRP to Detect Emergency Radios," Tokyo, 2009.

[35] Y. Peng, F. Liu, and S. Liu, "Active contours driven by normalized local image fitting energy," *Pattern Recognit.*, vol. 26, no. 43, pp. 1199–1206, 2010.

[36] W. A. Gardner, "Exploitation of spectral redundancy in cyclostationary signals," *IEEE Signal Processing Magazine*, vol. 8, no. 2. pp. 14–36, 1991.

[37] L. C. Tran, D. T. Nguyen, F. Safaei, and P. J. Vial, "An Experimental Study of OFDM in Software Defined Radio Systems Using GNU Platform and USRP2 Devices," in *2014 International Conference on Advanced Technologies for Communications (ATC 2014)*, 2014, pp. 657–662.

[38] D. Cabric, D. Cabric, A. Tkachenko, A. Tkachenko, R. W. Brodersen, and R. W. Brodersen, "Experimental study of spectrum sensing based on energy detection and network cooperation," *Proc. first Int. Work. Technol. policy Access. Spectr.*, no. Article no. 12, 2006.

[39] R. Tandra and A. Sahai, "Fundamental limits on detection in low SNR under noise uncertainty," in *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, 2005, vol. 1, pp. 464–469.

[40] D. Cabric and R. Brodersen, "Physical layer design issues unique to cognitive radio systems," in *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2005, pp. 759–763.

[41] A. Khandakar, "Performance Analysis in Cognitive Radio Networks Using USRP2 Practical Experiment," Masters dissertation, Qatar University, Doha, 2015.

[42] S. Atapattu, C. Tellambura, and H. Jiang, *Energy Detection for Spectrum Sensing in Cognitive Radio*. Springer International Publishing, 2014.

[43] D. Liu and J. Yu, “Otsu method and K-means,” in *Proceedings - 2009 9th International Conference on Hybrid Intelligent Systems, HIS 2009*, 2009, vol. 1, no. 2, pp. 344–349.

[44] G. Zhang, S. Chen, and J. Liao, “Otsu Image Segmention Algorithm Based on Morphology and Wavelet Transformation,” in *Computer Research and Development (ICCRD), 2011 3rd International Conference*, 2011, pp. 279–283.

[45] Q. Wang, H. Zhang, Q. Dong, Q. Niu, G. Xu, and Y. Xue, “Otsu thresholding segmentation algorithm based on Markov Random Field,” in *2011 Seventh International Conference on Natural Computation*, 2011, vol. 2, pp. 969–972.

[46] S. V. Vaseghi, *Advanced Digital Signal Processing and Noise Reduction*, 3rd Editio. England: John Wiley & Sons, Ltd, 2005.

[47] S. O. Rice, “Mathematical Analysis of Random Noise,” *Bell Syst. Tech. J.*, vol. vol. 23, no. July, pp. 282–332, 1944.

[48] D. Datla, A. Wyglinski, and G. Minden, “A Statistical Approach to Spectrum Measurement Processing,” *Inf. Telecommun. Technol. Cent.*, 2005.

[49] W. Duma, “Impementation of a Testbed for MISO OFDM Communication Systems,” Masters dissertation, University of Kwazulu Natal, South Africa, 2012.

[50] A. Marwanto, M. A. Sarijari, N. Fisal, S. K. S. Yusof, and R. a. Rashid, “Experimental study of OFDM implementation utilizing GNU radio and USRP - SDR,” in *Proceedings - MICC 2009: 2009 IEEE 9th Malaysia International Conference on Communications with a Special Workshop on Digital TV Contents*, 2009, no. December, pp. 132–135.

[51] C. Sonntag, “Orthogonal Frequency Division Multiplexing (OFDM) implementation as part of a Software Defined Radio (SDR) environment,” Masters dissertation, University of Stellenbosch, South Africa, 2005.

[52] H. Kim, J. Kim, S. Yang, M. Hong, M. Yoo, W. Lee, and Y. Shin, “An effective MIMO-OFDM transmission scheme for IEEE 802.22 WRAN systems,” *Proc. 2nd Int. Conf. Cogn. Radio Oriented Wirel. Networks Commun. CrownCom*, vol. 55, no. 8, pp. 394–399, 2007.

[53] C. L. H. D. Schulze, *Theory and Applications of OFDM and CDMA: Wideband Wireless Communications*. Meschede, Germany: John Wiley & Sons, Ltd, 2005.

[54] M. Majo Boter, “Design and implementation of an OFDM-based communication system for the GNU Radio platform,” Masters dissertation, University of Stuttgart, Germany, 2011.

[55] W. Zhou, G. Villemaud, and T. Risset, “Full duplex prototype of OFDM on GNURadio and USRPs,” in *2014 IEEE Radio and Wireless Symposium, RWS*, 2014, pp. 217–219.

[56] X. Wang, T. T. Tjbung, and C. S. Ng, “Error probability performance of ofdm-adsl systems,” in *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration . IEEE*, 1998, vol. 6, pp. 3326–3331.

[57] M. Poggioni, L. Rugini, and P. Banelli, “A novel simulation model for coded OFDM in Doppler scenarios,” *IEEE Trans. Veh. Technol.*, vol. 57, no. 5, pp. 2969–2980, 2008.

[58] J. de Mingo, P. L. Carro, and P. García-Dúcar, “Antenna effects in DVB-H mobile rebroadcasters,” *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1155–1161, 2009.

[59] A. Y. Gafer, S. Elsadig, and J. Varun, “Front-end signal to noise ratio estimation for DVBT fixed reception in flat-fading channel,” in *ICIAS 2012 - 2012 4th International Conference on Intelligent and Advanced Systems: A Conference of World Engineering, Science and Technology Congress (ESTCON) - Conference Proceedings*, 2012, vol. 1, pp. 296–300.

[60] Y. Li, “Blind SNR estimation of OFDM signals,” in *2010 International Conference on Microwave and Millimeter Wave Technology*, 2010, pp. 1792–1796.

[61] Y. Li, “Estimation of Signal to Noise Ratio in Receivers,” 2013.

[62] S. Baumgartner and G. Hirtz, “A Blind ML-SNR Estimation Method for OFDM Systems in Dispersive Fading Channels,” in *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, 2014, no. 1, pp. 475–479.

[63] H. Xu and H. Zheng, “The Maximum-Likelihood SNR Estimation Algorithm for QAM Signals,” in *2006 8th international Conference on Signal Processing*, 2006, pp. 2–5.

[64] A. Scaglione, “A sparse EM algorithm for blind and semi-blind identification of doubly selective OFDM channels,” in *Signal Processing Advances in Wireless Communications (SPAWC), 2010 IEEE Eleventh International Workshop on*, 2010, no. 2, p. PP. 1–5.

[65] P. Jos, “Experimental Study on Spectrum Sensing for Cognitive Radio Networks,” Masters dissertation, Instituto Superior Tecnico, 2011.

[66] A. S. Harrington, C. Hong, A. L. Piazza, and S. Jones, “Software Defined Radio: The Revolution of Wireless Communication,” Team Paper, Ball State University, Muncie, United States, 2004.

[67] A. Azarfar, J.-F. Frigon, and B. Sanso, “A performance comparison of cognitive versus traditional radio networks,” in *2010 IEEE Globecom Workshops*, 2010, pp. 778–782.

[68] Wireless Innovation Forum, “What is Software Defined Radio,” *Forum Am. Bar Assoc.*, p. 6, 2011.

[69] K. Dabcevic, “Evaluation of Software Defined Radio platform with respect to implementation of 802.15. 4 Zigbee,” Masters dissertation, Malardalens University, Sweden, 2011.

[70] C. G. Germano, “Development of a Software Defined Radio (SDR) for Cognitive Radio (CR) Communication Systems,” Masters dissertation, Instituto Superior Tecnico, Lisboa, Portugal, 2014.

[71] D. Fiske, “Federal Communications Commision.” [Online]. Available: http://www.fcc.gov/Bureaus/Engineering_Technology/News_Releases/2000/nret0004.html. [Accessed: 01-Jan-2015].

[72] Wireless Innovation Forum, “Wireless Innovation Forum.” [Online]. Available: www.wirelessinnovation.org/. [Accessed: 20-May-2015].

[73] T. N. Jain, Shilpa, “Evolution from SDR to Cognitive Radio,” *Indian J. Appl. Res.*, vol. 4, no. 8, pp. 248–253, 2014.

[74] O. Sarwar and T. Kuhn, “Software Defined Radio (SDR) for Deep Space Communication,” Masters dissertation, Lulea University of Techonlogy, Sweden, 2013.

[75] J. D. Davis, C. P. Thacker, and C. Chang, “BEE3: Revitalizing Computer Architecture Research,” 2009.

[76] V. E. Chiriseri, “RHINO ARM Cluster Control Management System.,” Masters dissertation, University of Cape Town, South Africa, 2014.

[77] S. Winberg, A. Langman, and S. Scott, “The RHINO Platform – Charging Towards Innovation and Skills Development in Software Defined Radio,” *SAICSIT ’11 Proc. South*

African Inst. Comput. Sci. Inf. Technol. Conf. Knowledge, Innov. Leadersh. a Divers. Multidiscip. Environ., pp. 334–337, 2011.

[78] G. Inggs, D. Thomas, and S. Winberg, “Building a Rhino Harness A Software Defined Radio Toolflow for rapid prototyping upon FPGAs,” in *Proceedings of the IEEE International Conference on Industrial Technology*, 2013, pp. 1098–1103.

[79] M. Ettus, “Ettus Resarch.” [Online]. Available: http://www.ettus.com. [Accessed: 19-May-2015].

[80] S. Scott, “RHINO: Reconfigurable Hardware Interface for Computation and Radio,” Masters dissertation, University of Cape Town, South Africa, 2011.

[81] BEECUBE, “BEEcube.” [Online]. Available: http://www.beecube.com/. [Accessed: 11-Mar-2016].

[82] CASPER, “CASPER - Collaboration for Astronomy Signal Processing and Electronics Research.” [Online]. Available: https://casper.berkeley.edu/. [Accessed: 23-Jan-2016].

[83] D. C. T. and G. A. Tagliarini, “Prototyping with GNU radio and the USRP - where to begin,” in *IEEE Southeastcorn 2009*, 2009, no. March 2009, pp. 50–54.

[84] and I. T. L. A. M. L. Kuo-Hao, “Spectrum sensing based on time covariance matrix using GNU radio and USRP for cognitive radio,” in *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2011, no. May 2011, pp. 1–6.

[85] B. Li, “Analysis and Design of Software Defined Radio,” in *2011 International Conference on Internet Computing and Information Services*, 2011, pp. 415–418.

[86] Ettus Research, “Support: Knowledge Base.” [Online]. Available: http://www.ettus.com/kb/detail/usrp-bandwidth. [Accessed: 16-Jan-2016].

[87] S. M. Njoki, “Implementation of wireless communications on GNU Radio.,” Masters dissertation, University of North Texas, United States, 2012.

[88] R. Anil, R. Danymol, H. Gawande, and R. Gandhiraj, “Machine Learning Plug-ins for GNU Radio Companion,” in *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference*, 2014, pp. 1–5.

[89] A. Csete, “Simple SSB transmitter using complex bandpass filter.” [Online]. Available:

http://www.oz9aec.net/index.php/gnu-radio-blog/364-simple-ssb-transmitter-using-complex-bandpass-filter. [Accessed: 21-Jan-2016].

[90] M. Ettus, "USRP N210." [Online]. Available: http://www.ettus.com/product/details/UN210-KIT. [Accessed: 19-May-2015].

[91] M. Fähnle, "Software-Defined Radio with GNU Radio and USRP / 2 Hardware Frontend : Setup and FM / GSM Applications," Honours thesis, Hochschule Ulm University of Applied Sciences, Germany, 2010.

[92] Ettus Research, "Selecting an RF Daughterboard." [Online]. Available: https://www.ettus.com/content/files/kb/Selecting_an_RF_Daughterboard.pdf. [Accessed: 12-Jan-2016].

[93] S. Chuprun, J. Kleider, C. Bergstrom, M. Systems, and S. Group, "Emerging Software Defined Radio Architectures Supporting Wireless High Data Rate OFDM," in *Radio and Wireless Conference, 1999. RAWCON 99*, 1999, no. 480, pp. 117–120.

[94] M. Ettus, "USRP Hardware Driver (UHD) - Ettus Research." [Online]. Available: https://www.ettus.com/sdr-software/detail/usrp-hardware-driver. [Accessed: 05-Feb-2016].

[95] S. Shellhammer, "Spectrum Sensing in IEEE 802.22," in *2008 IAPR Workshop on Cognitive Information Processing (CIP '08)*, 2008, pp. 1–6.

[96] C. E. Metz, "Basic principles of ROC analysis," *Semin. Nucl. Med.*, vol. 8, no. 4, pp. 283–298, Oct. 1978.

[97] A. Slaby, "ROC analysis with Matlab," in *Proceedings of the International Conference on Information Technology Interfaces, ITI*, 2007, pp. 191–196.

[98] S. Atapattu, C. Tellambura, and H. Jiang, "Analysis of area under the ROC curve of energy detection," in *IEEE Transactions on Wireless Communications*, 2010, vol. 9, no. 3, pp. 1216–1225.

[99] E. Blossom, "Overview of GNU Radio System." [Online]. Available: https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion. [Accessed: 17-May-2015].

[100] E. Blossom, “UbuntuInstall - GNU Radio - gnuradio.org.” [Online]. Available: https://gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall. [Accessed: 12-Jan-2016].

[101] MacPorts, “MacPorts and the Shell.” [Online]. Available: https://guide.macports.org/chunked/installing.shell.html. [Accessed: 12-Jan-2016].

[102] Y. Fang, W. Zhao, X. Wang, F. Jiang, and W. Yin, “Comparative Analysis of Windowing Techniques in Minimizing side lobes in an Antenna Array,” in *Communications and Signal Processing (ICCSP), 2014 International Conference*, 2012, no. 5, pp. 1380–1383.

[103] J. Zhong, Z. Han, and W. Lu, “Design of windows with steerable sidelobe dips,” *IEEE Trans. Signal Process.*, vol. 40, no. 6, pp. 1452–1459, 1992.

[104] Sherif, “BER simulation in OFDM with amplitude clipping,” 2011. [Online]. Available: http://www.mathworks.com/matlabcentral/answers/2784-ber-simulation-in-ofdm-with-amplitude-clipping. [Accessed: 15-Jun-2015].

Appendix A

The MATLAB code for SNR versus the probability of detection for varying sample size is presented in this section. This is as explained in section 4.4 of Chapter 4. Part of the signal generation code is available from [104] in MathWorks.

```
%%-----SNR vs. Pd varying sample size------%%
clc
close all
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Parameters_128 FFT%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
K = 4; % QPSK constellation
L = 128; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;
```

```
% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;
if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd1(i) = Detect/kk;
end
plot(S_N_dB,Pd1, '-go');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Parameters_256 FFT%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
K = 4; % QPSK constellation
L = 256; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
```

```
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;
if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-b^');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
K = 4; % QPSK constellation
L = 512; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
```

```
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;

if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
```

```
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-ms');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Parameters_1024FFT%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
K = 4; % QPSK constellation
L = 1024; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
```

```
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy


%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;
if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-r*');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Parameters_2048_FFT%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
K = 4; % QPSK constellation
L = 2048; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end
```

```
% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;

if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-cp');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');
```

Appendix B

The MATLAB code for SNR versus the probability of detection with varying probability of false alarm is presented in this section. This is as explained in section 4.5 of Chapter 4. Part of the signal generation code is available from [104] in MathWorks.

```
%%%%------------SNR vs. Pd with varying Pfa--------------%%%%
clc
close all
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%ParametersPfa=0.01%%%%%%%%%%%%%%%%%%%%%%%%%%%

K = 4; % QPSK constellation
L = 2048; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end
```

```
% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.01;
for kk=1:10000 % Monte Carlo Iterations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;
if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-mo');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ParametersPfa=0.1%%%%%%%%%%%%%%%%%%%%%%%%%%

K = 4; % QPSK constellation
L = 2048; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source
```

```
% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.1;
for kk=1:10000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;

if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-r*');
```

```
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ParametersPfa=0.2%%%%%%%%%%%%%%%%%%%%%%%%%%

K = 4; % QPSK constellation
L = 2048; % fft length
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;
S_N_dB = -25:1:0; % snr range
S_N = 10.^(S_N_dB./10); % linear scale

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source


% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);


% 3. IFFT computation
columns=length(qpsk_mod_data)/L;%L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);


% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end


% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end


% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;


% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);
for i=1:length(S_N_dB)
Detect=0;
Pfa=0.2;
for kk=1:10000 % Number of Monte Carlo Simulations
```

```
%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N(i)).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold = (qfuncinv(Pfa)./sqrt(L))+ 1;
if(test_stat >= threshold) % Decision making.
Detect = Detect+1;
end
end
Pd(i) = Detect/kk;
end
plot(S_N_dB,Pd, '-bs');
hold on
grid on
xlabel('Signal To Noise Ratio (dB)');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix C

The MATLAB code for probability of false alarm versus the probability of detection for varying SNR is presented in this section. This is as explained in section 4.6 of Chapter 4. Part of the signal generation code is available from [104] in MathWorks.

```
clc
close all
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -6 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
%-----S_N in decibels-----%
S_N_dB = -6.2206;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;
K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end
```

```
% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
end
plot(Pfa, Pd, '-m*')
hold on plot(Pfa, Pd_the, 'g')
grid on
title('ROC plot for Probability of False Alarm vs Probability of Detection for
S_N');
xlabel('Probability Of False Alarm');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -11 %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
%-----S_N in decibels-----%
S_N_dB = -11.4532;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
```

```
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%
```

```
%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
end
plot(Pfa, Pd, '-go')
hold on plot(Pfa, Pd_the, 'g')
grid on
title('ROC plot for Probability of False Alarm vs Probability of Detection for
S_N');
xlabel('Probability Of False Alarm');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -15 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -15.4248;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM                                         SIGNAL
GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy
%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;

if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
end
plot(Pfa, Pd, '-cs')
hold on plot(Pfa, Pd_the, 'g')
grid on
title('ROC plot for Probability of False Alarm vs Probability of Detection for
S_N');
xlabel('Probability Of False Alarm');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -17.8534;

%-----Linear Value of S_N-----%
```

```
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end
% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy
```

```
%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
end
plot(Pfa, Pd, '-b^')
hold on plot(Pfa, Pd_the, 'g')
grid on
title('ROC plot for Probability of False Alarm vs Probability of Detection for
S_N');
xlabel('Probability Of False Alarm');
ylabel('Probability Of Detection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -22 %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -22.48;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;
K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);
```

```
% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
end
plot(Pfa, Pd, '-rp')
hold on plot(Pfa, Pd_the, 'g')
grid on
title('ROC plot for Probability of False Alarm vs Probability of Detection for
S_N');
xlabel('Probability Of False Alarm');
ylabel('Probability Of Detection');
```

Appendix D

The MATLAB code for probability of false alarm versus the probability of misdetection for varying SNR is presented in this section. This is as explained in section 4.7 of Chapter 4. Part of the signal generation code is available from [104] in MathWorks.

```
clc
close all
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -6 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -6.2206;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
```

```
end
% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;

if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
Pm(p) =1-Pd(p);
end
plot(Pfa, Pm, '-m*')
hold on plot(Pfa, Pd_the, 'g')
grid on
xlabel('Probability Of False Alarm');
ylabel('Probability Of Misdetection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
%-----S_N in decibels-----%
S_N_dB = -11.4532;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);
```

```
%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
```

```
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
Pm(p) =1-Pd(p);
end
plot(Pfa, Pm, '-go')
hold on plot(Pfa, Pd_the, 'g')
grid on
xlabel('Probability Of False Alarm');
ylabel('Probability Of Misdetection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -15 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -15.4248;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;
K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
```

```
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
Pm(p) =1-Pd(p);
end
plot(Pfa, Pm, '-cs')
hold on plot(Pfa, Pd_the, 'g')
grid on
xlabel('Probability Of False Alarm');
ylabel('Probability Of Misdetection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -17 %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

%-----S_N in decibels-----%
S_N_dB = -17.8534;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);
```

```
%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;

K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of False
Alarm (Pf) %
for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy
```

```
%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
Pm(p) =1-Pd(p);
end
plot(Pfa, Pm, '-b^')
hold on plot(Pfa, Pd_the, 'g')
grid on
xlabel('Probability Of False Alarm');
ylabel('Probability Of Misdetection');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SNR -22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
%-----S_N in decibels-----%
S_N_dB = -22.48;

%-----Linear Value of S_N-----%
x=S_N_dB./10;
S_N = 10.^(x);

%-----Probability of False Alarm-----%
Pfa = 0.01:0.05:1;
L = 2048;
K = 4; % QPSK constellation
cyclic_prefix_len = ceil(0.25*L); % cyclic prefix
ifft_points = L;
fft_points = L;

%%%%%%%%%%%%%%%%%%%%%%%%%%%OFDM_SIGNAL_GENERATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 1. 1 x 2048 data vector
data = randsrc(1, L, 0:K-1);% data source

% 2. Modulating QPSK
qpsk_mod_data = pskmod(data, K);

% 3. IFFT computation
columns=length(qpsk_mod_data)/L; %L= block_size
data_matrix = reshape(qpsk_mod_data, L, columns);
cyclic_prefix_start = L - cyclic_prefix_len;
cyclic_prefix_end = L;
for i=1:columns,
ifft_data_matrix(:,i) = ifft((data_matrix(:,i)),ifft_points);

% Adding Cyclic Prefix
for j=1:cyclic_prefix_len,
actual_cyclic_prefix(j,i) = ifft_data_matrix(j+cyclic_prefix_start,i);
```

```
end

% OFDM block creation
ifft_data(:,i) = vertcat(actual_cyclic_prefix(:,i),ifft_data_matrix(:,i));
end

% 4. Serial stream generation
[rows_ifft_data cols_ifft_data]=size(ifft_data);
len_ofdm_data = rows_ifft_data*cols_ifft_data;

% Generated OFDM signal
ofdm_signal = reshape(ifft_data, 1, len_ofdm_data);

%% Simulation to plot Probability of Detection (Pd) vs. Probability of    False
Alarm (Pf) %

for p = 1:length(Pfa)
Detect = 0;
for kk=1:40000 % Number of Monte Carlo Simulations

%-----AWGN noise with mean 0 and variance 1-----%
Noise = randn(1,L);

%-----PU Signal------%
ofdm_signal = sqrt(S_N).*randn(1,L);
received_signal = ofdm_signal + Noise; % received signal with noise
energy_calc = abs(received_signal).^2; % received signal energy

%-----test statistic------%
test_stat =(1/L).*sum(energy_calc);%

%-----threshold valuation-----%
threshold(p) = (qfuncinv(Pfa(p))./sqrt(L))+ 1;
if(test_stat >= threshold(p)) % Decision making.
Detect = Detect+1;
end
end
Pd(p) = Detect/kk;
Pm(p) =1-Pd(p);
end
plot(Pfa, Pm, '-rp')
hold on plot(Pfa, Pd_the, 'g')
grid on
xlabel('Probability Of False Alarm');
ylabel('Probability Of Misdetection');
```

Appendix E

This appendix presents the code used to make the OOT modules as described in section 5.3.2.1 of Chapter 5.

E. 1: Modified XML file

```
<?xml version="1.0"?>
<block>
  <name>average_ff</name>
  <key>howto_new_average_ff</key>
  <category>howto_new</category>
  <import>import howto_new</import>
  <make>howto_new.average_ff($sample_size)</make>
  <param>
    <name>Sample Size</name>
    <key>sample_size</key>
         <type>int</type>
  </param>
<sink>
    <name>in</name>
    <type>float</type>
  </sink>
<source>
    <name>out</name>
    <type>float</type>
  </source>
</block>
```

E. 2: Cmake command

```
cmake -DCMAKE_MODULES_DIR=/opt/local/share/cmake -
DCMAKE_INSTALL_PREFIX=/opt/local -
DPYTHON_EXECUTABLE=/opt/local/bin/python2.7 -
DPYTHON_INCLUDE_DIR=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/H
eaders -
DPYTHON_LIBRARY=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/Python
-DSPHINX_EXECUTABLE=/opt/local/bin/rst2html-2.7.py -
DGR_PYTHON_DIR=/opt/local/Library/Frameworks/Python.framework/Versions/2.7/lib/pytho
n2.7/site-packages ..
```

Appendix F

This section presents the C++ code for the threshold block work function. This is modified from the *gr_modtool* provided in GRC.

```
int
  threshold_ff_impl::general_work (int noutput_items,
                     gr_vector_int &ninput_items,
                     gr_vector_const_void_star &input_items,
                     gr_vector_void_star &output_items)
  {
    const float *in = (const float*) input_items[0];
    float *out = (float*) output_items[0];

 // Do <+signal processing+>
    //define variables
    double threshold, set_Pfa;
    int i, set_N;

    set_Pfa = 0.1;
    set_N = 2048;

    //theoretical value of threshold
    threshold = ; the threshold is dynamically set

    if (in[i]>=threshold)
    out[i]= 1 ;

    else
    out[i] = 0 ;

    // Tells runtime system how many input items were consumed on each input stream.

    consume_each (noutput_items);

    // Tells runtime system how many output items were produced.
    return noutput_items;
  }
```