DEVELOPING AN INTERNET OF THINGS PROTOCOL TEST BED FOR EVALUATING A MODULATION TECHNIQUE AND A MEDIUM ACCESS CONTROL PROTOCOL

by

Ranganathan Sampathkumar, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Engineering
May 2020

Committee Members:

Harold Stern, Chair

Semih Aslan

George Koutitas

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

I am dedicating this thesis to my parents, and my thesis advisor and committee members who helped in each step of this defense

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr Harold Stern for this continuous motivation and supervision throughout my master's at Texas State University. His guidance helped me a lot to complete this thesis in a successful way. The technical advice, suggestion and forwardable approach are one of the main reasons for completing this research.

I would like to thank my committee members Dr. George Koutitas, Dr. Semih Aslan for giving me necessary tips and information that helped me in completing this defense. The committee members advise helped me to understand the technology and helped me in gaining more knowledge in this IoT field. I would also like to thank my graduate advisor Dr Vishu Viswanathan who guided me in a right path to choose subjects that would help my thesis and my future career.

I would like to extend my gratitude towards my family and my friends who directly or indirectly helped and motivated me with my research.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATION

Abbreviation	Description
ADC	Analog-to-Digital Convertor
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CA	Collision Avoidance
DAC	Digital-to-Analog Convertor
DSP	Digital Signal Processor
ELF	Extreme Low Frequency
EHF	Extreme High Frequency
FCS	Frame Check Sequence
FFD	Full Function Device
FPGA	Field Programmable Gate Array
FPRF	Field Programmable Radio Frequency
GUI	Graphical User Interface
GRC	GNU Radio Companion
GPL	General Public License
HF	High Frequency
I and Q	In-Phase and Quadrature
IoT	Internet of Things

ITU	International Telecommunication Union
IC	Integrated Circuit
LQI	Link Quality Indication
LLC	Logical Link Control
LPWPAN	Low Power WPAN
LSB	Least Significant Bit
LF	Low Frequency
LTE	Long Term Evaluation
MF	Medium Frequency
MAC	Medium Access Control
MHR	Medium Access Control Header
MPDU	Medium Access Control Protocol Data Unit
MSB	Most Significant bit
MSK	Minimum Shift keying
OQPSK	Offset Quadrature Phase Shift Keying
OS	Operating System
PAN	Personal Area Network
PHR	Physical Header
PHY	Physical Layer
PPDU	Physical Protocol Data Unit

PSDU	Physical Service Data Unit
PC	Personal Computer
PSK	Phase Shift Keying
RFD	Reduced Function Devices
SDR	Software Defined Radio
SFD	Start Frame Delimiter
SHR	Synchronization Header
SSCS	Service Specific Convergence Sublayer
UMTS	Universal Mobile Telecommunication System
USRP	Universal Software Radio Peripheral
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

ABSTRACT

Internet of Things is used in a wide range in our day to day life, but there are many potential new IoT applications that cannot be implemented due to power and capacity limits in IoT systems realizable with current technology and existing standards. New wireless protocols and advanced modulation techniques are promising features for the development of real time, higher capacity, lower power, user friendly IoTs, but there are currently no low-cost commercially available hardware tools that can probe deeply into the operation of the physical and link layers of current IoT systems. This lack of low-cost tools makes it difficult for researchers to develop new modulation techniques and medium access protocols and to evaluate how they will act in an IoT system. The main aim of our research is to develop a low-cost test bed using an open source Software Defined Radio (SDR) that has full duplex capability along with the open-source software. The test bed will have the capability to control and evaluate wireless protocols and modulation techniques. This test bed will be extremely useful for future IoT development.

1. INTRODUCTION

1.1. Overview

Radio Frequency (RF) generally refers to an oscillation rate of alternating electric current in the frequency range from 20 kHz to around 300 GHz. Radio frequencies are generated and powered within many electronic devices such as transmitters, receivers, computers, and televisions. Radio frequencies are also used in communications, signaling, and control systems [1]**.** The International Telecommunication Union (ITU) has subcategorized radio spectrum into different classifications according to different frequencies. Figure 1 below shows the classification of radio frequencies and their bandwidth ranges.

RADIO FREQUENCIES

CLASS	ABBREVIATION	RANGE
extremely low frequency	ELF	below 3 kilohertz
very low frequency	VLF	3 to 30 kilohertz
low frequency	LF	30 to 300 kilohertz
medium frequency	MF	300 to 3000 kilohertz
high frequency	HF	3 to 30 megahertz
very high frequency	VHF	30 to 300 megahertz
ultrahigh frequency	UHF	300 to 3000 megahertz
superhigh frequency	SHF	3 to 30 gigahertz
extremely high frequency	EHF	30 to 300 gigahertz

Figure 1. Classification of Radio Frequencies and their Bandwidth Ranges [2]

RF circuits are applied in many wireless communication systems, including cellular modems, RF transceivers, medical devices, and wireless communication chips for Wi-Fi, Bluetooth, and GPS receivers [1]. The Internet of Things (IoT) is the system of interrelated computing devices, mechanical, or digital machines that have been provided with a unique identifier and have the capacity to transfer data over the network without the need of human-human or human-computer interaction [3]. Advances in RF signaling,

and IC technology have led us to widespread usage of low power, low cost wireless communication systems such as RFID, Wi-Fi, Zigbee, and Bluetooth. These wireless communication systems are generally referred to as the Internet of Things. These systems, all of which have standards, support many applications but the standards and current technology are limiting the systems' ability to increase transmission speeds, expand range, improve accuracy, and reduce power consumption. These systems generally use Star or Mesh networks to implement communication. Ever since the initial implementations of IoT there have been concerns about security to protect data, and these concerns have been handled side by side by scientists and researchers. The IoTs are generally applied in four different fields. They are

1. Consumer applications – Created for smart homes and elder care etc.,
2. Commercial applications – Used as medical and healthcare, transportation etc.,
3. Industrial applications – Used at manufacturing companies, agriculture etc.,
4. Governmental applications – Used at military, government agencies, etc.

Figure 2 below gives the representation of the technological development of IoTs from the year 2000 to now and to the near future years. The Y-axis represents the technology reach whereas the X-axis represents the year of IoT evaluation reach. One can see the IoT that started as a small sensor in the internet world has identified itself as a need among people in day to day life. Although IoT looks interesting and easy to use from the outside view, it has very tough architecture with different layers that make the IoT fully functional. The different details about the size, space considerations, and protocol layers for various applications will be further explained in Chapter 5.

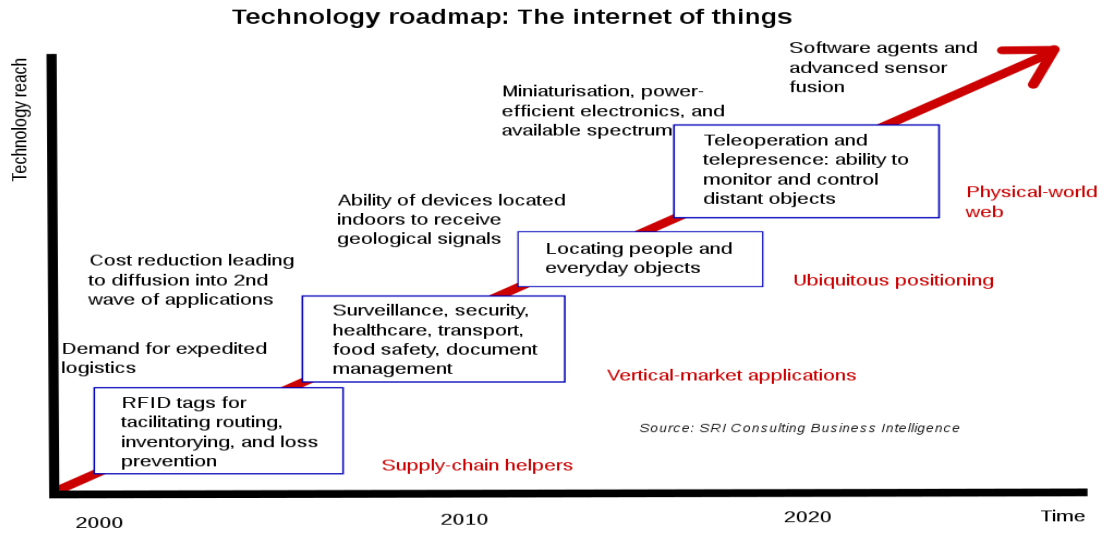


Figure 2. Evaluation of IoT with Respect to Time [3]

Figure 3 below represents the Internet of Things and its day to day work with human and computer life in business, social and commercial workspaces.

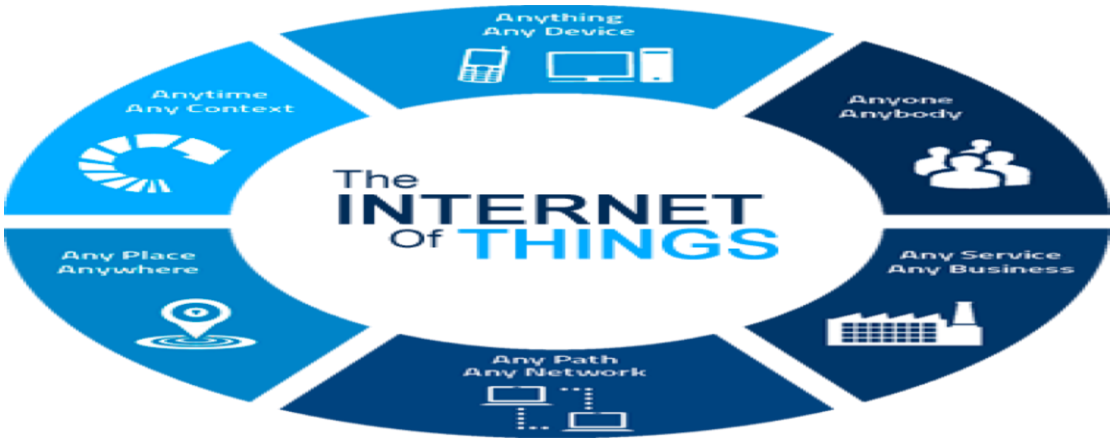


Figure 3. Internet of Things [4]

1.2. Current Research Gaps and Problem Description

Internet of Things systems are becoming more popular day by day. Newly developing Internet of Things systems have high chances of providing more real time applications, but some of the applications are not being achieved well because of the limits in the protocols and the modulation techniques of current IoT systems. There are

currently no low-cost commercially available hardware tools that can probe deeply into the operation of the physical and data link layers of current IoT systems. This lack of low-cost tools makes it difficult for researchers to develop new modulation techniques and medium access protocols and to evaluate how they will act in an IoT system. The currently available hardware development tools are costlier than many researchers can afford, and even most radio tools available to evaluate the Internet of Things systems like Zigbee, Wi-Fi and Bluetooth do not have the capability to act as transmitter and receiver at the same time. These wireless communication systems, all of which have standards, support many applications but the standards and current technology are limiting the systems' ability to increase transmission speeds, expand range, improve accuracy, and reduce power consumption. Development of new standards and new wireless systems will enable additional applications, and the development efforts will be optimized if engineers have access to relatively inexpensive hardware and software development tools. The most promising tool to the research scientists and developers for making the deep dive into the IoT protocol field is the Software Defined Radio (SDR). Most currently available low cost SDRs are half duplex. In addition to being half-duplex, most of the low-cost SDRs do not have the frequency range, the flexibility, or the speed and accuracy necessary to support hardware development for a wide set of new IoT protocol systems. Most researchers try to combine two half duplex SDRs or work with two different channels in a single half duplex SDR, making the communication more complex with less throughput. When IoT operates at different standards some of the SDRs that operate only at low frequency and are half duplex are insufficient to get all the necessary outputs that the researchers need.

1.3. Thesis Focus

In this thesis we are going to develop a low-cost, flexible IoT testbed. We will achieve this goal by integrating our hardware SDR with open source software to create a system capable of providing researchers with data to perform detailed evaluation of existing and newly proposed IoT protocols. We are going to use one of the most advanced inexpensive hardware tools that has the capability to work as full duplex and that will be integrated with GNU Radio software that helps in capturing the transmission and reception of RF signals.

The operation of the proposed IoT test bed can be verified by creating a data packet containing a message, formatting and modulating it under a certain protocol using Lime SDR as the primary hardware to transmit it over the airwaves, then receiving the same packet over the airwaves and demodulating it using the Lime SDR as primary hardware and extracting the original message. The thesis will also conclude by stating how other modulating techniques can be achieved in the future with the help of this IoT test bed and its hardware and software.

1.4. Thesis Organization

The thesis is organized into chapters as follows:

Chapter 1 provides an introduction and is subdivided into three subsections. They are the overview of the thesis, problem description and research gaps. This last subsection a short introduction of what is lagging in the current research, and the thesis focus that provides the scope for the work. Next, we focus more deeply by shifting to **Chapter 2** that describes Software Defined Radios and justifies selecting the Lime SDR as the appropriate SDR for our work. **Chapter 3** describes our open source software,

GNU radio, and its installation and coding process**. Chapter 4** describes Wireshark Protocol Analyzer and its applications **Chapter 5** describes the Zigbee IoT protocol and its standard, IEEE 802.15.4. We will be using Zigbee devices to verify the operation of our testbed. **Chapter 6** explains the work of Embedding GNU Radio blocks with the Lime SDR. **Chapter 7** evaluates the previous work done by the researchers and the gap which we fill with our current work in the thesis. **Chapter 8** explains the IoT protocol transceiving using our Lime SDR and GNU Radio. **Chapter 9** explains the output result and analysis that have been obtained with the help of Wireshark. **Chapter 10** shows the conclusion of thesis work and suggests future work that can be made through Lime SDR for evaluation.

2. HARDWARE – SOFTWARE DEFINED RADIO

2.1. Introduction of the Software Defined Radio

Software Defined Radio (SDR) is a radio communication system that has been implemented by software along with the hardware like an embedded system rather than implementing directly by the hardware. The hardware has the capability to embed mixers, amplifiers, modulator, demodulator, and detector blocks in a personal computer. A basic SDR consists of a personal computer equipped with a sound card, another Analog to Digital converter, and some form of RF front end [5]

2.2. History of SDR

The term software radio was coined at Garland, Texas around the year 1984 by a division of E-Systems Inc to refer to a digital baseband receiver, as published in the E-team company newsletter. The term included further research and later was developed to a full-fledged Software Defined Radio in the year 1995 by Stephen Blust [6]. The SDR technology was so advanced and successful that it attracted the US military and specifically the US Air-Force. One of the first public software radio initiatives was the U.S. DARPA-Air Force military project named Speak Easy. The primary goal of the Speak Easy project was to use programmable processing to emulate more than 10 different types of existing military radios, operating in the frequency band between 2 and 2000 MHz. Another Speak Easy design goal was to be able to easily incorporate new coding and modulation standards in the future, so that military communications could keep pace with advances in coding and modulation techniques. The Speak Easy project was further elaborated into Speak Easy phase 1 and Speak Easy phase 2 [5]. Subsequently SDRs came into use in wider areas of research and in commercial

applications. SDRs are attractive for many military and commercial applications because they are flexible and can be reconfigured easily. Figure 4 shows the functional blocks of a software defined radio.

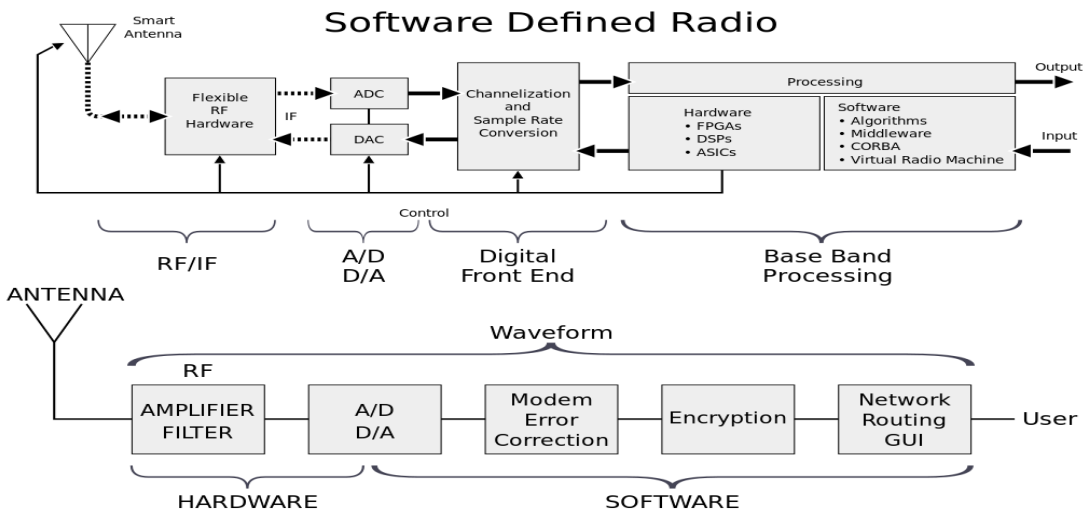


Figure 4. Software Defined Radio Block [5]

2.3. SDR Architecture / Operation Principle

Ideally an SDR should include its own analog stage in its hardware. It means that the ideal SDR would use an Analog to Digital Convertor (ADC) to convert analog signals to digital signals, a Digital to Analog convertor (DAC) to convert the digital to analog signals, and a baseband processor to process the signal. After reading the samples from the ADC, the next step is to process the signal which is handled by the software. In general, the SDR has a smart antenna attached to it. Every SDR has its own antenna or a separate flexible antenna attached to it that helps in receiving or transmitting radio frequency signals. In Figure 4 the antenna has been attached to the physical hardware which has DAC/ADC in it for digitizing the analog or making the incoming digital signal as analog. The number of antennae used, and the number of channels used changes according to the full duplex or half duplex capacity of the SDR. After sampling the

signal, the signal goes to the channelization block where it packetizes the signal according to Time Division or Frequency Division Multiplexing. Then it includes an important section of baseband signal processing where the hardware and software collaborate with each other and the hardware includes the DSP, FPGA etc., whereas the software includes algorithms, middleware, firmware driver installation, and coding the signal. To guarantee accuracy and appropriate security, the incoming signals entering the software signal blocks are first checked for modem error correction and sent for encryption before handing the message to the user.

2.4. Types of SDR

There are different types of Software Defined Hardware currently available on the market. Each SDR has its own standard for capturing the RF signal and working on the protocol level standards depends on each SDR's capacity. Different SDRs have different frequencies to control the IoT protocol. The best controlling depends on the capacity of the SDR too. Most low cost SDRs are half-duplex. Other SDR devices on the research markets include Blade RF, USRP, Air spy SDR, Hunter SDR and others. Each SDR has its own frequency to get integrated with GNU radio and in controlling the IoT protocol platform. Some of the SDRs come as a chip and some come in the form of USB format. One of the best examples of an SDR in USB format is the Hack RF. Hack RF, developed by great Scott Gadgets, is similar to the hardware that we ultimately selected but the main disadvantages of Hack RF are less capacity and that it is half duplex and cannot operate as a full duplex. These problems mean it cannot do both transmission and reception of radio frequency signals at the same time with a single channel. Hack RF which is half duplex can be directly connected with the PC, which is similar to the hardware that we

are going to integrate. Researches must use two hardware devices each with single channel or a single hardware device that uses both channel A and B in the software to control the protocols. This ends up in loss of packets. To overcome these disadvantages, there is an inexpensive, full duplex, and best SDR available on the market that has attracted many researchers and scientist. The hardware is called the Lime SDR.

2.5. Lime Software Defined Radio

In this thesis, we are going to use Lime SDR USB as our hardware to control and evaluate the IoT protocol. Figure 5 below shows the Lime SDR USB.

Figure 5. Lime SDR USB [7]

Figure 5 shows the Lime SDR USB used with a Type B plug. Lime SDR USB is a low-cost software defined radio platform based on Lime Microsystem. It uses the second generation FPRF transceiver LMS7002M, Altera Cyclone FPGA and Cypress FX3 USB3 superspeed microcontroller. This implementation allows the user to have control on the resources of communication standards, wireless networking etc., [7]. Lime SDR can send and receive UMTS, LTE, Bluetooth, Zigbee, RFID, and Digital Broadcasting Protocol. The Lime SDR platform gives students, inventors, and developers an intelligent and flexible device for manipulating wireless signals, so they can learn, experiment, and

develop with freedom from limited functionality and expensive proprietary devices [7]. Though we are using Lime SDR USB there are different types of Lime SDRs available in the market. They are Lime SDR USB, Lime SDR Mini, Lime Net Micro, Lime SDR PCIe, Lime SDR QPCIe, and Lime SDR GPIO Board. Up to the present time, SDRs have been of limited use as development tools for new RFID and IoT technologies. The Lime SDR is a significantly more advanced hardware tool compared to Hack RF One. It has a higher bandwidth capacity and full duplex capability so that uplink and downlink communication (that is the transmission and reception) can be performed at the same time. Lime SDR uses the latest technology and it is superior to other Software Defined Radios in all kind of aspects like bandwidth, flexibility and programmability. Lime SDR helps us in our thesis to work along with GNU Radio to evaluate the IoT protocols. Although we are using the Lime for IoT protocol evaluation, it can be used for RF capture, Audio Transmission and Reception, etc. Lime cannot operate standalone, instead it gets embedded with GNU Radio to work on evaluating IoT protocols. New wireless protocols are a promising feature for upgrading Internet of Things products. Devices can currently communicate over Internet, Wi-Fi, Zigbee, Bluetooth or any other IoT standard, but each IoT protocol requires its own radio with reliable hardware to deliver or interpret the signals received. In this thesis, the OS we use is the Linux Ubuntu whereas some of the SDRs that embed with GNU radio companion can work well with Windows and Mac, too. The physical structure of Lime SDR the same size as the Hack RF One. Although Lime simultaneously performs both the transmission and reception, its cost is still similar to other, half-duplex SDRs. Zigbee protocol that is acting under the IEEE standard 802.15.4 is one of the low-rate Wireless Personal Area Network (WPAN) protocols.

Further explanation of this protocol along with the hardware and the software we used will be discussed in the upcoming chapters

2.5.1. Features and Specifications

Table 1. Features and Specifications of Lime SDR USB

S. No	Features	Specifications
1.	RF Transceiver	Lime Microsystem LMS7002 MIMO FPRF
2.	FPGA	Altera Cyclone EP4CE40523
3.	USB Controller	Cypress USB 3.0
4.	Frequency	100 Khz-3.8Ghz
5.	Power Output	Up to 10dbm
6.	Multiplexing	2*2 MIMO
7.	Power Supply and status	USB power supply and LEDs

Lime SDR is the important tool for us to work on thesis. The RF transceiving is Lime microsystem LMS 7002 MIMO FRPF and FPGA is Altera Cyclone EP4CE40523 and USB Controller is Cypress USB 3.0. The above table explains the hardware features and specifications of Lime SDR USB. Each OS need USB3 drivers to read Lime SDR hardware in the PC. There are special installation procedures of the drivers in Windows and MAC book, but Linux Ubuntu comes inbuilt with the drivers that helps in detecting

the Lime SDR USB so that we don't need a special installation process for reading the Lime. In this thesis, our OS is going to be Linux Ubuntu.

2.6. Lime Suite Application in Lime SDR

Lime Suite is a collection of software supporting hardware platforms like Lime SDR, drivers for the LMS7002M transceiver RFIC, and other tools for developing LMS7-based hardware [8]. Some of the hardware supported by the lime suite includes

- Lime SDR
- STREAM LMS7002M UNITE (EVB7)
- LMS7007 UNITE (EVB7) through COM port
- Novena Laptop with LMS7 RF board

To install the Lime Suite drivers at Ubuntu Terminal download

sudo add-apt-repository -y ppa:myriadrf/drivers

sudo apt-get update

sudo apt-get install limesuite liblimesuite-dev liblimesuite-udev limesuite-images

sudo apt-get install soapysdr-tools soapysdr-module-lms7

Lime suite alone cannot work without installing its necessary dependencies for our test tool. Please note the code changes are for ubuntu and MAC users. Our thesis uses only Linux Ubuntu. The code for installing package dependencies is given at the end of the thesis report in the **appendix** section. The above code is the main terminal code for installing the Lime Suite package. Lime Suite driver components are shown in the figure below.

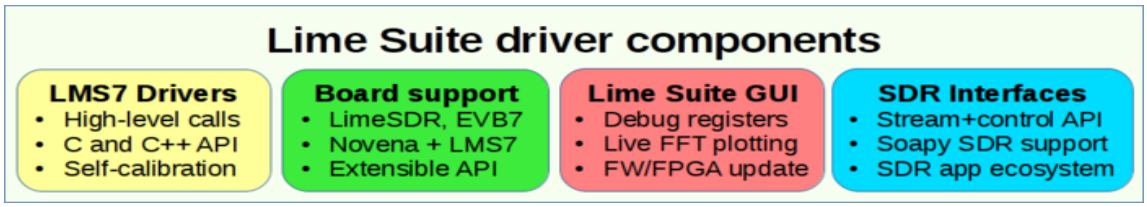


Figure 6. Lime Suite Driver Components [8]

2.7. Self-test with Lime Suite GUI and Lime SDR

The Lime Suite GUI is a special software tool for debugging and for checking the initial stage of work with Lime SDR. The developers provided self-test software prewritten that helps us to check the receiving of the Radio Frequency signals. This receiving helps us to know if the Lime SDR USB is working in a proper way. It is generally called a loopback test

2.8. Performance of a Loopback Test

Generally, a waveform is sent by uploading the "self test.ini" to the host computer and this uploaded host test file is made in connection with the Lime SDR USB by accessing the hardware in Lime Suite at the connection setting option in the option menu. The WCDMA signal is generated in the FPGA chip. An onboard RF switch connects from TX to RX. If the final received signal is generated in the FFT viewer, then we can confirm our hardware is working and the self-test is successful. This can also be checked by transmitting the signal from Lime Suite GUI and receiving it on the spectrum analyzer at the other end. The below figure shows the display corresponding to a successful loopback test.

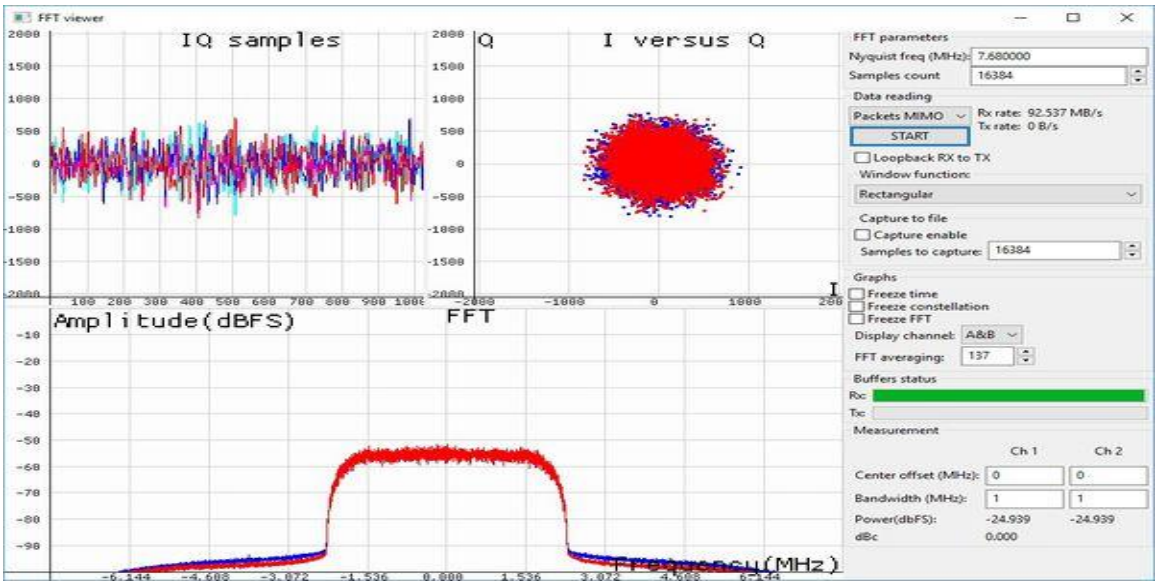


Figure 7. W-CDMA Receiver Graph [8]

3. GNU RADIO SOFTWARE

3.1. Introduction to GNU Radio

GNU Radio is an open source software development toolkit for SDR. GNU Radio can be simply considered as block-based software. GNU Radio when combined with minimal hardware allows construction of radios and thus turns usual hardware implementations into predominantly software implementations. The main goal of GNU Radio is to allow easy combination of signal and data processing blocks into powerful modulation, demodulation, and more complex signal processing systems [9]. GNU Radio is the main software that we use for our thesis that controls the Hardware Lime SDR. GNU Radio software will be in the form of blocks. The blocks can communicate using different data types. Generally, blocks are written in C++ or Python programming language. Python language acts as an interface to the signal processing block. The power of the scripting language is used to simply connect the signal processing blocks which can run at native speed without any interpretation [9]. GNU Radio Companion is the Graphical User Interface (GUI) offered by GNU Radio. This is generally referred to as GRC. Blocks can relate to each other and formed as a flow graph. These relation helps the user to manually edit the code and run the flow graph according to their research. Different projects have been already developed, for example GFSK receivers, GFSK transmitters that communicate at different rates. These previous research efforts help future developers to edit the generator code or to just edit the appropriate blocks to obtain the different transceiving of the radio signals. Each block usually does one job to keep them modular and flexible. The next chapter will describe the installation process and other working of GNU Radio.

3.2. History of GNU Radio

When engineers started to communicate using radio signals, they had to develop a special circuit for detection of a specific signal class or, more recently, design a specific integrated circuit to encode and decode the radio signals. These circuits could be used in tools to test, develop and debug wireless systems, but the test equipment was costly and not so friendly to use. SDR usually takes the analog signal processing technique and moves it to digital processing of the radio signal using algorithms developed in standard software. But this process quickly becomes tough to use even though it's possible. Each time when a researcher is developing a new device if the researcher uses a filter or other process, the researcher had to do it from scratch. The effort became much easier when GNU Radio was introduced into the market. The first GNU Radio software was published in 2001 by philanthropist John Gilmore with the funding of $320,000 (US) to Dr. Eric Blossom for code creation and project management duties [10]. GNU Radio is a framework that enables users to design, simulate, and deploy highly capable real-world radio systems. It is a highly modular, "flowgraph"-oriented framework that comes with a comprehensive library of processing blocks that can be readily combined to make complex signal processing applications [11]

3.3. Installation Procedure of GNU Radio

The main terminal code of Linux Ubuntu for installing GNU Radio is

$ apt install gnuradio

3.4. GNU Radio Companion (GRC)

The GNU Radio Companion was the first GNU Radio application and it's the front end to the Gnu Radio libraries for signal processing. GRC is effectively a Python

Code generation tool. When the flowgraph is compiled in GRC, it generates python code that creates desired GUI gadgets and connects the blocks in terms of flowgraph [10] .

3.5. Installation Procedure of GNU Radio Companion

$ gnu radio - companion

Note: GRC cannot be installed without installing GNU Radio.

3.6. Working with GRC

GRC usually starts with covering its interface. There are 5 parts of the interface. They are

- Library
- Toolbar
- Terminal
- Workspace
- Variables

The below figure shows the GNU Radio Companion with basic blocks.

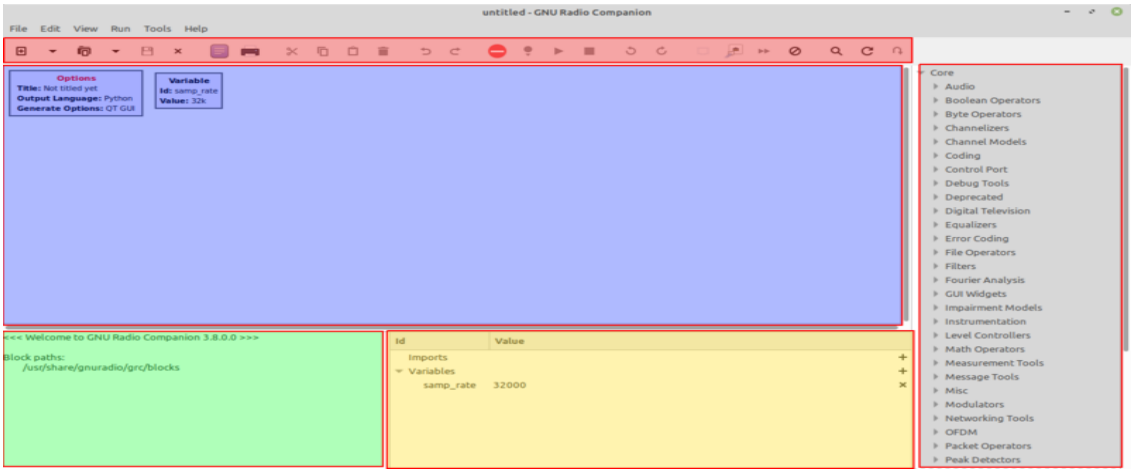


Figure 8. GNU Radio Companion Outline [8]

Some of the process that can be done with the GRC is searching for blocks like Waveform Generator, Boolean operators, Modulators, GUI Widgets, QT GUI, WT GUI etc. CTRL+F helps us to find the necessary blocks that can be converted to flowgraphs in the GNU Radio. GRC gives us the possibility to even modify properties like searching the blocks. Once we search a particular block and it appears on the screen, GRC gives a facility to right click the block and change properties like frequency, wavelength, disable/enabling a block etc., according to an individual's research. This technique allows us to edit the property and form a flow graph according to our desired frequency. One of the important properties needed at the start of the any flowgraph is the option block. It is a special block because the ID of the project will be assigned there including the title of the .grc file plus the generating options like QT GUI / WT GUI. If the ID turns blue, it, means that the project has not been saved yet. If the ID part is in red, then there is an error in the way the ID name was given. ID part allows us to manage our file space. Saving the file can be done by," filename.grc". GRC is a graphical interface that sits on the top of the GNU Radio which is created using Python code. This means when we save a file in .grc and run the flowgraph it runs a python file in terms of flowgraph. So, the whole file is in ". grc" whereas the ID is saved in ".py" format. An example of generated flowgraph is shown in Figure below.

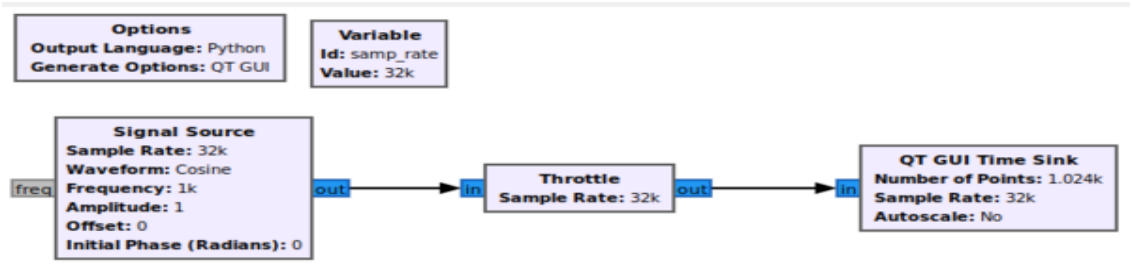


Figure 9. GNU Radio Generated Flowgraph [12]

The above flowgraph figure shows the simple example of the generated flowgraph that has the option file in QT GUI (Graphical Sink). The file will be saved in .grc before running it. The block signal source can be taken from the list of pre generated blocks in GRC. The throttle helps the flowgraph to make sure it does not consume all 100% cycle in the CPU and helps the computer to be responsive during execution. The final block is the important block “QT GUI Time Sink” that shows the output of Amplitude vs Time whereas if we select “QT GUI Frequency Sink” then the .grc will generate output according to the frequency. Sample rate has a significant effect on both the time and frequency sinks at the output graph. There are three main buttons for execution of the flowgraph. They are generating, execute and kill the flowgraph, accessible through F5, F6, and F7 respectively [12]. The flow graph also changes according to the data type such as float, complex etc. The difference of flowgraph in float and complex is shown in Figure 11 below. We cannot run a flowgraph’s block of different data types like one having Float data type and the other having complex data type. The complex flowgraph that includes the real and imaginary part is given in the figure below:

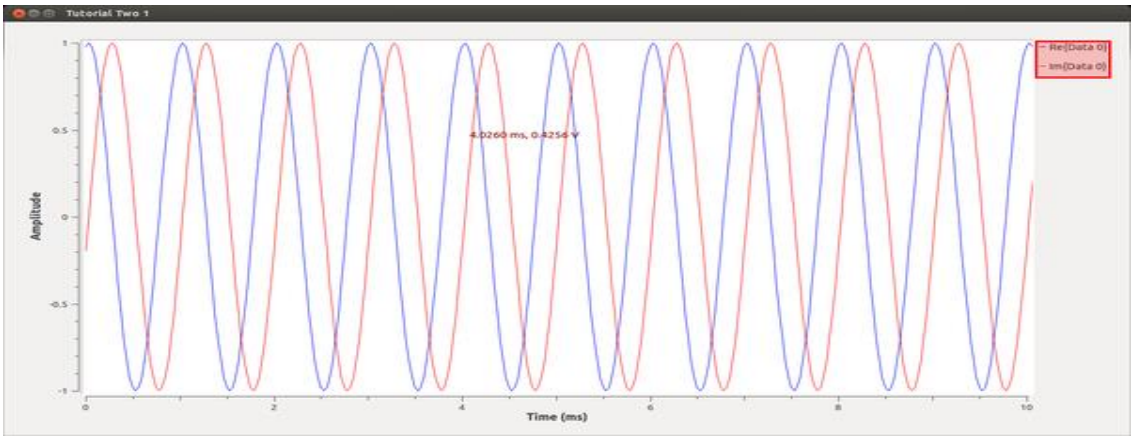


Figure 10. Graphical Representation of Complex Flowgraph's Output [12]

The block diagram and graphical output for the float type sine wave signal instead of complex type that excludes the imaginary part is given in the figure below

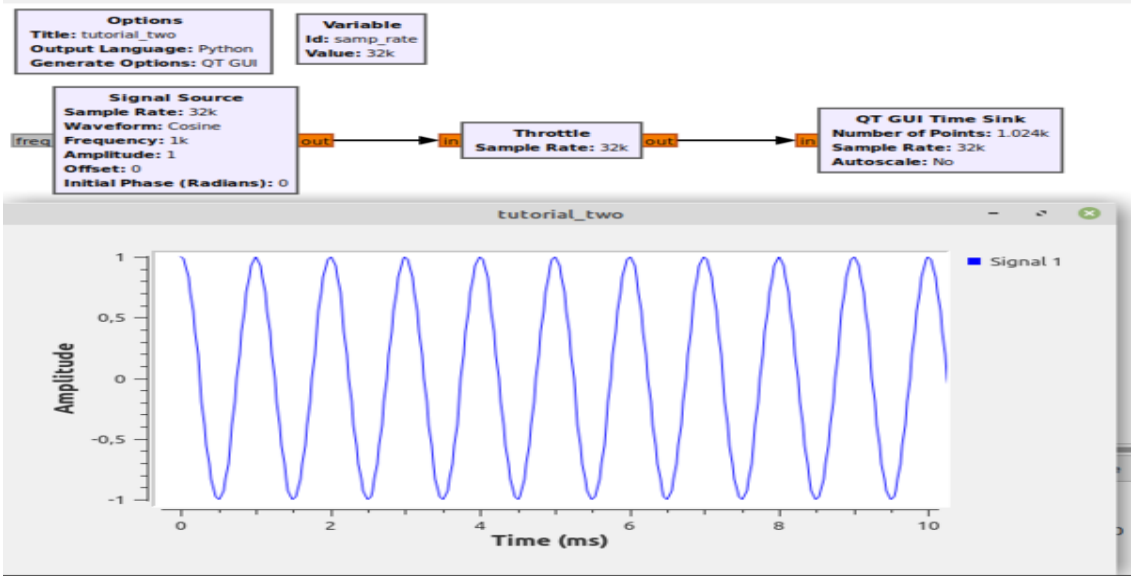


Figure 11. Cosine Wave Graphical Flowgraph's Output [12]

Output does not necessarily have to be a graphical output in GRC. GRC has the capability to produce a signal in terms of audio sink, packet sink etc, in our thesis the output will be in the form of message, thus the output sink changes according to it. Unlike SDR, GNU Radio has the capability to perform communication through digitized signals. Generally, a person's voice in phone becomes an electrical signal by converting the varying pressure but the audio signal at that point is analog. The computer cannot deal with analog as it deals with only 1s and 0s, so it has to be sampled and digitized. A fixed interval between the samples gives the signal sampling rate. This whole process, shown in Figure 12, is performed by an ADC (Analog to Digital Convertor) that plays an important role in our thesis. Like ADC, the DAC (Digital to Analog Converter) converts back the digital signal to an analog signal. Thus, we have ADC converting an analog signal into a sequence of numbers that helps in applying digital filters, speech recognition

or transmitting a signal using a digital link [11]. Some of the important blocks of GNU Radio include waveform generators, modulators, demodulators, filters, FFT, Fourier analysis etc.

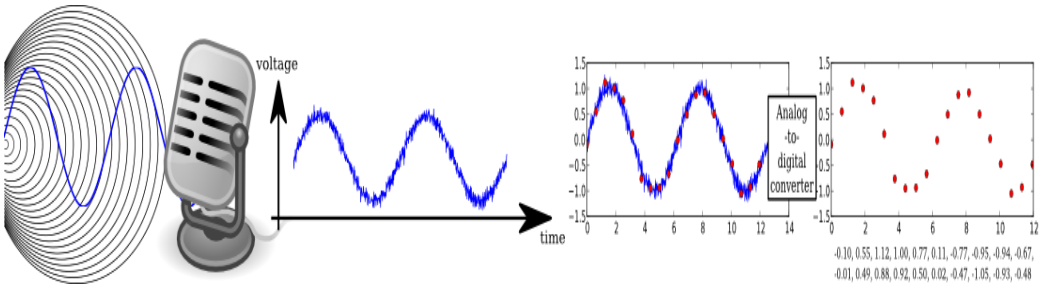


Figure 12. Radio Analog's to Digital Signal [11]

3.7. Problems Faced in Current SDR with GNU Radio for Protocol Controls

GNU Radio by itself can neither be a solution to an application nor can it operate by itself to talk to any hardware according to the communication standard like Bluetooth, LTE, Wi-Fi or Zigbee protocols). It generally needs separate hardware to embed with to hear a certain protocol or radio communication system. SDR takes the analog part of the signal from the radio and moves it physically to the computer that has software like GNU Radio or any other software that has been embedded on it. GNU Radio is the framework dedicated to writing signal processing applications for multiple computers [11]. We know that different SDRs have different capacity to control the protocol. This can sometimes be a disadvantage. This is because not all the SDRs have the best effect of controlling the protocols especially in full duplex. Current research professors and scientists used USRP/ Hack RF in most of the GNU Radio applications to control the protocol or to hear the necessary radio signal. These SDRs consume more power, need daughter board support or can’t operate at full duplex. The solution for this will be explained in further chapters.

4. WIRESHARK PROTOCOL ANALYZER

4.1. Introduction of Wireshark

Wireshark is a free network protocol analyzer. It lets us see different protocol like TCP, HTTP in the network. Since it has the availability of checking the ACK when we capture the protocol, we have the capability to check for packet loss. Wireshark works as a de-facto standard and it captures the packet at a very minute level. Wireshark was first introduced in the late 1990 by Gerald Combs. The stable release of Wireshark was realized recently. It is written in C, C++ level language. The type of OS in Wireshark is the Cross-Platform OS Wireshark is a packet analyzer, and the license used is GPLv2 [13]. Wireshark is similar to TCP Dump but has some GUI plus filtering and sorting options [13]. To ease the debugging and to allow monitoring communications in the WSN, we will use the Wireshark which reads the output in .pcap format that is given by GNU Radio [14]. The Wireshark tool helps us to calculate functionalities like throughput and delay calculations. Wireshark also has built-in statistical plug ins. Though initial releases were mainly for TCP/IP networks, Wireshark also helps in providing the complete framework for any type of packetized network protocol. This helps in our thesis to understand the packets of Zigbee protocol. There are different protocol analysis tools available. One of the other examples of packet analyzing protocol is created by Texas Instruments. It is capable of monitoring one channel at a time and can report Received Signal Strength Indicator (RSSI) and Link Quality Indicator (LQI) that are returned by the hardware platform for each packet. In our thesis for capturing the IEEE802.15.4 protocol, we use the Wireshark protocol tool that gives us the different functions including the IO graph, LQI, number of packets received, and demodulated packets.

4.2. Basic Architecture of Wireshark

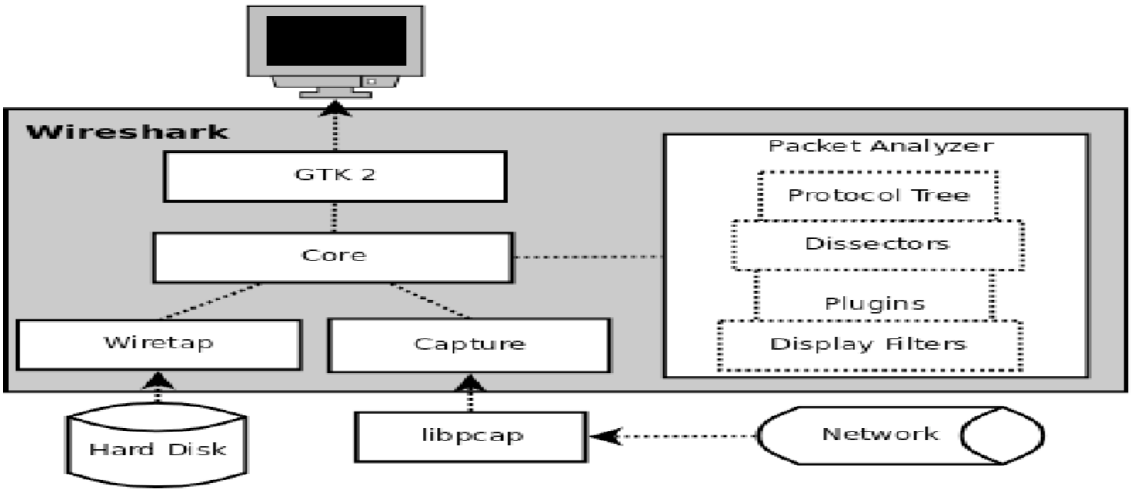


Figure 13. Architecture of Wireshark Protocol Analyzer [15]

The above figure shows the architecture of the Wireshark protocol analyzer, which can be simply explained in the following ways. The incoming protocol traffic from the network is in the form of packets (at the libpcap) and the hard disk is fed into the wiretap and capture where the Wireshark captured the TCP, HTTP packets or any other IoT packets that use IEEE802.11, IEEE 802.15.4 protocols . Libpcap is also one of the open source libraries to interface all the different Network Interface Cards (NIC) [15]. The next block is the core block which helps in building all the network together. GTK is the open source Graphical User Interface of the Wireshark that provides the graphical support to the whole software that helps in generating the necessary graphs for the project. The packet analysis portion of Wireshark performs the logical dissection of packet contents and provides statistic plug ins access to packets as they are processed [15]. In our thesis since we are using IEEE802.15.4 protocol layer, the packet analyses work as follows. The incoming packet is dissected first by a Physical layer and handled over to the MAC layer and then the network layer of the OSI models. Plug ins are

configured and notified about the packet arrival. The plugins help in statistical calculation. These plug in use GTK to create a GUI to display results. These plugins give the packet analyzers the ability to build custom real time visualization of packet activity [15]. The packet analyzer block is the brain of the Wireshark protocol analyzer blocks because it analyzes what type of protocol signal it is and also checks if all the packets are received perfectly in order. Further explanation of IEEE802.15.4 packet analyzing will be provided in the upcoming chapters to show how the analyzer receives the decoded packets in the form of 1s and 0s. The built-in packet analysis feature of different IoT protocols makes Wireshark one of the best options for researchers to use.

4.3. Installation Procedure for Wireshark at Ubuntu

Since we use Ubuntu in our thesis, the Wireshark is installed as follows:

$sudo apt update

$sudo apt install wireshark

$sudo usermod -aG wireshark $(whoami)

$sudo reboot

To run the wireshark after installation, the terminal command should be entered as: **$wireshark or $sudo wireshark** . The connection between the hardware, software, and the Wireshark protocol analyzer will be discussed in further chapters

5. IEEE802.15.4 PROTOCOL AND ZIGBEE APPLICATIONS

5.1. Introduction to Zigbee IoT

Zigbee is an IEEE802.15.4 standard based specification for a suite of high-level communication protocol to create a Personal Area Network (PAN) with small, low power digital radios designed for small scale projects that need wireless connection. It enables low power low data rate proximity wireless ad-hoc connection [16]. It is relatively cheaper compared to the other WPAN IoTs such as Bluetooth and Wi-Fi. The Zigbee protocol standard of IEEE802.15.4 was standardized by the Institute of Electrical and Electronics Engineer in the year 2003. This standard is mostly concentrated on making this IoT for Low Rate Wireless Personal Area Networks (LR-WPAN) that are cheap and do-able for research purposes [16]

5.2. Applications of Zigbee IoT

Zigbee has shown itself as the best use case for many applications such as

- Home control center,
- Wireless Sensor network,
- Industrial area,
- Building automation,
- Smoke Warning,
- Building automation,
- Medical data collection,
- Wireless microphone configuration

5.3. Introduction to IEEE802.15.4 Protocol

IEEE802.15.4 is the technical standard that defines the operation of Low Rate Wireless Personal Area Networks. The standard specifies the Physical and MAC layer. Zigbee uses this IEEE 802.15.4 standard to handle the physical and MAC layer of the stack whereas the Zigbee Alliance that works under the standard usually handles network layer and application layer (i.e., the Zigbee object/message packets, Zigbee Framework and support sublayers). The upper layer that further extends from the standard is used by 6LoWPAN as thread. A thread is an IPV6 based low power mesh networking technology for IP Products that is intended to be secured and encrypted [17]. Thread uses 6LoWPAN is used at IEEE802.15.4 but we could say it as an extended version of the standard with mesh communication like other Zigbee, but it adds IPV6 to be more secure and encrypted. Compared to the other standards used by Wi-Fi and Bluetooth, the Zigbee standard of IEEE802.15.4 requires less bandwidth and lesser power. The standard intends to offer a fundamental lower network layer of a WPAN type that requires very little power to work. The main focus of the WPAN is a low cost, low power, short range and very small size network that uses less Bandwidth. Like we saw earlier, the low rate WPAN being IEEE802.15.4 , most commercial applications use IoT Bluetooth, specified by IEEE 802.15.1. Bluetooth fits in common commercial devices like laptops and mobile phones. Bluetooth is a medium rate WPAN. Though Bluetooth started with IEEE 802.11.1, its standard changed due to the tight security concerns and it is very difficult to decode or encode using embedded devices. Suitable devices for the IEEE 802.15.4 standard include home sensor devices. Some of the characteristics of using this Low Rate Wireless PAN are given as follows [18]

- Star or peer – to – peer connection
- Allotted 16-bit short or 64-bit extended addresses
- Optional allocation of guaranteed time slots
- Carrier sense multiple access with collision avoidance (CSMA – CD) channel access
- Over-the-air data rates of 250 kb/s, 100 kb/s, 40 kb/s and 20 kb/s
- Fully acknowledged protocol for transfer reliability
- Low power consumption
- Energy detection
- 16 channels in the 2.4 GHz band of frequency
- Link Quality Indication

5.4. Node Types in IEEE802.15.4 Protocol

This protocol standard is classified into two different types of nodes . They are

- Full Function Device (FFD)
- Reduced Function Device (RFD)

5.4.1. Full Function Device

FFD serves as a coordinator for a Personal Area Network. It implements a general model of communication that allows it to talk to other devices. It can talk to both RFDs and FFDs. It usually functions as a common node [19]

5.4.2. Reduced Function Device

Reduced Function Devices, commonly referred as RFDs, are usually used as end devices. They have very moderate resource and communication requirements. RFD is eligible to communicate with Full Function Device but cannot coordinate itself like

FFDs. RFD usually consumes much less energy due to its moderate requirements [19]

5.5. IEEE802.15.4 Network Topologies

IEEE802.15.4 network topologies have two main standards. They are:

- Star network topology
- Peer to Peer topology

Though the network has 2 types of topologies, both should at least include one FFD or more to work as a coordinator of the network. IEEE802.15.4 has a 64-bit identifier. Out of 64 bits if the conditions are met, 16 bits can be used within a restricted environment. PAN in its own domain uses its short identifiers that are 16 bits identifiers.

5.5.1. Star Topology Vs Peer to Peer Topology

Basic configurations of a star topology and a peer to peer topology

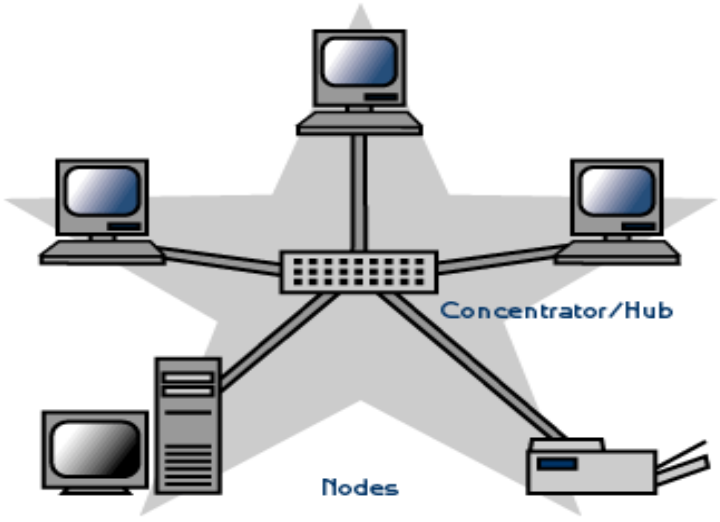


Figure 14. Star Network Topology [20]

In star network topology one node is selected as a PAN coordinator and all other nodes should communicate through the particular coordinator. The Full Functional Device can itself become a PAN coordinator by establishing its own network and

choosing an available PAN identifier for a unique network. The one PAN coordinator usually has more power and will have more computing resources. Usually, Star network topologies are used for home networks or used for home sensors since it uses a single central control point for detection. [18] [19]. In Peer to Peer topology any node can communicate through any neighboring nodes within the particular receiver range. There is still a PAN coordinator but the peer to peer allows more complicated networks . Sometimes the peer to peer network has more than 1 PAN Coordinator. But only the first PAN coordinator usually directs or instructs a device to become a PAN coordinator for a new cluster network. Such complex topology is applied in big industrial applications, big office environments or in inventory tracking. Multiple Peer to Peer topologies will help in formation of a cluster tree network.

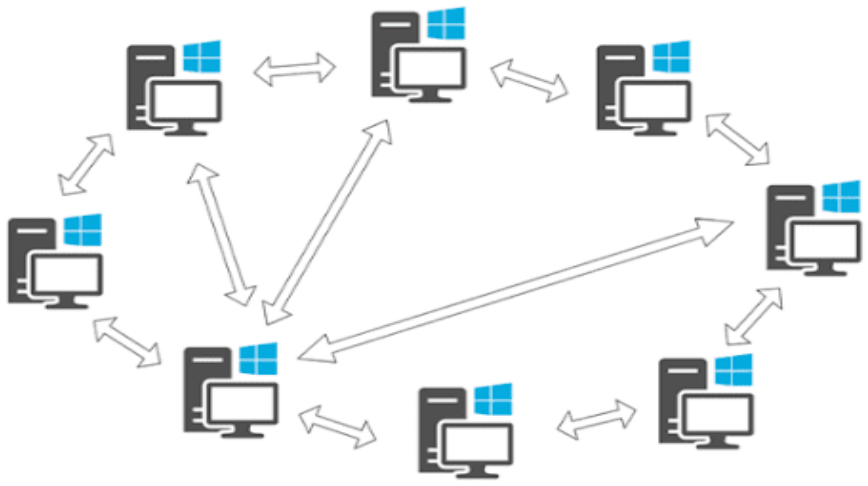


Figure 15. Peer to Peer Network Topology [21]

The main disadvantage in the cluster tree or tree topology is that if one node fails, all the other child nodes also fails and thus the other nodes are disconnected, and it will lose its ability to communicate to other network[19] [18]

5.6. IEEE802.15.4 Architecture

The IEEE802.15.4 protocol has two layers in its architecture. The lower level OSI architecture consists of two layers. They are the Physical layer and the MAC layer. We all know that the devices using IEEE802.15.4 interact through a wireless medium. The wireless medium on the network relies on the OSI layers for better communication [19]. There is a seven OSI layer model in a networking field. Although we use only the first two layers for the working of IEEE802.15.4, communication with upper layers depends on the use of IEEE802.15.4 Logical Link Control. The IEEE802.15.4 PHY and MAC layer along with the embedded Zigbee's Network and the application layer provide the following features [22]:

- Security Encryption
- Low power Consumption
- Reliable data transfer
- Short Range Operation
- Ease of Implementation and Lower Cost

Figure 16 represents the IEEE802.15.4 Protocol architecture and its explanation is given

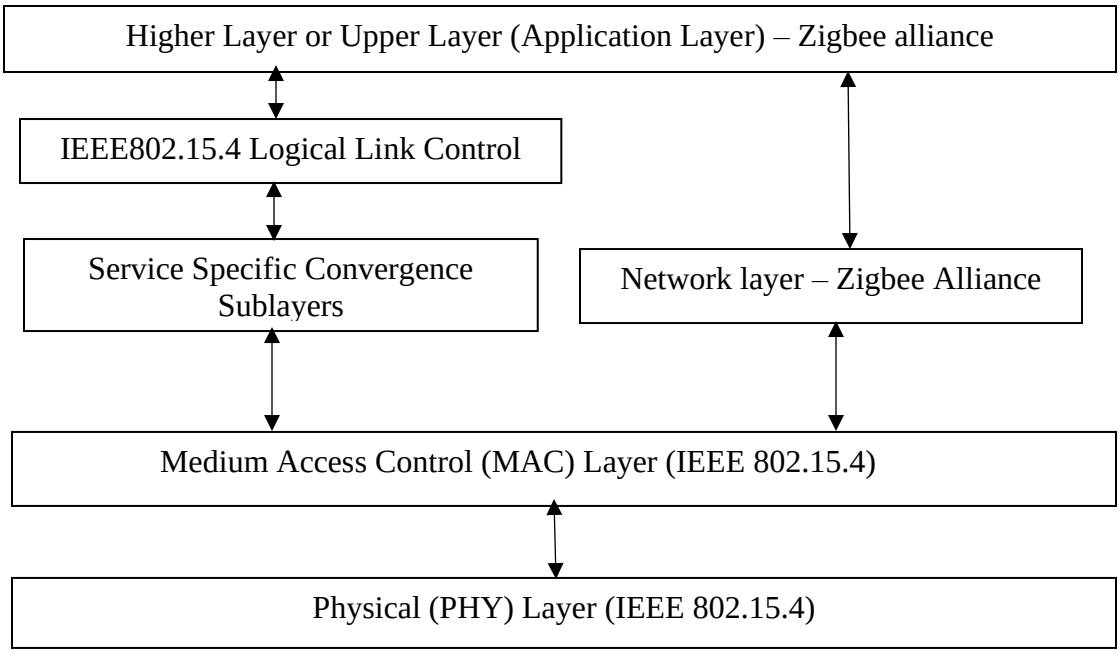


Figure 16. IEEE802.15.4 Architecture Flowchart

5.6.1. Physical Layer (PHY) – IEEE802.15.4

Physical layer is the lowest layer in the OSI Layer model. Physical layer is used for basic transmission and reception. The modulation and demodulation task are performed in this layer for incoming and outgoing signals [23]. Physical layer is the layer that connects directly to the physical medium. It helps the physical medium become introduced to the OSI layer model to perform the desired operations. The physical layer is designated to accommodate the need for a low cost yet allow high levels of integration. Zigbee's use of IEEE802.15.4 Physical networks allows it to handle any number of devices along with the MAC layer. The Physical layer has two services: They are the data service and the management service. The data service handles the transmission and reception of the Physical Data Protocol Unit. The management service is used for calling

various management functions. For example, if the next upper layer wants to read or know what is happening at the protocol stack. The Physical layer uses Direct Sequence Spread Spectrum (DSSS) for data transmission and reception by which each data byte is converted into 32 bits data type and each bit is again modulated for transmission [23],

The Physical layer uses three different frequency bands for IEEE802.15.4 they are:

- 868 MHz – In and around European Countries
- 915 MHz – In and around North America
- 2.475 GHz – Worldwide

Physical layer can also perform other tasks like activation and deactivation of the radio transmission and reception, Link Quality Indicator, Channel Selection, Clear Channel Assessment (CCA) and transmitting and receiving packets in the physical medium. Although it operates in three different frequency bands, we use only 2.4GHz in our thesis [22] [15]. The modulation, demodulation, and power of the total Zigbee protocol depends on the working of the Physical layer. In our thesis, we are going to select the 2.4 GHz frequency band which has been used worldwide. The detailed explanation of how the modulation process of Physical layer works will be given in Chapter 8 which describes syncing Lime, GNU and the protocol. The next layer to the Physical Layer comes the MAC Layer.

5.6.2. Medium Access Control Layer

The Medium Access Control layer acts as the mid layer between the network layer (Zigbee)/Upper layer and the Physical Layer. It is used to handle point to point communication in a network and provides ACK formed by the MAC Layer. The

transmission of MAC frames is enabled by the MAC layer through the use of a physical channel. It was designed to allow multiple topologies without complexity. It can handle more devices without requiring other devices to be in an idle state [23]. Like the Physical Layer, the MAC Layer also provides the Management Service. It accesses the physical channel and hears the message again from the PHY Layer. It acts as an interface between the network beaconing and Physical medium. There are two different modes of operation in MAC apart from its service. They are Beacon enabled and the Non-Beacon enabled. Both uses Carrier Sense Multiple Access (Collision Avoidance). The difference between the Non beacon enabled and the beacon enabled is that the Non-Beacon enabled is the simplest mode and the devices are not usually synchronized, and the beacon enabled requires all devices to synchronize on the super frame, and frames are divided into slots and slots can be assigned exclusively to the device depending upon its latency. The multiple modes of operation are not required under the Power Management in MAC Layer. When it receives the bit, it handles the frame validation, time slots and handles node association. The MAC and network layer of the IEEE802.15.4 protocol offer the hook point for secured services [23]. It helps in the network start up and configuration of the new devices. The IEEE802.15.4 PHY layer supports only up to 127 Bytes. The encryption and roaming are usually handled by the Medium Access Control Layer [19]. The next layer to the MAC Layer comes the Zigbee alliance – Network Level Layer controlled by Rime Stack Application in our thesis.

5.6.3. Network Layer – Zigbee Alliance

IEEE802.15.4 specifies only the PHY and MAC layers. The higher-level layers, including the network layer, Application support sublayer, Security Service Provider, and

Application Object along with the Zigbee device object, are specified by the Zigbee Alliance. The network layer acts as the security layer between the Application layer and the lower level layers. Zigbee routing and Zigbee traversing more than one hop are the main uses of the network layer. Zigbee network layer usually forms the necessary networks. A beacon network frame that has been explained above usually helps in the formation of networks in the network layer and it also determines if other networks are in the range. For transceiving an IEEE802.15.4 protocol, the network layer is used in terms of Rime stack layer, which will have the communication with the Physical and MAC layer. Sometimes , if we are sending the messages to be broadcasted to the outside Hardware like XBee or CC2531 then we say that as application level layer that comes under Zigbee Alliance.

5.6.4. Application Layer – Zigbee Alliance

Application layer in Zigbee usually helps in addressing objects like profiles, clusters, and end points. It helps in binding requests from the end points and facilitates application services. The upper layer, application layer is the next level layer of Zigbee after networking layer that provides network management facilities and also provides the service discovery facilities [23]. The Zigbee application layer usually contains three parts that are application framework, application support sublayers and Zigbee Device Objects.

6. EMBEDDING GNU RADIO WITH LIME SDR

This chapter will provide the plug-in installation instruction and the block explanation of Lime SDR in GNU Radio. The current plug-in is tested and running on Linux Ubuntu platform.

6.1. Installing Dependencies for the Lime SDR in GNU Radio

6.1.1. Initial Command to Install Dependencies

This command should be entered in the terminal box to install the Boost and SWIG, a proper driver to make the Lime SDR work in the GNU environment

sudo apt-get install libboost-all-dev swig

6.2. Building Gr-Limesdr from the Source

The command to install gr-limesdr plugin is implemented by entering the below code in the terminal. The code is:

git clone https://github.com/myriadrf/gr-limesdr

Although the above command ensures only the downloading of the gr-lime sdr plugin, to make, build and install it in GNU Radio, we need to enter further terminal commands [24]. They are:

cd gr-limesdr

mkdir build

cd build

cmake ..

make

sudo make install

sudo ldconfig

All source code is available through myriad website to plug in gr-limeSDR in GNU Radio. Now we need to open the GRC to check if the Lime Suite appears on the block list. The above terminal commands are the installation procedure only for Linux Ubuntu since we use only Linux Ubuntu OS for this thesis. For other OS, the commands are given in the appendix section. The two main blocks that we will see in the GNU Radio block section are Lime Suite TX and Lime Suite RX. Both blocks should be configured properly to have a connection with the Lime SDR USB to get embedded and produce an RF signal or necessary output.

6.3. Toolbar Section of GNU Radio

Figure 17. Toolbar in the GNU Radio [12]

The above figure represents the toolbar section in GNU Radio. There are different sections available as icons in the toolbar. The first icon with the plus symbol represents the “new project” section, and the second icon represents the select / open project, the X at the fourth icon represents closing a particular project. The green icon with the triangle symbol is the important icon that is the “run/execute the project” icon. It helps us to view the graphical result of the thesis. The search icon at the far right helps us to filter the different blocks that are available on the GNU Radio. The generate icon right left to the run icon represents the generation of Python code or generation of Blocks if either one of them is ready. It is mainly used to create code from the blocks. The heir blocks are one of the important blocks of GNU. They are also known as the built-in blocks. There are four important section that are needed to enable the use of heir blocks. These will be explained further when we describe PHY layer in IEEE802.15 later in this thesis.

6.4. Lime Suite TX Block in GNU Radio

Figure 18 represents the Lime Suite Sink (TX) in the GNU Radio environment

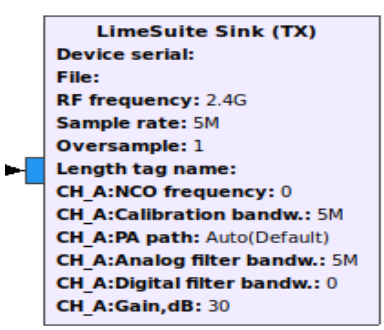


Figure 18. Lime Suite TX Block in GRC (Taken from our thesis)

We can see the multiple parameters that have been designed using Python and C++ in the GNU Radio for Lime SDR hardware.

6.4.1. Properties and Settings of Lime Suite TX

The next step is to double click on the Lime Suite sink (TX) or right click and go to the properties section. Then a property dialog box appears below that has been shown in Figure 19 as a screenshot

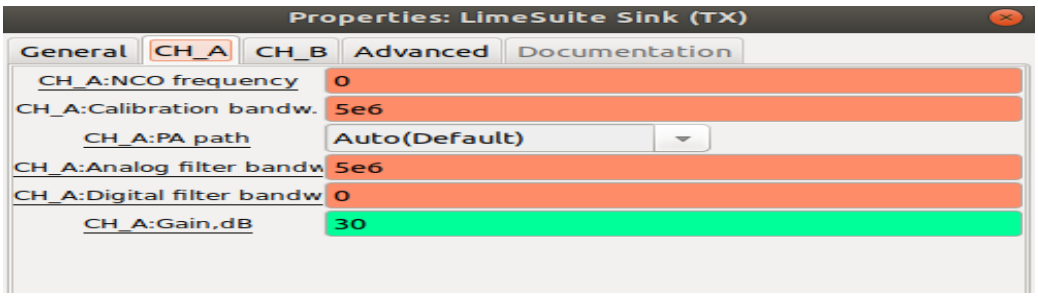


Figure 19. Properties Block of Lime Suite TX

There are 4 sections that are currently included in Lime Suite sink (TX) packets. They are General, Channel A, Channel B, Advanced section, and a separate documentation tab.

The general tab consists of the ID section, the device serial, the file to upload, the channel section, Radio Frequency, Sample rate, Oversample, and Length Tag name. The ID tag in the general section of the GNU Radio represents the ID name given to the particular project. It is generally the default and left unchanged. The device serial represents the use of selected hardware with multiple devices available on our personal computer. The command line for filtering the particular device is given as **"LimeUtil –find**". Next comes the file section which we use to load an encrypted file that can be read by the GNU Radio. For example, in the previous chapters we saw the .ini test file to upload and later configured it in Lime Suite GUI. There are 2 channels generally in the Lime SDR. They are Channel A and Channel B. Note that there will be two channels in both the Lime Suite RX and the Lime Suite TX. The important point to understand is that we have to select the same Channel and same file for both the Lime Suite RX and TX to make it work successfully. Next comes the important section called RF Frequency. There are different types of Lime that use different bands of frequencies. Generally, in our thesis we use Lime SDR USB. So, we are maximum allowed to use 100-3.8GHz of RF center frequency that helps in supporting our project. Although the frequency we allot in our thesis depend on the range of the protocol that we use. The Sample rate section in Lime Suite TX helps us to select the different number of samples per second depending on the source it carries that maybe audio or binary files. It's generally measured in terms of Hz or kHz. For the Lime SDR USB device, we should not fix the sampling rate to be more than 61.44 MS/s. Like the channel and file, the sample rate between the Lime suite source (RX) and Lime suite sink (TX) should match with each other. The next section is the oversampling, which is the process of sampling a signal at higher than the Nyquist

rate. Oversample in this Lime suite TX block lets us sample the TX signal at a higher rate than would be required to just preserve the desired signal frequencies. The increment in the oversampling rate will be dependent on the sampling rate value that we use [24]. Since we use Lime SDR USB , the possible oversampling value for the sample rate range 30.72<x<=61.44 MS/s would be 1,2,4, or 8 , for the sample rate range from 15.36<x<=30.72 MS/s would be 1,2,4,8, or 16, and for the sample rate x<=15.36MS/s would be 1,2,4,8,16, o r32. [24].

Generally, Channel A and Channel B have same section in the Lime Suite TX and RX. Since Lime has the capability to transmit and receive simultaneously there is no necessity to use both the channels, although the channel that is used should be same in RX and TX. Since we have obtained successful result using channel A only, we consider the sections in channel A here. The first section is the Numerically Controlled Oscillator Frequency. It is commonly called the NCO frequency. It helps in the creation of a discrete sinusoidal waveform acting as a digital signal generator. Usually we keep it in "0" which is the OFF state. The length tag name section helps us to set the name of the incoming tag. It tells us how many bursts it has, the length of the burst, and where the burst is located. The next section is the Calibration bandwidth. Like NCO frequency, if we set it to be 0 then it's in the OFF state. The calibration bandwidth in our thesis is generally set to 5MHz. Note that the calibration bandwidth should also be same for the Lime suite RX. The general acceptable range of Calibration Bandwidth must be from 2.5 MHz to 120 MHz. The PA path section of the Lime suite TX helps us to select the active power amplifier of each channel. The possibility can either be auto, Band 1 or Band 2. Usually in our thesis since we use Lime SDR USB we use the Auto (Band) for the PA

path section. PA path is the callback function value. The next two sections are the analog and digital filter bandwidth. The TX analog frequency usually ranges from 5 MHz to 130 MHz. 0 represents the off state. The TX digital frequency usually has a range that should not be more than half the sampling rate provided in the Lime suite TX and RX. Gain in TX controls the TX channel gain. The gain for our GNU Radio and Lime usually ranges in between 0dB – 60dB. [24]. TCXO: DAC value at the advanced tab settings modify the DAC parameter value used to calibrate the reference clock. Enabling the parameter in advanced tab is to enable the parameter we should go to, ”allow TCXO DAC Control“ in the “Advanced” tab and it must be set to “yes”. When the power is off, the DAC return to its “OFF initial state”. For the Lime SDR USB in our thesis, the default value is 125, and the range is [0,255] [24]

6.5. Lime Suite RX Block in GNU Radio

Most of the sections are similar to the Lime suite TX sections, but there are some sections where the range of values changes between the Lime Suite TX and RX. The analog filter BW value range can be between 1.5MHz – 130 MHz. The RX digital filter bandwidth should not be higher than half the sampling rate like Lime Suite TX. The gain value can be between 0dB to 70 dB. The oversampling range also has a change. For the sampling rate range between 30.72<x<=61.44 MS/s , the possible oversampling value should be 1 or 2. For the sampling rate of range between 15.36<x<=30.72 MS/s, the oversampling value can be 1,2, or 4. It goes up to 1,2,4, or8 for the range 7.68<x<=15.36 MS/s and up to 1,2,4,8,or 16 for the range 3.84<x<=7.68 MS/s . The final range is X<=3.84MS/s, where the oversampling rate could be 1,2,4,8,16, or 32 [24]. Note that these ranges are fixed and applicable only for Lime SDR USB and Lime SDR – PCIe

type hardware devices. Unlike PA path in Lime suite TX, the Lime suite RX has LNA path. It helps in reducing the noise of the amplifiers thus acting as a low noise amplifier. Although Auto (Default) mode is available in LNA path, the Lime SDR USB hardware type generally uses one of the following selections: LNAH, LNAW, LNAL[24].

6.6. Documentation Tab and the Other Works of Gr-Limesdr with GNU Radio

General information about each section is given as a definition in the documentation tab .Once the initial set up has been done in the GNU Radio , the gr-lime SDR will be available any time to use for any projects that we work on. The radio's main help is having itself get connected to a particular IoT protocol and to hear messages from that protocol. Thus, by having its connection with GNU Radio, we can make its connection to the heir PHY and MAC layer of the IEEE802.15.4 protocol in our thesis. Though there are several research papers with IoT and different SDR, the use of the new Lime SDR with the IoT protocol makes it cost effective and helps in transceiving the packets without any loss. The other research work done by other developers will be seen as the literature review in the next chapter below

7. LITERATURE REVIEW

7.1. Previous Research Work

There are several papers researching applications of the IEEE802.15.4 standard at different stages. We need to deep dive into these papers to understand the basics of the Zigbee standard and the needs of researchers for a flexible, low-cost IoT test bed. This chapter highlights some of the important papers that help in the research of IEEE802.15.4 test tool transceiving using GNU Radio plus Lime SDR.

In the paper [6], " Using Lime SDR for RF measurement", the author observes that the Internet of Things is driving the need of wireless connectivity to unprecedented scale. He explains the main reason behind the market disruption in the study and development of new IoT systems is the high cost and risks of ASIC development. The paper also explains that one way to reduce the cost is to create evaluation systems based on low cost SDR tools each making its own work in IoT fields. The author also gives the basic idea that new IoT and Wireless communication systems can be produced by small scale industries with the help of SDR tools [6] . However, the author limits the SDR only for making RF measurements in his paper. This paper underutilizes the SDR for RF measurements but not directly for the larger, more significant job of IoT development. Even though the author focuses exclusively on RF measurements, he uses the best software tool that is the Lime SDR for the measurement of RF transceiver from LMS 7002M FPGA Transceiver that covers from 100kHz to 3.8 GHz. The author explains that a variety of RF measurements can be performed by reconfiguring the flexible structure of Field Programmable RF (FPRF) chips in SDR platforms. The author explains the architecture and features of Lime SDR platform in the context of RF measurement. The

author explains that LMS7002M can be internally reconfigured for FDD or TDD. In FDD mode, the TX and RX are at different frequencies and in TDD mode the TX and RX are at the same frequencies. The author explains the use of FDD and TDD both at the same time can be helpful in low power consumption and ensures TX-RX phase coherence used for phase measurement [6]. The author limits his research with RF measurement by using three important functions

They are :

1. Basic Spectrum Analyzer and Signal Generator

2. Two Tone IP3 Measurement

3. Scalar and Vector Network Analyzer.

The author concludes his paper by demonstrating that many RF measurements can be performed by configuring Lime SDR in different operating modes such as FDD and TDD utilizing the resources such as TX/RX, NCO digital filters etc. [6] The paper ends by claiming that its work is helpful in enabling individuals and startups to develop and debug wireless products and disrupt the IoT market [6].

In the next paper the Universal Software Radio Peripheral (USRP) SDR is tested on a Narrowband-IoT (NB-IoT) type protocol [25]. The author explains about the Software Defined Radio for generating the downlink signal according to LTE R13 Narrowband IoTs (NB-IoT) specs. This is one type of specific IoT system used in the market, like Zigbee and Bluetooth. The IoT used in this paper is based on the Licensed Ni-IoT standard finalized in mid-2016 by 3GPP [25]. For testing this type of IoT, we need to have a vector signal generator capable of generating the NB-IoT DL OFDMA signal.

Then the paper explains about the NB – IoT six PHY Physical signals and channels, namely [25]

1. Narrowband Primary Synchronization Signal (NPSS)
2. Narrowband Secondary Synchronization Signal (NSSS)
3. Narrowband Reference Signals (NRS)
4. Narrowband Physical Broadcast Channel (NPBCH)
5. Narrowband Physical Downlink Control Channel (NPDCC)
6. Narrowband Physical Downlink Shared Channel (NPDSC)

These are some of the important physical signals and channels that may be used for developers who choose this IoT for their future work. The author concludes by saying that the NB-IoT operation mode is standalone, the same VSG flow can be applied to in band and guard band modes [25]. Another point made in the above paper is the NB-IoT protocol is still rapidly changing. The authors claim that the rapid changes reduce the market acceptance of the protocol as small-scale companies are not ready to incorporate such a fluid system. Another major problem is that the SDR used in the paper is utilized only as a half – duplex hardware downlink vector signal generator. A problem in paper [25] is the maximum bandwidth limits itself to 180 KHz, where the transmission scheme remains the same as LTE, which is OFDMA for downlink and SC-FDMA for uplink vector signal generator. Narrow bandwidth and half-duplex limitations will not be present in the LIME SDR [25]

Reference [25] does not address the need to change hardware tools for configuring different types of IoTs - the USRP hardware tool limits itself for NB – IoTs whereas the Lime SDR acts as an open hardware tool for changing modulation

techniques and multiple access scheme for different type of IoTs. The paper does not explain how a modulation technique can be changed with the help of SDR for different types of IoTs. The USRP SDR platform used here has a problem of less frequency range and bandwidth and higher cost compared to the Lime SDR thus making the Lime SDR, which is going to be used in the upcoming thesis, a better tool. In the framework of an LTE system and a specific protocol, [25] uses an USRP SDR tools to solve the problem between the IoT devices and the smartphones. We propose a much more flexible SDR toolset with the software GNU Radio system.

In a paper titled ”Reconfigurable IoT gateway based on SDR platform”, the author addresses a clear issue that’s hindering our current IoT systems. He states the current enabler of IoT is short range terrestrial communications embedded on affordable, low-power wireless devices that often do not provide connectivity to a specific gateway that enables them to communicate with the world [26]. The author also mentions clearly that the SDR will be the best solution for this by proposing an architecture for IoT based on SDR which includes extra blocks like a GPS module or GPS interface capabilities and yet has reduced total power consumption. Proposed new 5G waveforms are attempting to use the existing network infrastructure to get more users and devices and to squeeze out more bits per Hertz [26]. The author’s main goal in the paper is to analyze the SDR and how it is used in a gateway and to describe its implementation in communicating with IoT applications and to successfully forward the data to WAN or terrestrial cellular networks [26]. Some of the common IoTs like Bluetooth, Zigbee and Bluetooth Low Energy and Long-Range WAN can be reconfigured by the design of the gateway using SDR.

SDR also plays a major role in the field of vehicular communication [26]. The author provides an advanced architecture that helps in reducing the power and simplifying the analysis of existing SDR implementation and performs tests that show positive results in terms of connectivity with sensor nodes, both for the COTS (Commercial off the Shelf) modules and for the software implemented protocol stacks for Zigbee and Bluetooth. The author realizes the communication standard using MQTT protocol [26]. Though the author describes the outline of reconfiguring IoTs using SDR, it is just the classical system that has been improved with a GPS receiver on it. The authors conclude by stating that the future work will include the improvement of user interface, extensive tests on the combined use of software – implemented protocol stacks versus COTS components. The paper doesn't speak about transceiving of a particular protocol.

In Reference [27], the author shows the demo architecture of a Wireless IoT platform based on SDR technology that gives us the general idea of how the implementation of Wireless IoTs works in SDRs composed by the Wireless Edge Appliance (WEA) as the base station and the Wireless Access Appliance (WAA) as Customer Premise Equipment .The WEA acts as a laptop and through the unique optimization technique the signal processing is completed with high speed whereas the WAA is the low power and low cost embedded system. The author mostly shows the general demo scenario that helps us in our current thesis work to build the communication infrastructure in IoT applications [27], by improving its flexibility, stability and reliability.

The above papers show research on the basics of either SDR or the protocols in IoT development. These papers provide an understanding of the purpose of using SDR in measuring RF and in IoTs and reinforce the need for a flexible, low-cost, SDR-based IoT testbed. The following papers provide details about the GNU Radio and its work with the IEEE protocols and SDRs. One of the important research papers that we have gone through is done by the UCLA research team naming, " GNU Radio Encoding and Decoding" by "Thomas Schmid" at NESL. The author in this paper gives the general report on the implementation of encoding and decoding blocks and also verify the implementation by sending and receiving messages to and from actual IEEE802.15.4 radio, using a transceiver chip and he gives the solution out of it [9]. He explains in detail about the IEEE protocol standard like our thesis, but he uses USRP instead of our hardware SDR. The main disadvantage in USRP is that it is half duplex and need both the channel to work at a time. Thus, it can create more packet loss.

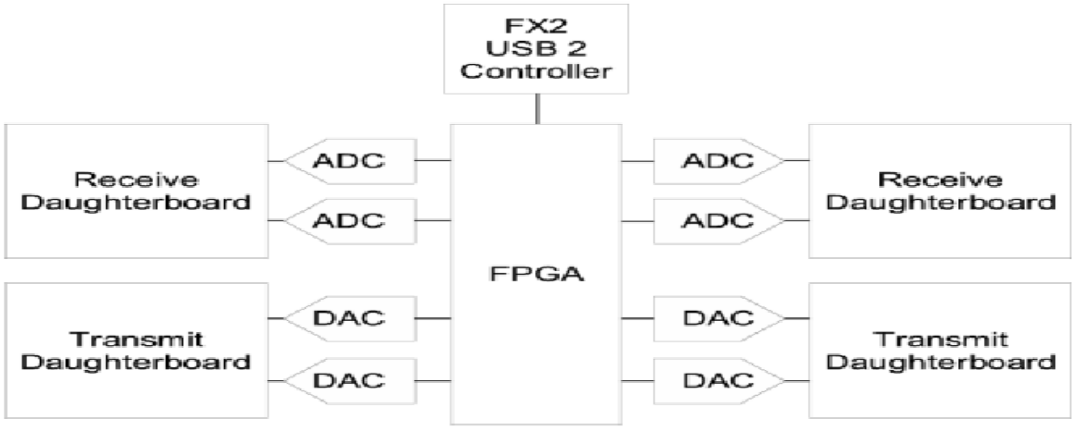


Figure 20. USRP Block Diagram [9]

The author uses a different SDR which is also a low-cost tool but a half-duplex SDR tool in his thesis. It consists of a mother board which holds the ADC, DAC, and a

FPGA to work on the Bandwidth , power consumption and data rate calculation. The further explanation about USRP has been given by mentioning that there will be 4 high speed ADC connections which are 12-bit analog devices AD9862 and have 64MS/s whereas the DAC has 14 bits ,128 MS/s and allow to generate signals up to 50MHz. He states that the antenna is connected to either one of the daughter board side A or side B . These 2 boards of side A and side B usually combines to work on the applications like walkie-talkies, Wi-Fi, or IEEE802.15.4. The MUX is the brain of the USRP because it acts as a router to send different ADC output signals to the corresponding digital down convertors. Like Lime SDR , USRPs are also configured from python [9]. He further explains about the USRP connection to PC and says how two channels with their daughterboard helps in Decoding and encoding IEEE802.15.4 protocol through GNU Radio. Here the transmission and reception are performed separately next to next where one goes in OFF. The author uses 2 spreading sequence for the PHY layer of IEEE802.15.4 one is using OQPSK whereas the other spreading sequence using MSK [9]. The main disadvantage in the above paper is that they use the hardware USRP . Even though USRP has proved its work in the previous research work , the technology has been developing day by day. So, researchers are moving forward to a standalone hardware that don't necessarily need a mother board because to reduce cost and also need to work on both transmission and reception in a single channel. So, we do have a better solution in our thesis for replacing the current USRP hardware [9]

One of the IEEE papers that has been updated with the above paper has been written by Bastian Blossel at the University of Innsburg at the Austria. He developed a GNU radio based IEEE802.15.4 testbed using USRP equipment with a full transceiving

technique. He made a change to the GNU Radio protocol world by providing code as open source software which is fully interoperable with off the shelf sensor motes (TelosB) and Contiki sensor mode OS from the physical layer to the network stack [14]. For the PHY layer he took the USRP SDR radio that communicates with the MAC layer and the upper level network layer of Rime Stack. Rime is a modular, light weight network stack which is part of Contiki OS [14]. Contiki is in turn a state-of-the-art OS for research in Wireless Sensor Networks. He took the open source OQPSK physical layer developed by the above paper by Thomas Schmid in 2006 and implemented a receive and transmit chain and verified the work with Crossbow MicaZ motes. There are some papers that focused on only the PHY Layer whereas some papers gave importance to the MAC Layer. Blossel has represented and given equal importance to both the PHY and MAC layers of IEEE802.15.4 and has provided a way for future research people for extending the research. In his paper he used two USRP N210 SDRs from Ettus Research. Since it is USRP , he added it with an XCVR2450 daughterboard as a radio front end. These daughterboards can operate on the 2.4GHz ISM band in half duplex mode. The blocks are connected to the physical layer encapsulated in the heir block that contains most of the modulation process. Once the transceiving becomes successful , he sends the message to the TelosB sensor mote that has Texas Instruments chip to make the evaluation via the Contiki firmware. The main disadvantage of this project is producing a combined transceiving signal from the transmission and reception [14]. The author himself says it is not as easy as it seems because he uses an USRP SDR. The main disadvantage is that since USRP is half duplex it has to switch between the half duplex and full duplex for receive and send mode of every packet that has been sent. But the switching happens

automatically inside the USRP. The author says the switching from receiving to transmitting is not a problem but the other way around it is. The author explains this issue clearly by stating that when a packet is being sent and the sample stream to a device stops, the USRP first assumes an under-flow control occurrence, that is the PC cannot deliver the samples fast enough. So, the device does not switch back to receive mode immediately but instead waits for a time out to occur [14]. The problem is not during the time when the packet gets received in wireshark as a .pcap file, but when it's communicating with the actual hardware like TelosB motes, the time out period is so long . The another disadvantage of this project is the wireshark may face some packet loss because of the half duplex issue in USRP and if one channel at the USRP fails it does not have any ways to pick up with the other channel since it needs both the channel A and B at the GNU Radio to work perfectly [14]. The solution for this issue has been introduced in our thesis using Lime SDR hardware . The above paper provides a tool that allows the rapid prototyping of new physical layers and shows a way for research people with its PHY, MAC and network stack. This helps future researchers by enabling modifying the hardware SDR or updating the blocks in the GNU Radio software. The usage of Lime SDR in the above projects will make a change in the evaluation of future protocols.

In a small section of her thesis paper, Ms. Chaithra Radhakrishna at Texas State University provides a clear understanding of the work of Zigbee protocol. In her paper the author explains the Zigbee Architecture and its layers. The author further explains about the nodes and topologies in Zigbee and further explains about the Zigbee Device types that we have seen previously. The main aim of the thesis research was to use the

IoT based Management Portal for Augmented Reality Applications. Since the thesis research involves Augmented Reality, she shifts the research focus more into networking and application protocols such as AODV Routing, Many-to-one Routing, and Source Routing , [23] The author uses Digi XBee S2D modules to make two XBee communicate with each other by installing a XCTU application platform. The author explains further about how two XBee communicate with each other through XCTU Console by sending messages to the Zigbee Coordinator and receiving back the messages at the Zigbee router. This paper does not include any use of our GNU Radio or SDR but explains clearly how the communication happens in the Zigbee level networking and application layers and IEEE802.15.4 level Physical and MAC layers [23]. This paper helps us in better understanding the IEEE802.15.4 protocol that we use and also helps us to know that IoTs can be used for Augmented Reality Applications, too. So, evaluating a protocol has good impact in the protocol world and each and every step will help us pave the way for future developers.

8. TRANSCEIVING IEEE802.15.4 TESTBED USING LIMESDR

8.1. Introduction to IEEE802.15.4 Transceiver

The GNU Radio along with Lime SDR can perform many transmission and reception functions. The Lime has a full capacity to not only receive the Radio Frequencies but also to control different protocols. It helps in modulating and demodulating the messages that pass through different protocols. Though many of the transmitting and receiving operations are performed individually, some protocol operations can be performed by combining transmission and reception and generating a file at the end according to our desired output format. This SDR test bed is based on the open source IEEE802.15.4 generated and created by Dr. Bastian Bloessl [14]. The author's code is based on Dr. Thomas Schmid code from UCLA [9]. The main difference is both the author used the USRP hardware along with a daughter board whereas we are using gr-limesdr source and gr-limesdr sink in the thesis. We make a change in total hardware and requirements according to it in the thesis to have better uplink and downlink at lower cost with Lime SDR. The Lime SDR is a more advanced, full duplex SDR and it reduces the packet loss when the transceiving operation happens. At present many popular industries are based on IEEE802.15.4 protocol and it's very important to know the protocol's operation in a detailed way before describing our testbed. The whole transceiver follows the OSI layer model that helps in controlling the entire device and the system. Since we already saw the outline of the Physical and MAC layers that we are going to use in the transceiver system, we can directly jump into the work of the Physical and MAC layers in this particular architecture. We have already described the work of Network Topologies, Type of Device etc. at the physical layer level. Now we can see

how the transceiver at GNU by Lime SDR controls the whole protocol with its modulation and demodulation techniques.

8.2. Basic Work of Lime Suite Source and Sink

8.2.1. Lime Suite TX and RX Source and Sink

Once we install the Lime Suite Source Rx and Lime Suite sink TX like mentioned in the chapter 2.6, and once we reload the block in the GRC, we can now see the Lime Suite Source and Lime Suite Sink in the GNU Radio. It is the main hardware that is going to control our protocol stack in the GNU radio. Lime SDR as a hardware helps in controlling the wireless medium in the flexible manner. It helps in working on ADC to DAC and vice versa to implement the sample and send it to the monitor screen on the Host PC. A usual GNU Radio blocks with the SDR can be represented as follows

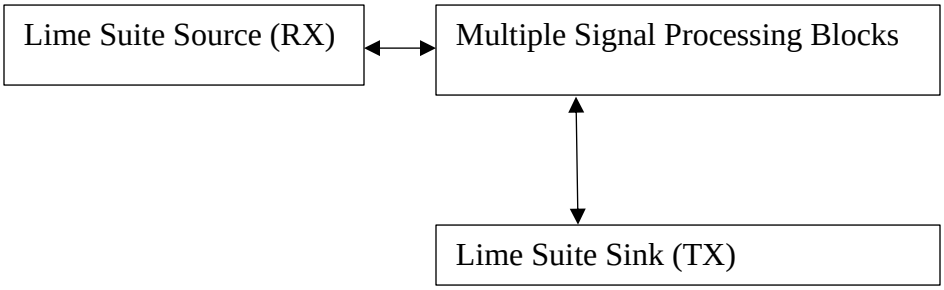


Figure 21. Outline of GNU Radio Represented in Flowchart

The above figure represents the outline of signal processing blocks in GNU Radio. The initial block that is the PHY layer will directly talk with the physical medium that is our Lime Hardware. The Lime interacts with the gr-limeSDR source and sink to understand in which Radio Signal the protocol is going to pass through. The middle block is the signal processing block that performs various operations like data processing , data modulating and demodulating depending upon the type of flow graph we initiate. Gr-Lime SDR Source (RX) generally receives the modulated signal from Gr-Lime SDR Sink

to send out the message for demodulation. It helps in getting the message from the signal processing blocks and transmits it into the Lime Suite Source RX for further modulation. Since we use Low Power Wireless PAN, we are using 2.4GHz of frequency in both gr-limeSDR source and sink blocks. Please note that since both the blocks act as IoT hardware to work on the protocol, both the SDRs need to be in same frequency and same channel to pass the message through air. Now let's develop and describe the whole flow graph that we are going to initiate and see its individual work and the total operation to control the IEEE802.15.4 Protocol.

8.3. Background Information of the Blocks and its Working Procedure

Since we already saw the working of gr-limesdr suite and sink, we can directly jump into the other signal processing blocks that help in controlling the IEEE802.15.4 protocol. All Low Power Wireless PAN usually follow the OSI layer architecture. There are seven layers in the OSI model. Each layer has its own capacity in its work and the control over the layer above. Example: Zigbee, Bluetooth etc., Each has its own standard of usage. The standard was initialized by the IEEE. Like we saw in previous chapters, Bluetooth uses IEEE802.11 whereas Zigbee, 6LoWPAN, and Wireless HART use IEEE802.15.4 protocols. In 6LoWPAN and HART additional features and direct IP connectivity have been added. We use IEEE802.15.4 of Zigbee alliance in our thesis. The protocol generally helps to share or transfer data among users within a range of, generally, 10m to 100m. The advantage of using it is it consumes low power compared to other wireless WANs and its low cost in the market. The IEEE802.15.4 uses the first 2 layers. the Physical layer and the MAC layer, to transmit a message from one node to the other node. Say for example: If we are transmitting a message from node 1 to node 6, all

other individual nodes from node 2 to 5 follow the same protocol in which the message might be in the encrypted format, plus each message will be in the form of frames with the header on it so that the end node can identify which node the particular frames of message came from and decrypt it accordingly. Data that is coming from the application layer, say for example an alliance of an IoT, is encapsulated by each sublayer of the OSI layer that is the MAC and PHY layer for the particular protocol's standard frame format.

8.4. Work of PHY Layer in IEEE802.15.4 Protocol

The Physical layer is the lowest level layer of the OSI model and the first layer in the IEEE802.15.4 protocol that directly communicates with the gr-limesdr (lime suite RX and TX) and transmits the encoded messages accordingly to the Lime Suite TX. Although we have been introduced to the physical layer in the previous chapters, let us see the performance of the Physical layer as a signal processing block. Since we have described the topologies and the device of IEEE802.15.4 in chapter 5, we can directly set our focus on the working of the Physical layer in the particular protocol. Physical layer provides the encoded data transmission service to the physical hardware under the frequency of 2.4 GHz (worldwide) using the DSSS format. For 2.4GHz, it uses channels 11-26 (16 channels total) and the best channel according to the researches will be channel 26. We use channel 26 for our thesis to avoid collision interference with other WPAN that operate at the 2.4 GHz. The BW of the channel is 2MHz and the channels are spaced at 5e6 (5MHz) in our thesis. The Physical layer has the capacity to use different modulation techniques like MSK (Minimum Shift Keying), BPSK (Binary Phase Shift Keying), ASK (Amplitude Phase Shift Keying), and OQPSK (Offset Quadrature Phase Shift Keying). Table for symbol to chip mapping in OQPSK is given below:

Table 2. Symbol to Chip Mapping (Redrawn from [9])

Symbol (Dec)	Symbol (Bin)	Chip sequence (Binary)(c0,c1,c2,…..,c31)
0	0000	11011001110000110101001000101110
1	1000	11101101100111000011010100100010
2	0100	00101110110110011100001101010010
3	1100	00100010111011011001110000110101
4	0010	01010010001011101101100111000011
5	1010	00110101001000101110110110011100
6	0110	11000011010100100010111011011001
7	1110	10011100001101010010001011101101
8	0001	10001100100101100000011101111011
9	1001	10111000110010010110000001110111
10	0101	01111011100011001001011000000111
11	1101	01110111101110001100100101100000
12	0011	00000111011110111000110010010110
13	1011	01100000011101111011100011001001
14	0111	10010110000001110111101110001100
15	1111	11001001011000000111011110111000

Now let us see what Phase Shift Keying is to understand the above mapping better. PSK modulates the data by changing the phase of a constant frequency reference signal [28]. Though the IEEE802.15.4 also uses MSK, BPSK, CSS concepts, we in this thesis focus on OQPSK because OQPSK has reduced interference and reduced noise issues. Usually the signal is modulated either by changing the sine or cosine pulse at a precise time. Mathematically QPSK is double the data rate compared to BPSK or it can

have the same data rate in half the Bandwidth. The QPSK works as it comes under the PSK but the problem with QPSK is that it has a state transition through zero means it does changes its phase by total 180 whereas OQPSK reduces the transmitter complexity and inefficiency by maintain its phase shift by 90 degree. This is done by delaying the offset by 2 in GNU Radio. Quadrature in a graph represents the four symbols say 00,01,10,11. These are separated according to the odd bits and even bits. Duration (T) is the time interval related to a single bit, for total time for sending one odd and one even bit will be 2T. In OQPSK if we consider the d(t) bits to be 100110110100 for example, the odd bits and even bits separate with each other and the odd bits are mapped to Di(t) (an in-phase component represented by cosine pulses) and the even bits are mapped to Dq(t) (a quadrature component represented by sine pulses). In terms of transmitting the symbols 00,01,10, and 11, the most significant bit of each symbol is mapped to the in-phase component whereas the least significant bit is mapped to the quadrature component. The final output at the OQPSK will be obtained with the time interval of t/2, where small t is the symbol time and in quadrature graph , if a first symbol takes 00, the next symbol would be 10 where the In phase changes from 0 to 1 and the next would be 11, 10 to 11 – in which the quadrature component changed from 0 to 1 at 0.5 Ts. In this case, a maximum of only 90-degree phase shift can occur which in turn reduces the amplitude fluctuation and increases spectral efficiency. Let’s see in the diagram below

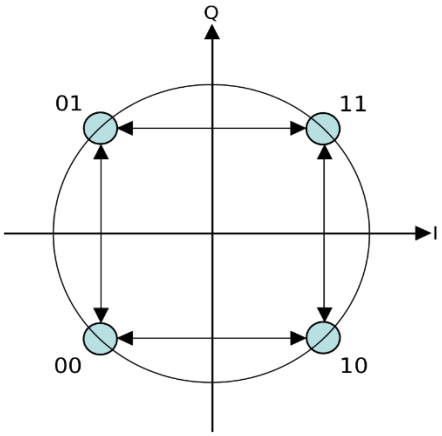


Figure 22. I and Q in Quadrature Graph [28]

In the above figure 22, as we saw earlier, phase can move no more than a 90 degrees in a single shift, which occurs when the 0 of either I or Q changes to 1 or 1 of either I or Q changes to 0 at next point in each index

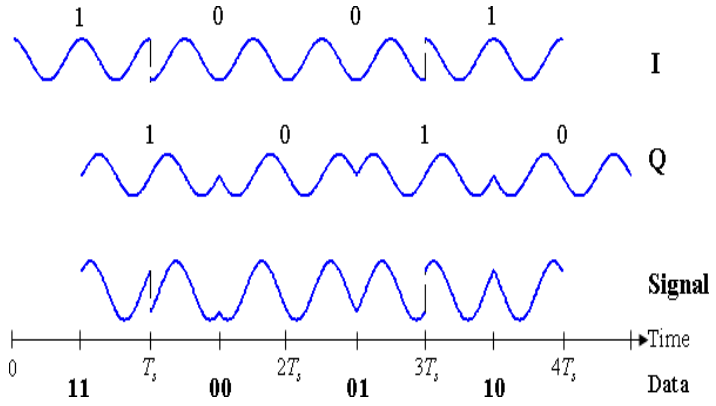


Figure 23. I and Q at Output s(t) in Graph [28]

In the next figure 23, we can clearly see that the I(t) and Q(t) are staggered by the interval 0.5Ts where if one I is 1 the other is 0 , shifted at 90 degree , with no huge phase shift like QPSK. In our thesis, at the frequency of 2.4 GHz, we transmit the data at the rate of 250 Kbits/sec, as shown in Table 3.

Table 3. Frequency, Symbol, Bit Rate, Chip Rate of the Protocol

Modulation	**PHY data rate(Kb /s)**	**Symbol rate (Ksymbol/s)**	**Chip rate(kchip/s)**	**PHY (Freq)**
BPSK,ASK,OQPSK	20,250,100	20,12.5,25	300,400	868/915
BPSK,ASK,QOPSK	40,250	40,50,62.5	600,1000,1600	868/915
OQPSK	250	62.5	2000	2400-2483.5

The above table shows the from the physical layer level frequency band to the bit rate, symbol rate , chip rate per second used by IEEE802.15.4. Each one has its own modulation with their parameter. There are total of 27 channels in 3 types of frequency band [18]. Since we use 2450 MHz center frequency, the channel ranges from 11-26 where the 26th channel has less interference with Wi-Fi that we are going to use for our thesis. The center frequency of 2450 MHz is achieved from the 2400-2483.5 widespread frequency band for the protocol. This exact frequency is obtained by the formula,.

Equation 1. $Fc = 2405 + 5(k-11)$ in MHz, for $K = 11$ to 26 [18]

Where, K is the channel number and since we use $K = 26$ for our thesis we substitute $K=26$ in the above formula, we get the center frequency as 2.4 GHz .

8.4.1. Frame Allotment in Physical Layer

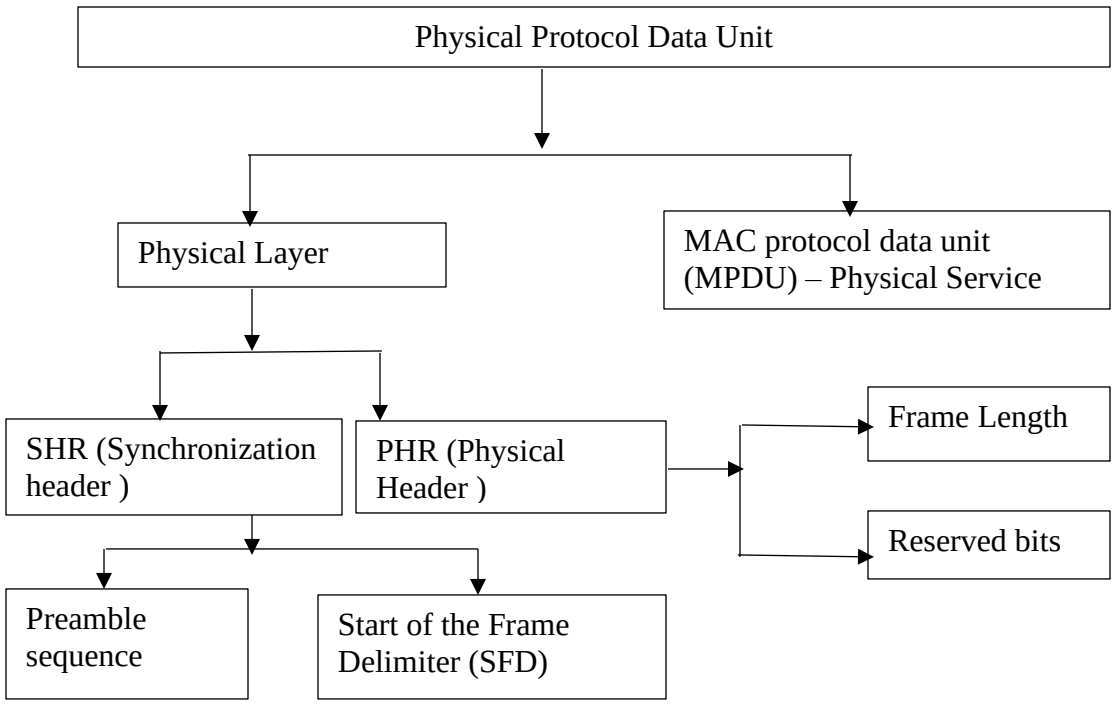


Figure 24. Physical Layer Frame Structure of IEEE802.15.4

The above figure shows the Physical layer level frame structure of IEEE802.15.4. The above figure does not show the PSDU (MPDU) in the physical layer – it will be shown in a separate figure included for the MAC Protocol Data Unit in a later portion of this chapter. Let us see the number of bytes for each section of the flowchart. The Physical Protocol Data Unit (PPDU) on total has 11+ (0 to 20)+ n bytes. The message for the modulation comes from the PPDU. It is the bottom most layer, and one layer above the PPDU is the Physical Layer. The Physical layer contains the SHR (Synchronization Header). SHR is generally known as receiving bits that determine if the particular message is in the IEEE802.15.4 format. SHR is classified into two types. One is Preamble and the other is SFD (Start of the Frame Delimiter). The Preamble sequence itself has 32 bits (4 bytes) and the range of the bit for preamble is 0xA7 in hexadecimal [14]. The Start of the Frame Delimiter is classified into 8 bits. The Physical layer also

includes the Physical Header (PHR). It is again classified into frame length and the reserved bit. The Frame length is the length of the Physical Service Data Unit or the MAC Protocol Data Unit and it is 7 bits and 1 reserved bit that makes a total of 1 byte. The total 11+(0 to 20) + n bytes have been subdivided and 6 bytes goes to PHY Layer. One Layer above this Physical layer is the MAC Layer and the MAC Layer has similar section where the number of frames is subdivided in each section. It will be explained in the detail below.

8.5. Work of MAC Layer in IEEE802.15.4 Protocol

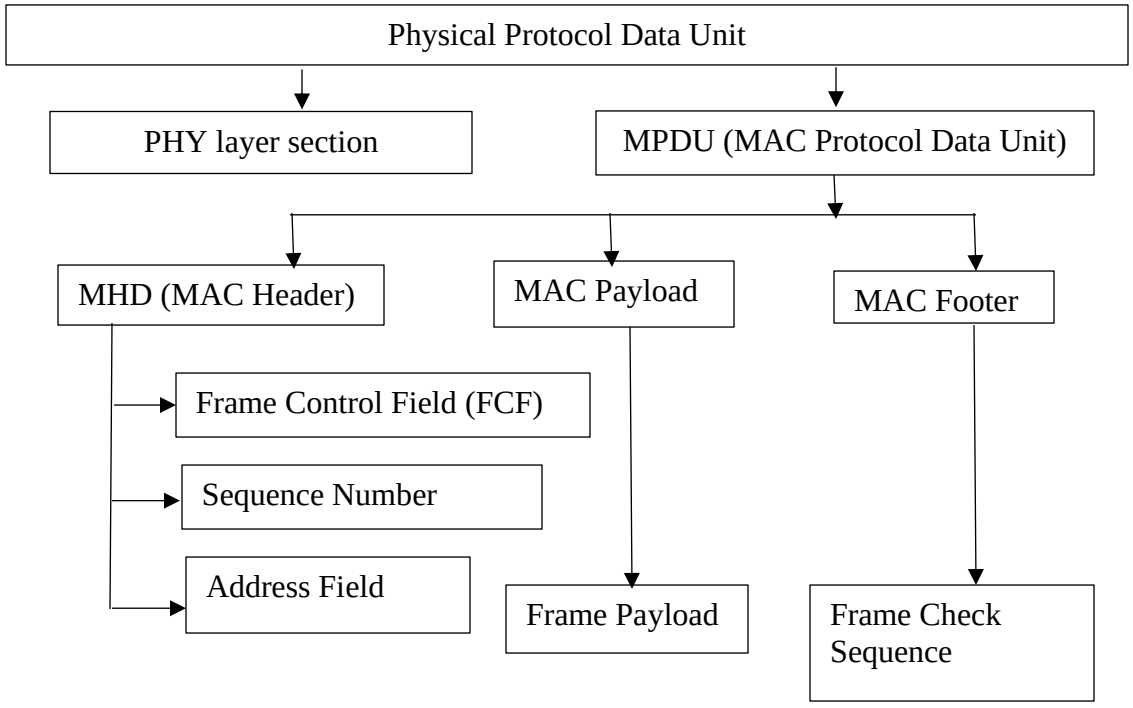


Figure 25. MAC Layer Frame Structure of IEEE802.15.4

The above figure shows the basics of MAC layer layout for IEEE802.15.4 protocol. The basic first unit is the same PHY Protocol Data Unit where the incoming message comes or the unit where it has direct connection with the hardware. Apart from the 11 bytes that go to the PHY Layer including the frame length and the reserve bits, the layer above the PHY layer is the MAC Layer. The MAC layer generally adds a header to

a packet before sending its information for modulation. The MAC is generally called the PSDU (Physical Service Data Unit) or MAC Protocol Data Unit (MPDU). The maximum MPDU size for this particular protocol is 127 bytes. The MPDU is classified into 3 sections - the MAC Header, MAC Payload and MAC footer. The MAC header is again classified into FCF (Frame Control Field), Address Information, and Data field. Frame Control Field gives the details of the way the frames are structured in other nodes. It checks what type of packets arrive and how the packets are received in the MAC layer. Data Sequence specifies the sequence of the data in a frame. The address information has information about the PAN ID, Source Address, Destination Address and Source PAN. Usually the FCF is 2 Octets / 16 bits, the sequence number is 8 bits, the destination PAN can range from 0 bits to 16 bits, and the Destination Address can be either 0 bits , 16 bits, or 64 bits. Source PAN is similar to the destination PAN and it is 0 bits to 16 bits. Source Address is similar to the destination address and can be either 0 bits , 16 bits, or 64 bits. Then comes the auxiliary security Header that can be either 0 bits, 40 bits, 48 bits, 80 bits, or112 bits. The FCS that comes at the end is the Frame Check Sequence that helps in checking the integrity of the packets. It is 16 bits total, thus doing a 16-bit CRC redundancy check of every frame that passes back to the physical layer and to the end Wireshark. MAC Payload has the number of bits that we send as data. It is also known as Frame Payload.

Frame payload comes under MAC Payload and the Frame Check Sequence is under the MAC Footer because it does the redundancy check at the end before making the flow of packets outside. The MAC layer, as soon as it receives a message, sends it out immediately but does not support any back-off time process since it does not include any

Carrier Sensing in Carrier Sense Multiple Access / Channel Allocation. (CSMA/CA) [14]. In this project, the most important block is the PHY layer block which does almost all of the modulation and demodulation process whereas the MAC shows here its limited work function. Usually the SDRs directly communicate with the Physical Layer instead of the MAC Layer. But the Wireshark connection is from both the Physical and MAC layer to ensure the proper functionality and to have a proper Frame Check Sequence. The work of the MAC layer before and after the PHY layer will be seen in detail in the next sections of this chapter.

8.6. Work of The Rime Network Stack in IEEE802.15.4 Protocol

One Layer above the MAC Layer is the network stack. It is usually specified by the “Zigbee Alliance” although it does not come under the IEEE802.15.4 category. The network stack and the MAC Layer ensures the redundancy of the packet. The actual message from the message strobe enters into this broadcast channel and then goes to the MAC Layer from the Rime Stack via “to MAC” pin and comes from the MAC to the Rime Stack, via “ From MAC” pin so that network and MAC form a closed loop with each other. The network stack also helps in broadcasting the asynchronous message to the MAC. It includes the “bcin” pin that helps in getting the incoming messages. The Network stack was implemented for Contiki by Dunkels et al. It is used as the Rime Stack in the Wime Project [14]. The Rime stack is the network communication stack designed specifically for the usage of Wireless Sensor Networks. Contiki is the OS for Wireless Sensor Networks that gets embedded with different Zigbee Hardware to sniff the packets and show its output in the senor format. The Rime network stack has a modular layered structure with more complex primitives that extend the underlying

simpler ones to offer advanced features [14]. Certain implementation has been included in this network stack as connection pins that can be seen at our GNU Radio platform. The further work concerning the Rime Stack connection on our GNU will be implemented in the operation cycle section of this chapter. The figure below illustrates the Configuration Interface with port number in the Rime Stack. It represents the normal flowchart of the Rime Stack Layer where we can see that the input message we send first enters the Rime Stack Layer for sending it to the MAC Layer, where the MAC Layer performs its duty as described before, and then the message goes to the Physical layer get modulated and demodulated and returned back as packets. These packets can later be used for external further verification purposes using Socket PDU if needed.

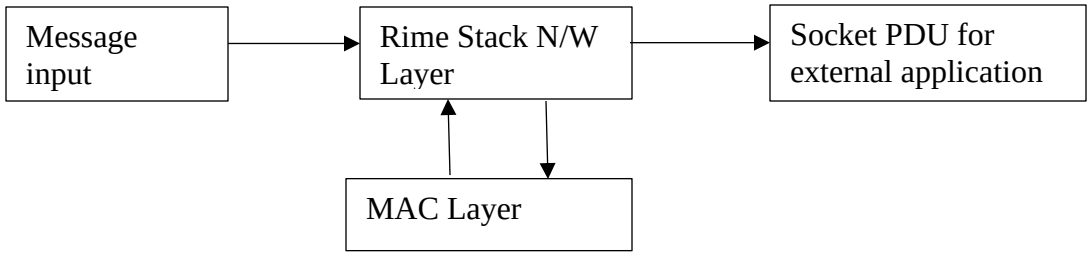


Figure 26. Rime Stack Network Level Layer of IEEE802.15.4

8.7. Architecture of IEEE802.15.4 with Lime SDR and GNU Radio

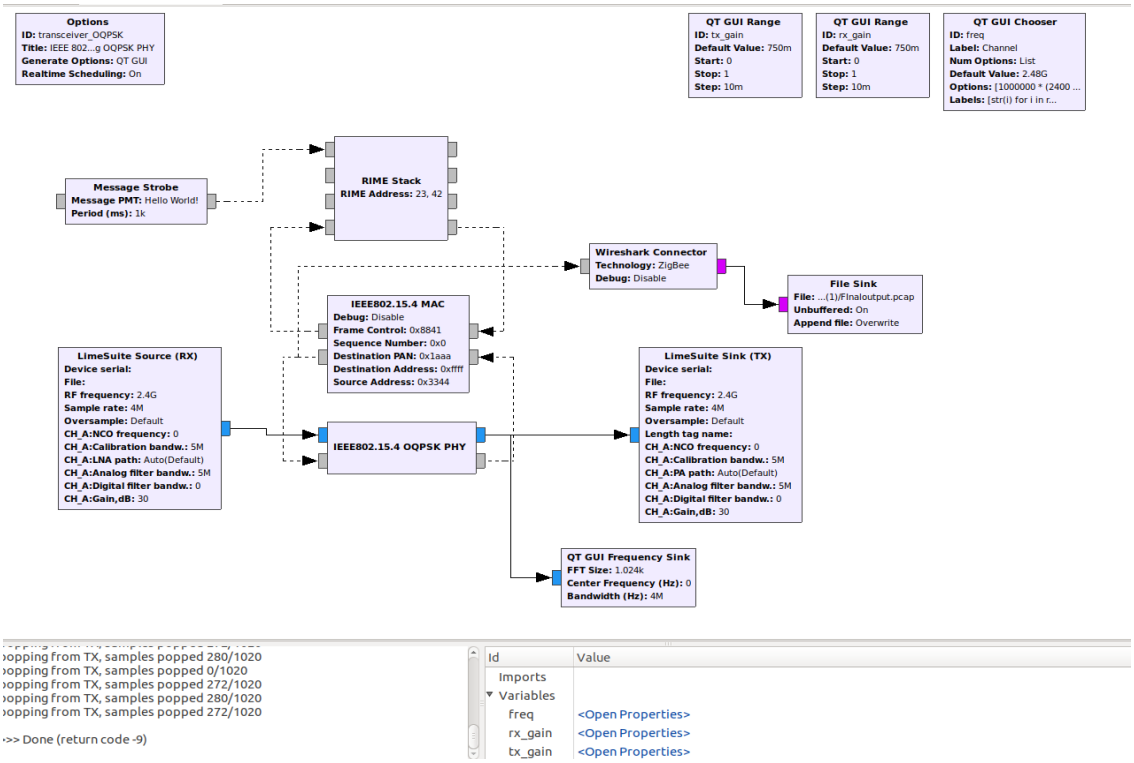


Figure 27. Architecture of IEEE802.15.4 Protocol with Lime SDR

The above figure shows the architecture of IEEE802.15.4 using Lime SDR. In our thesis, we develop a more advanced SDR called, " Lime SDR" that helps in transceiving the IEEE802.15.4 test bed using the GNU Radio. We not only replace the above SDR hardware tool with a full duplex tool, but also changed the channel frequency and the sampling rate of these blocks. The hardware source and sink we replaced with the gr-limesdr source and the gr-limesdr sink which is different from the initial projects where they used half-duplex hardware with a different sampling rate.

8.7.1. Modulation Process of IEEE802.15.4 Using OQPSK in PHY

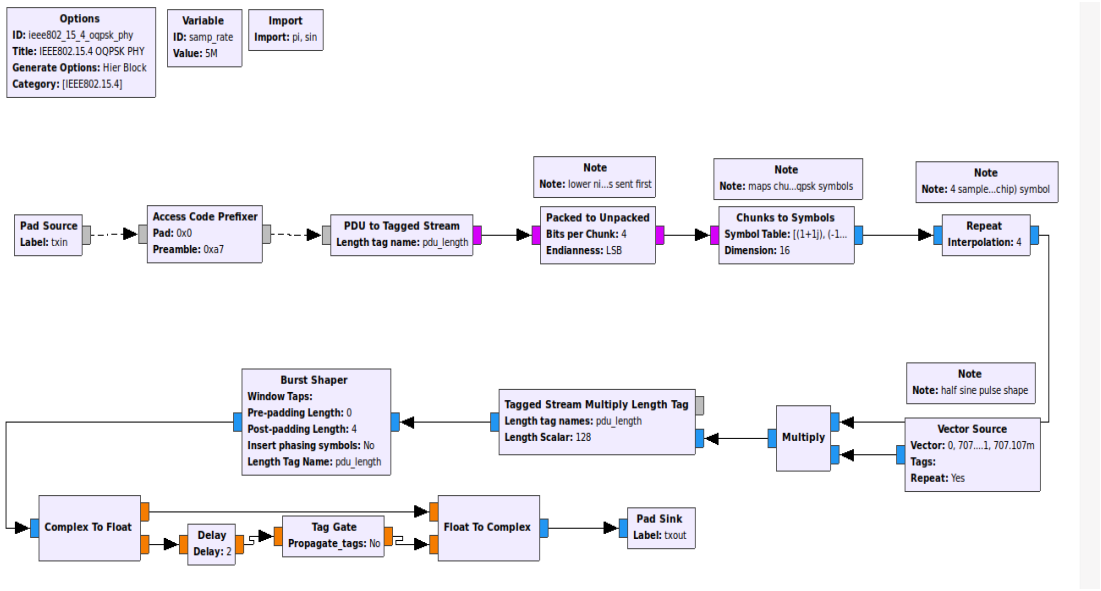


Figure 28. Modulation Block of OQPSK PHY Layer for Lime SDR

The whole modulation and demodulation process in the physical layer happen in the heir block at the GRC. Sample rate has been increased to 5 MHz according to Lime SDR. What is an hier block? An hier block or hierarchical block is the flowgraph that has been created in GNU Radio inside a single block. In our thesis, we use the Physical Layer as an heir block and compress the whole modulation and demodulation block flow graph inside a single flow graph. There are 4 important blocks that helps in the creation of this particular heir block. They are the Option Block, Pad Source, Pad Sink, and Parameter Block. The Option block must be a unique block where the name should be different from all other blocks [12]. The title parameter sets the display of the blocks. The generate option in the Option block must be set to the “Hier block”. The pad sources are the input and output of the hier block. The flow graph should have at least 1 pad source and 1 pad sink. Once we create everything along with the flow graph which is inside the pad source and pad sink, we have to click the generate option. Once we create the generate option,

the .py file for the particular hier block gets created. Then we can double check by clicking the refresh button if it is appearing on the block lists [12]. The PHY Layer modulation above follows the OQPSK modulation process as described earlier. In this modulation type, the incoming messages are converted to a stream using Physical Data Unit to Tagged Stream Block. The Physical stream of packets is at 250kbits/sec. In the next block, they are unpacked and converted into symbols. At this time, it is transmitted at the speed of 62.5 KSymbols/sec. Before modulating they need to be converted into the correct spreading sequence. The 4 Least Significant bits are converted into 1 symbol. One symbol represents 4 data streams of bits. Each 4 bit is then converted into a 16-bit symbol table using a symbol table sequence. Then these 16 bits are repeated using the repeat block for 4 times and converted into chips that has both In-Phase and Quadrature Phase component. The chips are transmitted at 2000Kchips/sec. The even bits go as the In-Phase component and the odd bits goes as the Quadrature phase component. The separated chips are then sent to the vector source with half sine wave pulse. The Burst Shapper block helps in adding padding that helps to add zeros when there is a gap at the end during the stream of blocks. We use a delay block here since we use OQPSK, instead of QPSK. The reason to give the Offset is to avoid more phase shifting and limit the phase shift to 90 degree, as explained in previous sections. The delay by 2 represents the delay by the offset of one of the components. Then the modulated output is sent to the Lime Suite Sink (TX) where the communication on air starts. The above explanation is the general modulation of OQPSK.. We can see the change in total architecture explanation. The general block for the above explanation is given as follows:

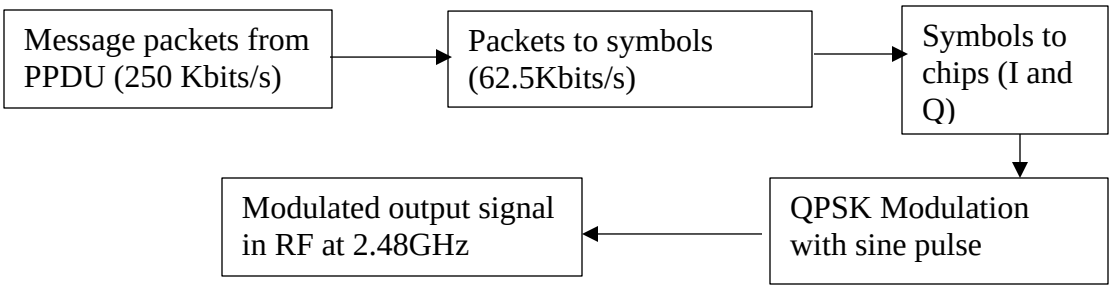


Figure 29. OQPSK Modulation Process in 2.4GHz in PHY Layer Level

8.7.2. Demodulation Process of IEEE802.15.4

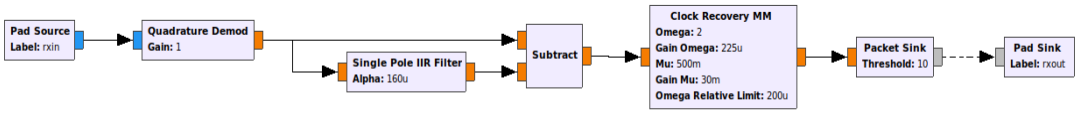


Figure 30. Demodulation Block of PHY Layer for Lime SDR

The modulated signals that is at the gr-limesdr sink (Lime Suite Sink TX) now starts its transmission over the air at 2.4GHz with channel frequency at 5MHz to communicate to the other gr-limesdr source (Lime Suite RX) block. Figure 30 shows the process of demodulation of the IEEE802.15.4 protocol in the PHY Layer level. This happens as soon as the Lime Suite Source (RX) starts its work to send the modulated bits to the PHY Layer. Note that as soon as the Lime Suite RX starts its work, the transmission ends its work until the source (RX) dumps all the modulated message to the PHY Layer. The Data packet that enters into the Physical layer goes to the demodulation block which is Rx Input. The modulated packets directly enter into the Quadrature Demodulation which performs the FM demodulation process for the incoming OQPSK modulated signal. Then it goes to the filter block where it drops out-of-band noise that enters from the demodulated packet before sending the signal to the clock recovery block. The Quadrature Demodulated signal with good dB strength directly goes to the clock

recovery whereas the one that has been mixed with the noise goes to the Single pole IIR Filter and then enters the clock recovery block. Clock Recovery MM block is the block in GNU Radio that helps in synchronizing the signal's frequency and phase so that the symbol that has been converted initially will be recovered back in the next sink level block [11]. The reason behind using the filter after the demodulator is that sometimes the bytes are so weak when the filter filters out the noise, there are chances that output tends to 0. So, if we use the demodulator first it demodulates the incoming modulated packets and the demodulated signal is filtered out depending upon its strength in dB.

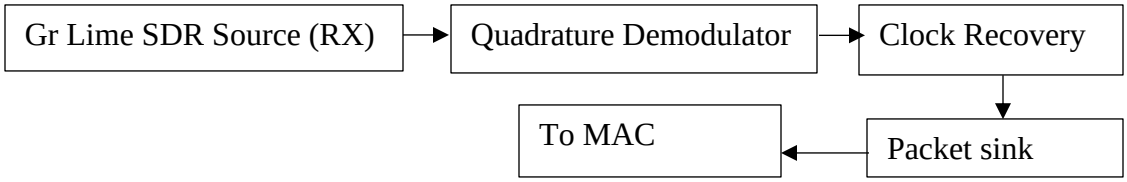


Figure 31. Demodulation Block in 2.47 GHz in PHY Layer Level

As soon as the demodulated signal comes to the clock recovery MM block as a stream, the clock recovery block performs its operation by decimating the signals and also the decimated signal is then followed by Muller and Muller Discrete time error tracking synchronizer. The synchronizer use is mainly to extract the chip sequence from the frequency and from the phase of the signal stream. There are a total of 5 sections in the clock recovery MM where the first 2 are the gain and gain of the frequency, the next two are the gain in mu and the gain of the phase, and the final section is the total gain limit. The first 2 sections synchronize the frequency of the signal into the chips whereas the next 2 synchronize the phase of the signal into the chip sequence. Now the recovered chip sequences are converted into the initial symbols using the packet sink block. The symbol output will be sent to the MAC where the second performance of MAC Layer

initiates.

8.7.3. Overall Working of the Protocol Cycle

In our thesis we start to send a sequence of message called, “Hello World” at a certain time interval from the message strobe. This message strobe sends this message continuously at a given time interval to the “ Rime Stack network layer”. Rime Stack gets the input through the broadcast input which is the “bin” pin in the Rime Stack. Then from the “bout” pin it goes to the MAC, - the pin on the MAC is the “to MAC” pin in the block. Let us name the Channel incoming and outgoing broadcasting packets at Rime Stack as [129,130] so that 2 pins, one for input and the another one for output, get created. Similar to the broadcast connection, we can also create a unicast or multiple unicast communication and connect it into the different socket PDU. This helps after the packets get processed in PHY and MAC Layer and arrive at the Rime to communicate with other hardware. In this way we can configure the Rime Stack on the transceiver. Although it helps to send the output packets to different hardware, in this thesis we use it to have a connection to MAC Layer by assigning it a broadcast channel number. We also assign a Rime Address randomly for the messages. The Network Layer for us is just to transmit the message to the MAC by adding an address to it here. The MAC Layer is implemented on top of the Physical layer. The MAC layer does really the basic work in this transceiver. The function of the MAC Layer is limited yet it helps in giving the essential features of the packets from the network layer with an IEEE802.15.4 header and helps in working the FCS (Frame Check Sequence of the packets). That is, it calculates the redundancy check of the incoming packets before sending them to the stream based physical layer. In Receiving, it has another operation which we will see

shortly. As soon as it adds an IEEE802.15.4 header for the broadcast message it sends the asynchronous message to the PHY Layer.

The PHY Layer receives the asynchronous message at the “Txin” pin from the “pdu out” pin of the MAC. Now the main layer Physical layer works on the modulation part by converting the message from message based to stream based and changes that to symbols and from symbols to chips using the chip sequence and modulates it further using OQPSK modulation using sine pulse with a delay of 0.5 in the Offset. The modulated output which is the form of packets is sensed by our Lime Suite TX that is our gr-limesdr sink. Now the output from the gr-limesdr sink is transmitted to the gr-limesdr source which is Lime Suite RX on the other side. The whole performance can be done using the Lime SDR hardware that is connected to the Laptop or PC through USB3.0. Since Lime SDR has the capacity to be full duplex once the gr-limesdr sink sends the message and stops its operation, immediately the hardware can shift the message to the gr-limesdr source RX without any delay. Unlike USRP, we don’t need channel A and B, or we don’t need two hardware or daughterboards to work to make the operation reliable. Note that the packet size that we get should match the IEEE802.15.4 standard, and in that case Lime SDR hardware packet size also gets matched. If it does not get matched the radio becomes silent. The RX side still waits for a packet without realizing the fact that the radio has become idle. Since our message satisfies the Lime SDR USB packet size, GNU radio packet size and the protocol PPDU byte size, it should work properly with less interference. The less interference maybe due to the fact the transmission to reception takes place at 2.4GHz. Also, the fact that we have selected channel 26 makes the interference less, so even if there is a drop in radio signal the packet still gets picked up

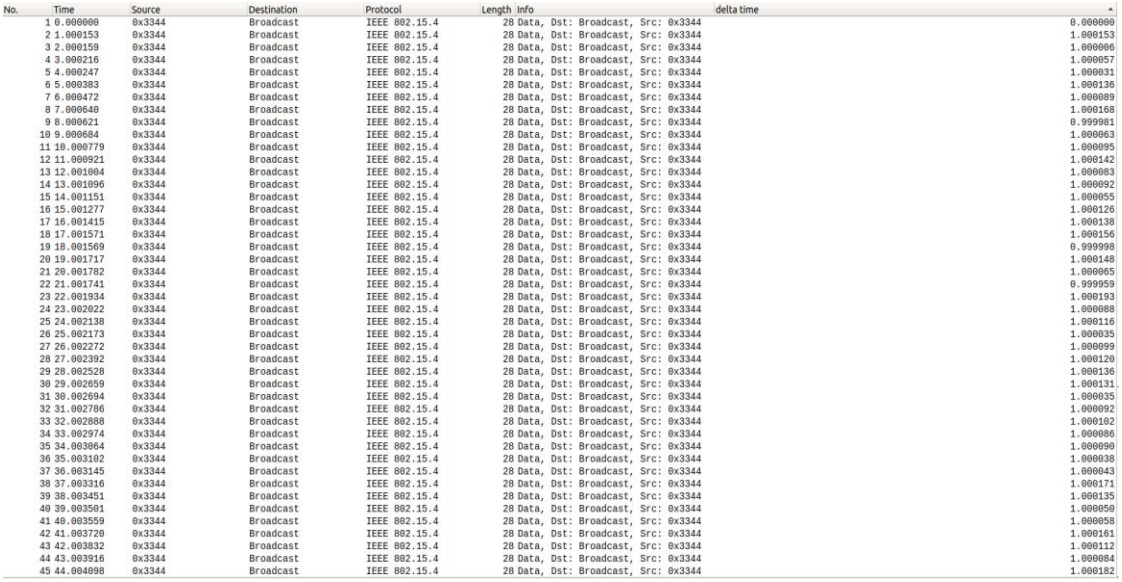
and arrives at the reception side perfectly without any packet loss. Thus, there are chances that if we use Lime SDR , we can get 100% packet Transceiving done without any loss. But other SDRs like USRP have issues like daughterboard issue, FPGA issue, dual Channel issue/dual hardware that may lead to packet loss.

Once it arrives on the other side of the gr-limesdr source (RX side), the modulated packets are sent into the PHY Layer again at the RX input side where it process the Radio Frequency demodulation using the Quadrature Demodulation as explained above in the demodulation process. The demodulated packets are then sent into the filters where the signal with the noise is filtered and only the signal that has good dB is passed into the clock recovery MM where the signals are changed into the chip sequence and then further passed to the packet sink where they are decoded and the chips are joined back to the symbols. The symbols that are framed with the header in the decoded packets are sent back to the MAC layer using “Rxout” to the “pdu in”. Now the MAC Layer does its work in the reverse way of the transmitting MAC. It first removes the header that it earlier added to the asynchronous message and checks its redundancy using the CRC Checksum to determine if it is from the correct path or not. If the MAC Layer does not find it right it drops the entire packet by flagging it as a “malfunctioned packet” and starts to detect the next packet. But the ACK Acknowledgement or back off time is still not supported yet because there is no Carrier Sense Multiple Access / Channel Allocation available in this particular MAC Layer. After removing the header and once it passes the CRC, the output gets directly connected to the Wireshark through a “Wireshark connector block”.

9. RESULT AND ANALYSIS

The implementation was designed on a Dell laptop with Linux Ubuntu OS running GNU Radio and Lime SDR at 2.4 GHz frequency with the strong work of Physical layer of the OSI Model at IEEE802.15.4 level and the Checksum work of MAC Layer at the IEEE802.15.4 level. To obtain the result for this transceiver test tool, a wireshark has been added to the flowgraph along with the file sink. The installation procedure of Wireshark in the Ubuntu is explained in chapter 4. The Wireshark can either be in the form of append or overwrite in its block. Fixing it as append or overwrite is according to our convenience. Append makes the .pcap file dump itself to the wireshark in the upcoming execution whereas overwrite helps in overwriting the previous .pcap file into the new .pcap file. The Wireshark output is then connected to the file sink. The file sink helps us in allotting the destination in our PC for the .pcap file. We are producing a connection to wireshark from the MAC Layer after it removes the header, that is, from the pin “app out” to the input in the Wireshark. The file sink has the destination file with the “hello world” packet that we transmitted into the Lime Suite Sink and Lime Suite Source are at the end as a .pcap file. If we go to the destination and double click on the .pcap file, the wireshark automatically opens and shows the output that has been obtained. In this thesis to see the frequency graph along with the result, the frequency sink block has been added that shows the flow of packets from gr-limesdr sink to gr-limesdr source at 2.478 GHz. The frequency block is also known as QT GUI Frequency block. The frequency sink block attached at the end of the “txout” pin in the Physical layer shows the modulated signal at the frequency sink that travels to the gr-limesdr sink (TX). Channel number at 26 can be seen at the bottom of the frequency sink block. The

thesis gives the flexibility to change the channel number from 11 to 26 according to the Zigbee protocol standard level using the QT GUI Chooser block in the GNU Radio by giving the formula in the label ,"[str(i) for i in range(11,27)]". By fixing the QT GUI range in the GNU Radio block we have the flexibility to change the transmitter and receiver gain as shown in Figure 22. Wireshark will be automatically opened by double clicking the .pcap file that we obtained at the destination folder in the file sink. The obtained Wireshark will be as follows:



No.	Time	Source	Destination	Protocol	Length	Info	delta time
1	0.000000	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	0.000000
2	1.000153	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000153
3	2.000159	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000006
4	3.000216	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000057
5	4.000247	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000031
6	5.000383	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000136
7	6.000472	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000089
8	7.000640	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000168
9	8.000621	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	0.999981
10	9.000684	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000063
11	10.000779	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000095
12	11.000921	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000142
13	12.001004	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000083
14	13.001096	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000092
15	14.001151	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000055
16	15.001277	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000126
17	16.001415	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000138
18	17.001571	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000156
19	18.001569	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	0.999998
20	19.001717	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000148
21	20.001782	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000065
22	21.001741	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	0.999959
23	22.001934	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000193
24	23.002022	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000088
25	24.002138	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000116
26	25.002173	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000035
27	26.002272	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000099
28	27.002392	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000120
29	28.002528	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000136
30	29.002659	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000131
31	30.002694	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000035
32	31.002786	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000092
33	32.002888	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000102
34	33.002974	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000086
35	34.003064	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000090
36	35.003102	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000038
37	36.003145	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000043
38	37.003316	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000171
39	38.003451	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000135
40	39.003501	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000050
41	40.003559	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000058
42	41.003720	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000161
43	42.003832	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000112
44	43.003916	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000084
45	44.004098	0x3344	Broadcast	IEEE 802.15.4	28	Data, Dst: Broadcast, Src: 0x3344	1.000182

Figure 32: Output of .pcap File in Wireshark Protocol Analyzer

The result has different columns that show each packet's arrival time and the destination address along with the source address. It is also showing the protocol type of our packet, that is IEEE802.15.4. In edit -→Preference-→ Columns sections, we have added the column that shows the amount of time between the packets as the delta time. The further encoded and decoded output is shown in the next section of the wireshark figure 32.

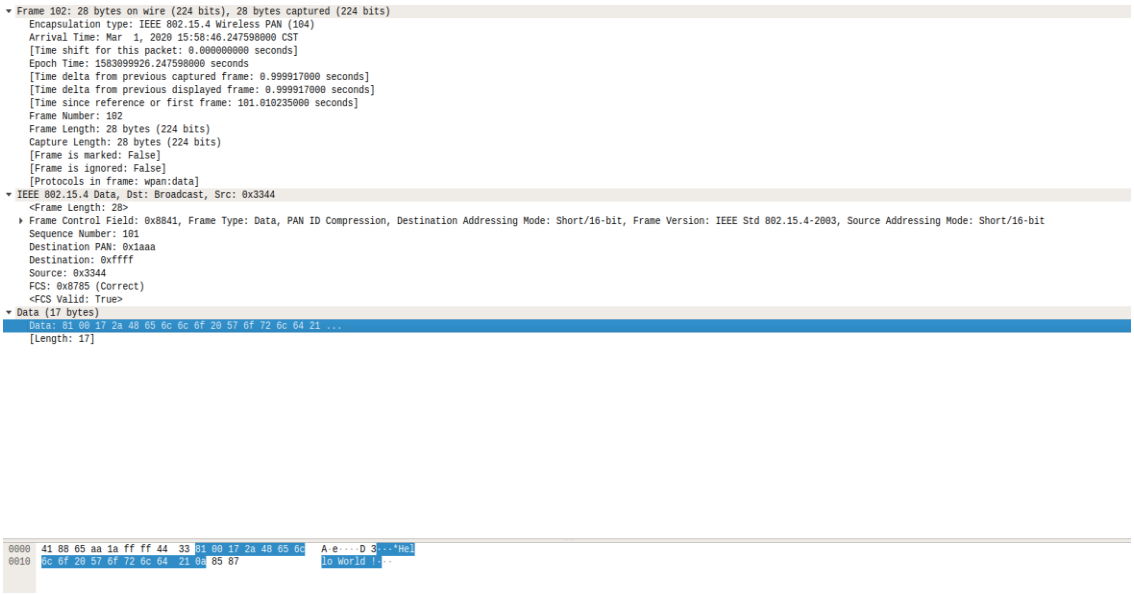


Figure 33. Details of the Frame in Wireshark Protocol Analyzer

The thesis shows the above testbed is successful by showing its encoded packets and decoded messages at the bottom of the Wireshark. MAC address in the thesis along with Hexadecimal address for the above packet can be seen at the Wireshark. We also received the decoded “Hello World” packet back at the Wireshark.

Each packet can be selected so that wireshark displays its information. Say for example, in the above list of packets we selected one, and at the bottom page of the wireshark we can see the frame number . “Number of bytes” on the wire that is being processed is shown as 28 bytes (224 bits) and also the number of bytes that we obtained is also 28 bytes (224 bits). This directly shows that our testbed is successful and the Lime SDR can control the protocol level packets. Unlike other SDRs we didn’t face any packet loss. We also obtained other information like the encapsulation type as IEEE802.15.4 Wireless PAN. Each packet shows its arrival time. Initial packet arrival time is 0.0000 seconds. We can also obtain the amount of time between the packets as delta time separately in the column section. Protocol in the frame is wpan:data which is true and is

successful. The MAC address is represented in the MAC as source address, destination address along with the destination PAN has been addressed in the Wireshark analyzer. The FCS section shows if the packet is valid and if it is arriving from the exact source to the destination. If there is any interference in the packet checksum, it will represent as Bad FCS. But immediately, the gr-limesdr source and sink have the capacity to send the packet without any loss. The 100% packet arrival is achieved. We can also get other information in the statistics section like I/O Graph, Protocol Hierarchy statistics, Packet length, End points etc., The I/O graph is given below:

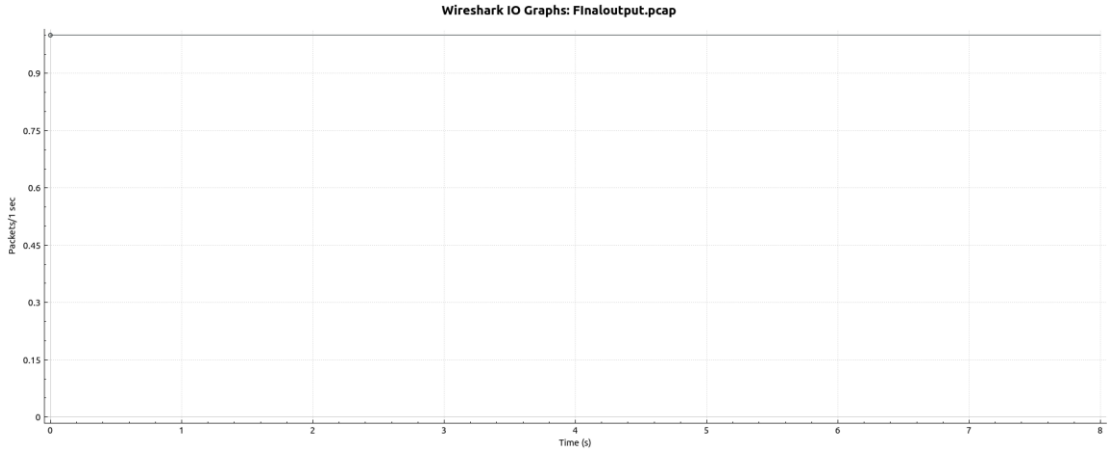


Figure 34. Wireshark I/O Graph

The above graph from the Wireshark represents the input/output graph obtained from the IEEE802.15.4 packets versus time. The straight line in the input/output graph shows that FCS is true, and the packets are received perfectly with respect to time. When there is a sudden drop in the graph and then an increase, it means that there is a temporary signal drop, but that Lime SDR recovered immediately and there will be no packet loss. The recovery time for Lime SDR if there is a small network or environmental issue is just 0.0001s. The packet length is 28 bytes . The wire byte (for TX) and the received bytes are equal which means there is no packet loss. Thus, we can say that the

Tx loss is 0% and packet loss is also 0%. But sometimes there may be a small network or environmental issue when generating the packets. But Lime SDR can recover immediately if there is a clearance of the issue. This may create a signal loss at wireshark I/O Graph. Although the signal may be temporarily dropped, the data won't be lost. Frame Check Sequence can give the LQI (Link Quality Index) correlation value. The Average LQI Correlation value for the ten packets in the above protocol is achieved to be 60.33 that falls under the correlation value of LQI range according to the IEEE802.15.4 standard. 21 to 30 is the channel where all the bytes can get transferred. We transferred at 2.4GHz in the 26^{th} channel. Each value changes according to the channel use and the interference at that point of time. This result leads us to many conclusions and suggestions for future work about this particular protocol since they can give 0% packet loss.

10. CONCLUSION AND FUTURE WORK

10.1. Conclusion

The GNU Radio Transceiver of IEEE802.15.4 protocol testbed with the help of Lime SDR has been successfully implemented. The testbed shows that the whole transmission and reception can be performed in Lime on single channel A in both Lime Suite source and Lime Suite sink without any packet loss.

Previous research has involved using half duplex radios with GNU Radio to control different protocols. New contributions by this thesis include the integration of the Lime SDR full duplex hardware with the open source GNU Radio software in the form of source and sink, that is transmitter and the receiver, to evaluate and control the IEEE802.15.4 protocol. To prove the work further we added a Wireshark custom block in the flowgraph where all the .pcap packets from the file sink are obtained and stored. We can successfully see in Wireshark how the Lime Suite source and sink has performed its operation through wireless transmission at 2.47GHz. We've shown how Lime SDR can help us create a low-cost testbed to develop full IoT systems in the future. We've also used GNU Radio to develop a new flowgraph and modify an existing flowgraph according to our hardware standard and produce performance graphs from the GRC tool. Thus the above test design, once it is uploaded in the online git hub site, will be available as an open source software and will help future researchers and developers to take the code and adjust the flowgraph according to their hardware to control the same protocol in a better way or to control different protocols like IEEE802.11 using the same Lime SDR hardware. As suggested below, the flowgraphs can be used to control the protocol in application level to different hardware.

10.2. Future Work and Suggestions

As the popularity of Internet of Things increases, we are automatically in need of radios that control the wireless protocols. The flowgraphs used and developed in this thesis will be an output of open source GNU Software stack that will help future researchers to develop or modify their research according to their wireless protocol standard. Proving that the Lime can control the IEEE802.15.4 protocol enables future researchers to consider working with Lime to control other protocols like Bluetooth and Wi-Fi.

This current test tool can be used by future researchers to control the Zigbee applications. Currently we designed this test bed only for PHY and MAC Layer . We can attach the Zigbee alliance that is the application level layer in the GNU Radio software to make the particular message communicate with other Zigbee hardware. To initiate this idea of communication and to prove this in application level, we can try to use the Texas Instrument hardware CC2531 that sniffs the IEEE802.15.4 protocol packets / Zigbee packets using Packet sniffer Firmware. This test lets us make the communication possible from the GNU Radio to the CC2531 Zigbee sniffer.

Another mode of communication to the application level layer can be performed by making this flowgraph communicate to the XBee hardware using XCTU console. The XBee in XCTU can act as either coordinator or router, that helps in either sending or receiving the message. After initiating the message in XBee, we can check at the Wireshark Layer level to determine if we are receiving the XBee messages through GNU Radio. Trying to send a message to the XBee XCTU console, making the XCTU console act as the end point, can also be tested. If the transmission is successful the console

should show us the successful receiving of messages from GNU Radio. These messages can be broadcast through the Socket PDU block in the GNU Radio Companion. This block helps to send the message in the form of TCP, UDP broadcasting format acting as an application level. This will evaluate the transmission and reception functionality of GNU Radio Companion to the application level outside Zigbee hardware.

Although the Lime SDR has been successful in transceiving the packet that it obtained from the wire, there are some constraints that limit its capability, such as network issues and Wi-Fi interruption. Lime SDR, even though it helped us to control a single modulation technique, is not limited by that technique.

Instead of OQPSK , future researches can also consider other modulation techniques along with Lime SDR to check how the changes affect the system. This flowgraph can be modified according to a different protocol to evaluate different modulation techniques and different Medium Access Control Protocols.

For example, the development of MAC Layer with a different Frame Control Field (FCF) than used in this thesis will enable the Lime to send and receive packets in a secure way. MAC has most of the security control over the frames. Improving it will help reduce the interference.

Other techniques include adding an Acknowledgement to the MAC Layer. This can be done by changing the FCF format that we previously allotted to the MAC Layer. The address bit at the MAC can be allotted in such a way as to check its interference from other hardware nearby. The current research also limits the number of bytes that we can send for modulation and demodulation processes. Future researchers may work on increasing the message size that we send for modulation and demodulation processes by

increasing the capability of SDR. The possibility of using the Lime SDR and GNU Radio for controlling multiple IoT protocols at a higher level is an additional subject for research in the near future.

APPENDIX SECTION

APPENDIX A

A.1. Drivers for lime suite at Ubuntu Terminal

sudo add-apt-repository -y ppa:myriadrf/drivers
sudo apt-get update
sudo apt-get install limesuite liblimesuite-dev liblimesuite-udev limesuite-images
sudo apt-get install soapysdr-tools soapysdr-module-lms7

A.2. Lime Suite Dependencies at Ubuntu Terminal

#soapysdr tools used to be called just soapy sdr on older packages and can be installed with:

sudo apt-get install soapysdr soapysdr-module-lms7

A.3. Lime Suite Dependencies at Ubuntu Terminal

#packages for soapysdr available at myriadrf PPA

sudo-add-apt-repository -y ppa:myriadrf/drivers
sudo apt-get update

#install core libraries and build dependencies

sudo apt-get install git g++ cmake libsqlite3-dev

#install hardware support dependencies

sudo apt-get install libsoapysdr-dev libi2c-dev libusb-1.0-0-dev

#install graphics dependencies

sudo-apt get install libwxgtk3.0-dev freeglut3-dev

A.4. Building Lime Suite

git clone https://github.com/myriadrf/Limesuite.git
cd Limesuite
git checkout stable
mkdir builddir && cd builddir
cmake ../
make -j4

sudo make install
sudo ldconfig

Linux users need to install the udev rules to enable non-root users to access USB-based devices like LimeSDR:

cd Limesuite/udev-rules
sudo ./install.sh

A.5. Installing GNU Radio in Ubuntu

$apt install gnuradio

Or

$sudo apt-get update
$sudo apt install gnuradio

A.6. Installation of GNU Radio Companion

$ gnu radio - companion

Note: GRC cannot be installed without installing GNU Radio.

A.7. Installation procedure for Wireshark at Ubuntu

$sudo apt update
$sudo apt install wireshark
$sudo usermod -aG wireshark $(whoami)
$sudo reboot

To run the wireshark after installation, the terminal command should be entered as:

$wireshark or $sudo wireshark

A.8. Terminal code to plugin Gr-Limesdr for GNU Radio

Command in terminal to install Boost and SWIG

sudo apt-get install libboost-all-dev swig

Downloading gr-limesdr source

git clone https://github.com/myriadrf/gr-limesdr

Building and installing gr-limesdr from source

```
cd gr-limesdr
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

APPENDIX B

Transceiver code for IEEE802.15.4 Protocol with LimeSDR and GNU Radio

```
# GNU Radio Python Flow Graph

# Title: IEEE 802.15.4 Transceiver using OQPSK PHY
# Generated: Wed Mar 11 12:10:23 2020
##################################################

from distutils.version import StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

import os
import sys
sys.path.append(os.environ.get('GRC_HIER_PATH',
os.path.expanduser('~/.grc_gnuradio')))

from PyQt5 import Qt
from PyQt5 import Qt, QtCore
from PyQt5.QtCore import QObject, pyqtSlot
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from ieee802_15_4_oqpsk_phy import ieee802_15_4_oqpsk_phy  # grc-generated
hier_block
from optparse import OptionParser
import foo
import ieee802154
import ieee802_15_4
import limesdr
import pmt
import sip
```

```
from gnuradio import qtgui

class transceiver_OQPSK(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "IEEE 802.15.4 Transceiver using OQPSK PHY")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("IEEE 802.15.4 Transceiver using OQPSK PHY")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "transceiver_OQPSK")

        if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
            self.restoreGeometry(self.settings.value("geometry").toByteArray())
        else:
            self.restoreGeometry(self.settings.value("geometry", type=QtCore.QByteArray))

        ##################################################
        # Variables
        ##################################################
        self.tx_gain = tx_gain = 0.75
        self.rx_gain = rx_gain = 0.75
        self.freq = freq = 2480000000

        ##################################################
        # Blocks
        ##################################################
        self._tx_gain_range = Range(0, 1, 0.01, 0.75, 200)
        self._tx_gain_win = RangeWidget(self._tx_gain_range, self.set_tx_gain, "tx_gain",
"counter_slider", float)
        self.top_layout.addWidget(self._tx_gain_win)
```

```
        self._rx_gain_range = Range(0, 1, 0.01, 0.75, 200)
        self._rx_gain_win = RangeWidget(self._rx_gain_range, self.set_rx_gain, "rx_gain",
"counter_slider", float)
        self.top_layout.addWidget(self._rx_gain_win)
        self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
        1024, #size
        firdes.WIN_BLACKMAN_hARRIS, #wintype
        0, #fc
        4e6, #bw
        "", #name
        1 #number of inputs
        )
        self.qtgui_freq_sink_x_0.set_update_time(0.10)
        self.qtgui_freq_sink_x_0.set_y_axis(-140, 10)
        self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB')
        self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE, 0.0, 0, "")
        self.qtgui_freq_sink_x_0.enable_autoscale(False)
        self.qtgui_freq_sink_x_0.enable_grid(False)
        self.qtgui_freq_sink_x_0.set_fft_average(1.0)
        self.qtgui_freq_sink_x_0.enable_axis_labels(True)
        self.qtgui_freq_sink_x_0.enable_control_panel(False)

        if not True:
          self.qtgui_freq_sink_x_0.disable_legend()

        if "complex" == "float" or "complex" == "msg_float":
          self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

        labels = ['', '', '', '', '',
                '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                "magenta", "yellow", "dark red", "dark green", "dark blue"]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                1.0, 1.0, 1.0, 1.0, 1.0]
        for i in xrange(1):
           if len(labels[i]) == 0:
              self.qtgui_freq_sink_x_0.set_line_label(i, "Data {0}".format(i))
           else:
           self.qtgui_freq_sink_x_0.set_line_label(i, labels[i])
           self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
           self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
           self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])
```

```
        self._qtgui_freq_sink_x_0_win =
sip.wrapinstance(self.qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_freq_sink_x_0_win)
        self.limesdr_source_0 = limesdr.source('', 0, '')
        self.limesdr_source_0.set_sample_rate(5e6)
        self.limesdr_source_0.set_center_freq(2.4e9, 0)
        self.limesdr_source_0.set_bandwidth(5e6,0)
        self.limesdr_source_0.set_gain(30,0)
        self.limesdr_source_0.set_antenna(255,0)
        self.limesdr_source_0.calibrate(5e6, 0)

        self.limesdr_sink_0 = limesdr.sink('', 0, '', '')
        self.limesdr_sink_0.set_sample_rate(5e6)
        self.limesdr_sink_0.set_center_freq(2.4e9, 0)
        self.limesdr_sink_0.set_bandwidth(5e6,0)
        self.limesdr_sink_0.set_gain(30,0)
        self.limesdr_sink_0.set_antenna(255,0)
        self.limesdr_sink_0.calibrate(5e6, 0)

        self.ieee802_15_4_oqpsk_phy_0 = ieee802_15_4_oqpsk_phy()
        self.ieee802_15_4_mac_0 =
ieee802_15_4.mac(False,0x8841,0,0x1aaa,0xffff,0x3344)
        self.ieee802154_rime_stack_0 = ieee802154.rime_stack(([129]), ([130]), ([131]),
([23,42]))
        self._freq_options = [1000000 * (2400 + 5 * (i - 10)) for i in range(11, 27)]
        self._freq_labels = [str(i) for i in range(11,27)]
        self._freq_tool_bar = Qt.QToolBar(self)
        self._freq_tool_bar.addWidget(Qt.QLabel('Channel'+": "))
        self._freq_combo_box = Qt.QComboBox()
        self._freq_tool_bar.addWidget(self._freq_combo_box)
        for label in self._freq_labels: self._freq_combo_box.addItem(label)
        self._freq_callback = lambda i:
Qt.QMetaObject.invokeMethod(self._freq_combo_box, "setCurrentIndex",
Qt.Q_ARG("int", self._freq_options.index(i)))
        self._freq_callback(self.freq)
        self._freq_combo_box.currentIndexChanged.connect(
        lambda i: self.set_freq(self._freq_options[i]))
        self.top_layout.addWidget(self._freq_tool_bar)
        self.foo_wireshark_connector_0 = foo.wireshark_connector(195, False)
        self.blocks_message_strobe_0 = blocks.message_strobe(pmt.intern("Hello
World!\n"), 1000)
        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1,
'/home/Sam/Desktop/Thesis final/transceiver_OQPSK.py', False)
        self.blocks_file_sink_0.set_unbuffered(True)

        ##################################################
```

```
        # Connections
        ##################################################
        self.msg_connect((self.blocks_message_strobe_0, 'strobe'),
(self.ieee802154_rime_stack_0, 'bcin'))
        self.msg_connect((self.ieee802154_rime_stack_0, 'toMAC'),
(self.ieee802_15_4_mac_0, 'app in'))
        self.msg_connect((self.ieee802_15_4_mac_0, 'pdu out'),
(self.foo_wireshark_connector_0, 'in'))
        self.msg_connect((self.ieee802_15_4_mac_0, 'app out'),
(self.ieee802154_rime_stack_0, 'fromMAC'))
        self.msg_connect((self.ieee802_15_4_mac_0, 'pdu out'),
(self.ieee802_15_4_oqpsk_phy_0, 'txin'))
        self.msg_connect((self.ieee802_15_4_oqpsk_phy_0, 'rxout'),
(self.ieee802_15_4_mac_0, 'pdu in'))
        self.connect((self.foo_wireshark_connector_0, 0), (self.blocks_file_sink_0, 0))
        self.connect((self.ieee802_15_4_oqpsk_phy_0, 0), (self.limesdr_sink_0, 0))
        self.connect((self.ieee802_15_4_oqpsk_phy_0, 0), (self.qtgui_freq_sink_x_0, 0))
        self.connect((self.limesdr_source_0, 0), (self.ieee802_15_4_oqpsk_phy_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "transceiver_OQPSK")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_tx_gain(self):
        return self.tx_gain

    def set_tx_gain(self, tx_gain):
        self.tx_gain = tx_gain

    def get_rx_gain(self):
        return self.rx_gain

    def set_rx_gain(self, rx_gain):
        self.rx_gain = rx_gain

    def get_freq(self):
        return self.freq

    def set_freq(self, freq):
        self.freq = freq
        self._freq_callback(self.freq)


def main(top_block_cls=transceiver_OQPSK, options=None):
    if gr.enable_realtime_scheduling() != gr.RT_OK:
```

```
        print "Error: failed to enable real-time scheduling."

    if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.aboutToQuit.connect(quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

APPENDIX C

Hardware, Software tools :

- LimeSDR – the main hardware component of the thesis
- GNU Radio – GRC
- Lime Suite Application
- Wireshark Protocol Analyzer
- File Types Used: .txt, .grc, .py, .png, .jpeg/jpg, .pcap
- IEEE802.15.4 Protocol
- XBee Hardware
- CC2531 TI USB

Computer Specification

- Dell Laptop – Intel core i7 8^{th} Generation Processor
- Multi-functional USB Type C
- Linux Ubuntu OS

REFERENCES

[1] Wikipedia , "Radio Frequency," Wikipedia, 2013 - 2019. [Online]. Available: https://en.wikipedia.org/wiki/Radio_frequency. [Accessed 12 December 2019].

[2] "Merriam-Websiter Dictionary," 2019. [Online]. Available: https://www.merriam-webster.com/dictionary/radio%20frequency. [Accessed 12 December 2019].

[3] wikipedia, "Internet of Things," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things. [Accessed 12 December 2019].

[4] Appricity Enterprise , "Appricity Enterprise IoT," Internet of Things , [Online]. Available: apptricity.com/what-is-iot/. [Accessed 12 December 2019].

[5] Wikipedia, "Software Defined radio," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_radio#/media/File:SDR_et_WF.svg. [Accessed 16 12 2019].

[6] P. J. M. S. Dusan N Grujic, "Using software defined for RF measurement," Belgrade, 2017.

[7] wiki, "Myriad RF," [Online]. Available: https://wiki.myriadrf.org/LimeSDR. [Accessed 16 December 2019].

[8] Wikipedia, "Limesuite," [Online]. Available: https://wiki.myriadrf.org/Lime_Suite. [Accessed 03 01 2020].

[9] T. Schmid, "GNU Radio 802.15.4 En- and Decoding," in *NESL*, Los Angeles, 2006.

[10] Wikipedia, "GNU Radio," GNURadio, [Online]. Available: https://wiki.gnuradio.org/index.php/Main_Page. [Accessed 05 01 2020].

[11] Wikipedia, "Guided_tutorial_documentation GNU Radio," [Online]. Available: https://wiki.gnuradio.org/index.php/Guided_Tutorial_Introduction. [Accessed 05 01 2020].

[12] Wikipedia, "Guided_Tutorial_GRC," [Online]. Available: https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC. [Accessed 01 05 2020].

[13] Wikipedia, "Wireshark," [Online]. Available: https://en.wikipedia.org/wiki/Wireshark#History. [Accessed 12 01 2020].

[14] C. L. F. D. C. S. Bastian Bloessl, "A GNU Radio-based IEEE802.15.4 Testbed," in *Computer and Communication Systems, Institute of Computer Science* , Austria.

[15] L. Choong, "Multi Channel IEEE 802.15.4 Packet Capture Using Software Defined radio," *UCLA Networked and Embedded Sensing Lab,* pp. 9-10, 2009.

[16] Wikipedia , "Zigbee," [Online]. Available: https://en.wikipedia.org/wiki/Zigbee. [Accessed 16 01 2020].

[17] Wikipedia , "Thread (Network Protocol)," [Online]. Available: https://en.wikipedia.org/wiki/Thread_(network_protocol). [Accessed 16 01 2020].

[18] R. Thandee, "IEEE802.15.4 Implementation on an Embedded Device," Virginia Polytechnic Institute and State University , Virginia , 2012.

[19] Wikipedia, "IEEE802.15.4," [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.15.4. [Accessed 19 01 2020].

[20] PACE Education, "webpage PACE networking star," [Online]. Available: http://webpage.pace.edu/ms16182p/networking/star.html. [Accessed 03 10 2020].

[21] Digital Ctizen, "Digital citizen life," [Online]. Available: https://www.digitalcitizen.life/what-is-p2p-peer-to-peer. [Accessed 10 03 2020].

[22] Shashidhar, "The Zigbee Technology," India .

[23] C. Radhakrishna, "IOT Based Network Management Poryal for Augumented Reality Applications," Texas State University, San marcos, 2018.

[24] Wikipedia , "Gr-limesdr_Plugin_for_GNURadio," [Online]. Available: https://wiki.myriadrf.org/Gr-limesdr_Plugin_for_GNURadio. [Accessed 27 01 2020].

[25] H. J.-K. C. M. C.-J. L. Feng Li Cheng, "SDR Implementation of LTE R13 NB-IoT Downlink Vector Signal Generator," in *IEEE Conference on Consumer Electronics* , 2017, 2017.

[26] C.-Z. K. M. A. V. P. Cristinel Gavrila, "Reconfigurable IoT Gateway Based on SDR Platform," in *Transilvania University of Brasov*, Romania , 2018.

[27] A. M. S. P. Manolis Sunligas, "Empowering the IoT heterogenous Wireless Networking with SDR," in *IEEE institute of CS foundation for Research and Technology*, Greece, 2015.

[28] Wikipedia , "Phase-Shift_Keying," [Online]. Available: https://en.wikipedia.org/wiki/Phase-shift_keying. [Accessed 18 02 2020].