Development of Simulation Components for Wireless Communication

by

Ujjwala

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved September 2017 by the
Graduate Supervisory Committee:

Daniel W. Bliss, Chair
Chaitali Chakrabarti
Thomas McGiffen

ARIZONA STATE UNIVERSITY

December 2017

©2017 Ujjwala

All Rights Reserved

ABSTRACT

This thesis work present the simulation of Bluetooth and Wi-Fi radios in real life interference environments. When information is transmitted via communication channels, data may get corrupted due to noise and other channel discrepancies. In order to receive the information safely and correctly, error correction coding schemes are generally employed during the design of communication systems. Usually the simulations of wireless communication systems are done in such a way that they focus on some aspect of communications and neglect the others. The simulators available currently will either do network layer simulations or physical layer level simulations. In many situations, simulations are required which show inter-layer aspects of communication systems. For all such scenarios, a simulation environment, WiscaComm which is based on time-domain samples is built. WiscaComm allows the study of network and physical layer interactions in detail. The advantage of time domain sampling is that it allows the simulation of different radios together which is better than the complex baseband representation of symbols. The environment also supports study of multiple protocols operating simultaneously, which is of increasing importance in today's environment.

DEDICATION

To my loving family and friends

ACKNOWLEDGMENTS

I would like to take this opportunity to thank a few people who's knowledge and assistance have been invaluable to me. First and foremost, I would like to extend my deep gratitude to my advisor Professor Daniel Bliss for mentoring me and providing invaluable insight at every stage of my research. I would also like to thank Professor Chaitali Chakrabarti and Dr. Thomas McGiffen for taking the time to be members of my committee. I would also like to thank Dr. Thomas McGiffen for his constant guidance, patience and support without which this work would have not been easy. I would like to thank Karteek Rupakula and all other WISCA labmates for being a constant support at every stage of my thesis. This work was partially sponsored by Google and School of Electrical Computer and Energy Engineering, ASU under the Google R2 program for which I would like to extend my sincere thanks. The views expressed are those of the author and do not reflect the official policy or position of Google. Last but not the least, I would like to take this opportunity to extend my gratitude to my parents and my sister and other family members for their unconditional love and support which have always been great driving forces for me.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

The current state of wireless communications is both amazing and something of a mess. Our contemporary communications standards are the result of local optimizations over time. These optimizations have led to a set of sub-optimal and fragile wireless standards. In WISCA lab at Arizona State University, we are developing a fluid set of communications standards co-optimized with flexible but power-efficient computational implementations that will enable the next revolution of wireless communications. This advanced fluid protocol standard and mixed Software-Defined Radios (SDR) with hardware accelerators can be used to provide flexibility in data rates and power consumption as per the needs of the user. This will also enable much higher data rates and much lower data rates with corresponding lower power consumption as the needs of the users vary. The SDR system is designed to allow users to easily explore new physical and network layer concepts by employing a simple development interface based on Matlab.

1.1 Software-Defined Radios

With the exponential growth in the ways and means by which people need to communicate, modifying radio devices easily and cost-effectively has become critical. SDR technology brings wide ranging benefits like flexibility, cost efficiency and power to drive communications forward. [10] SDR is a radio communication technology that is based on software defined wireless communication protocols instead of hardwired im-

plementations. In other words, frequency band, air interface protocol and functionality can be upgraded with software download and update instead of a complete hardware replacement. SDR provides an efficient and secure solution to the problem of building multi-mode, multi-band and multi-functional wireless communication devices.

A SDR can be re-programmed or reconfigured to operate with different waveforms and protocols through dynamic loading of new waveforms and protocols. These waveforms and protocols can contain a number of different parts, including modulation techniques, security and performance characteristics defined in software as part of the waveform itself.

1.2 Flexible Protocols

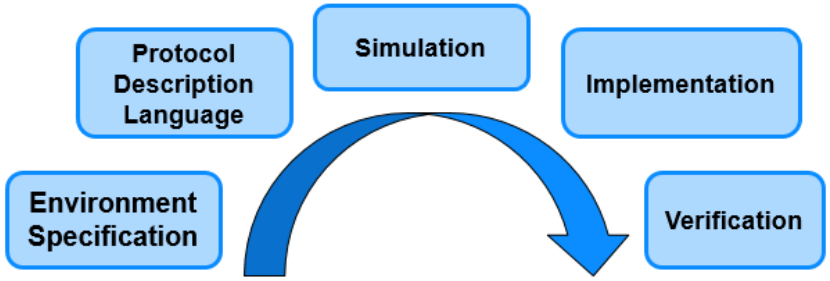

Figure 1. Steps involved in the protocol design process. First the user specifies the environment where the protocol will operate and then the protocol development software applies communication principles over it and come up with recommended parameters. The recommended parameters are then simulated which is followed by hardware implementation. In the end, the protocol is verified.

The popularity of 5G and Internet of Things (IOT) in the recent times clearly signify the importance of wireless systems. 5G provides the benefit of higher data rates. IOT provides diversified benefits like network reliability, optimized resource

consumption and communication diversity etc. As the needs of user vary, the need for different communication protocols increases. [3] Current protocols are rigid. They only provide solution to the current needs of user, ignoring the future needs. Instead of developing a rigid protocol, we propose a flexible or fluid protocol which enables the user to construct wireless systems according to his own needs. The intent of our Protocol Development Kit (PDK) is to automate, and therefore greatly reduce the workload and shorten the time to develop a new wireless protocol, facilitating wireless deployment in new applications.

1.3 PDK Definition and Architecture

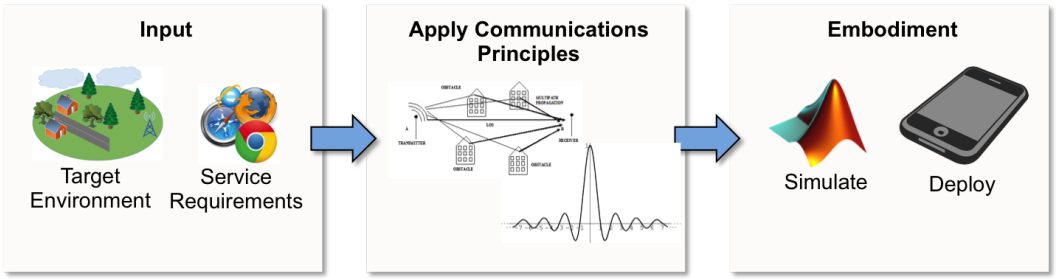

Figure 2. A general protocol design process consist of specifying the inputs according to the user needs which is followed by application of communications principles on the inputs to come up with some recommended parameters on which the protocol will govern and then finally simulation is done based on the recommended outputs from the second step to validate the performance.

As illustrated in Fig. 2, the PDK completes the following steps in developing a protocol recommendation:

- Obtain user input about the proposed operating environment and application, via a graphic user interface (GUI). User input on the environment includes the

transmission range, and if indoors or outdoors. User input on the application includes data rate and transmission power limit.

- Apply wireless principles to translate this input into a protocol recommendation
- Present this recommendation to the user for review and possible revision.
- Simulate radio functions (at various levels)

1.3.1 Simulation Environment: WiscaComm

The user is able to simulate the recommended protocol in an included simulation environment. Upon requesting a simulation, the details of the selected protocol are forwarded to this simulation environment, which appropriately configures the simulation. This environment includes a simulated wireless channel, digital radios supporting various physical layers such as:

- OFDM, with 4-, 16-, and 64- Quadrature Amplitude Modulation (QAM) and variable number of subcarriers
- SC modulation, with 4-, 16-, and 64- QAM, Phase Shift Key (PSK), and Gaussian Mean Shift Key (GMSK) support
- Bluetooth (frequency hopping)

Medium access layers, including:

- time division multiple access (TDMA)
- frequency division multiple access (FDMA)
- carrier sense multiple access (CSMA)

and topologies, including:

- Mesh

- Star
- Peer-to-Peer (P2P)

Figure 3. Steps involved in building a WiscaPDK simulation. First node class is created for a specific radio which the user wants to simulate which is then followed by selection of channel model and then the software writes a simulation configuration file which is used by WiscaComm to set the simulation in action.

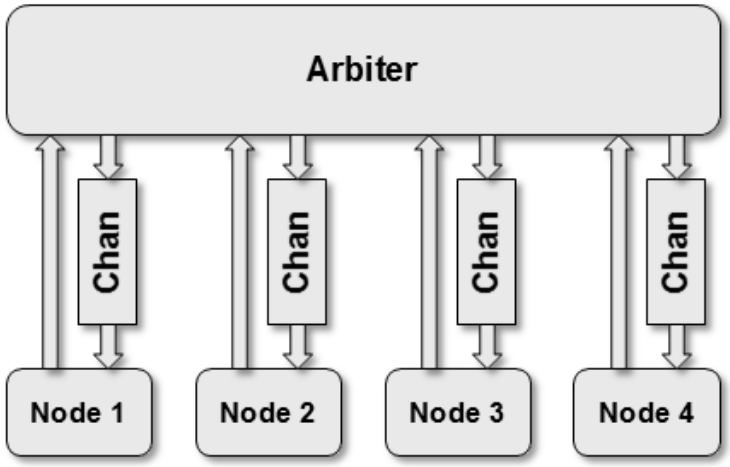

Figure 4. Simulator Architecture

The simulation environment is based on time domain sampling. It allows simultaneous simulation of lower level and higher level details. Because we use time domain sampling, simulating dissimilar radios, radios out of sync, or interference is straightforward. Additionally, modeling a network of radios is straightforward, as that just means managing a stream of samples from each radio at the physical layer, simulating channel processing, then managing a stream of appropriately processed

samples to each receiver. This architecture not only solves studying issues across multiple layers, it also facilitates studying a different radio protocol, as the existing simulation environment is reused, and software development is limited to the different radio. The environment also supports study of multiple protocols operating simultaneously, which is of increasing importance in today's environment. WiscaComm is a lightweight, sample-based platform for simulating wireless protocols and networks. Key features of WiscaComm are as follows:

- Automates wireless signal routing
- Bundles reusable components (processing blocks, channel models)
- Modular, extendable architecture
- Parallelization support

The process to build a typical WiscaComm simulation is shown in the Figure 3. The first step in building a WiscaComm simulation deals with creating a node class for any particular radio. Here the simulator inherits the required properties like carrier frequency, oversampling rate, modulation scheme etc. It can also acquire custom properties like wakeup sequence etc. In step 2, it chooses the channel model which can be Additive White Guassian Noise (AWGN), Rayleigh fading, Rician etc. In step 3, it writes the simulation configuration file using the arbiter as shown in Figure 4 which specifies the number of nodes, classes of nodes, parameters, channel used. WiscaComm uses this information to instantiate node/channel objects and set simulation in motion. Variables should be set in the configuration file, rather than in node's class code. This provides a repeatable, distributable simulation specification. Figures 5, 6 show typical WiscaComm simulation outputs.

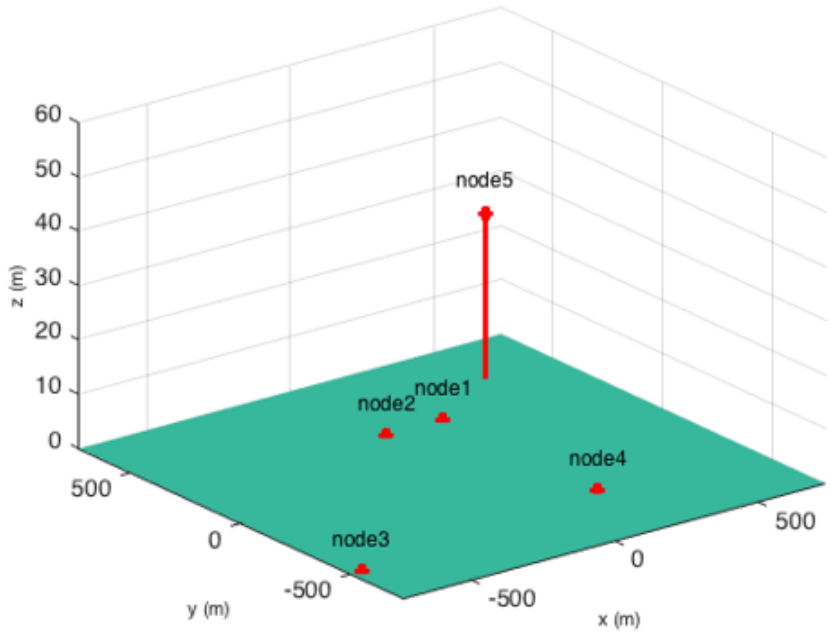

Figure 5. Output from a WiscaComm simulation showing the geographical location of the nodes created for the simulated radio.

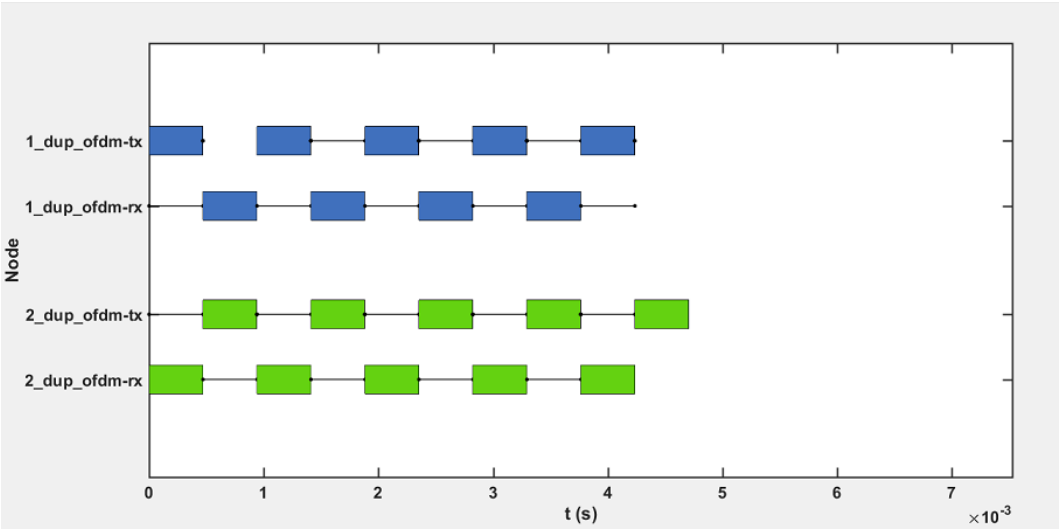

Figure 6. WiscaComm ouptut showing packets being transferred between nodes based on the sequence specified in the configuration file

1.4 Thesis Contribution and Organization

This thesis work is a part of Google ATAP R2 project where a fluid set of communication standards co-optimized with flexible but power-efficient computational implementations are being developed that will enable the next revolution of wireless communications. To simulate the fluid protocols, we have built a simulation environment, WiscaComm which is based on time-domain samples. WiscaComm allows simulation of both lower and higher level details. The environment also supports study of multiple protocols operating simultaneously. Another advantage is that modeling a network of radios is straightforward, as that just means managing a stream of samples from each radio at the physical layer, simulating channel processing, then managing a stream of appropriately processed samples to each receiver. It also facilitates studying a different radio protocol, as the existing simulation environment is reused, and software development is limited to the different radio. As a part of the simulator, I have worked on the physical layer of Bluetooth and Wi-Fi radios. WiscaComm has a common simulation environment for all the radios. As a part of the common simulation environment, I have worked on error correction coding schemes and soft and hard demodulation. Below is the list of my contribution in the project:

- Developed simulation components for wireless communication
- Implemented packet structure of Bluetooth
- Implemented flexible Orthogonal Frequency Division Multiplexing (OFDM) subcarrier density in an OFDM symbol
- Implemented soft and hard demodulation
- Implemented error correction coding like (15,10) hamming code, convolution code and turbo code

The thesis is organized in 5 chapters. Chapter 1, which is the current chapter, gives an introduction about the Google R2 project and also describes the simulator i.e. WiscaComm. Chapter 2 describes the Bluetooth Packet structure. Chapter 3 discusses the Error correction coding schemes like Convolution and Turbo Codes along with soft demodulation. Chapter 4 deals with the implementation of flexible OFDM sub-carrier density in an OFDM symbol. Chapter 5 describes some possible future work for WiscaComm.

Chapter 2

BLUETOOTH PACKET STRUCTURE

In this part of thesis, I have developed and implemented the packet structure for a basic rate Bluetooth packet. The simulation outputs for Bluetooth using WiscaComm are shown in the end of the chapter.

2.1 Introduction

Bluetooth was invented by telecom vendor Ericsson in 1994. Bluetooth is managed by the Bluetooth Special Interest Group (SIG). Bluetooth was standardized as 802.15.1 by Institute of Electrical and Electronics Engineers (IEEE) . [16]
Radio waves operating in the unlicensed Industrial Scientific and Medical (ISM) band with 2.45 GHz frequency are used as the communication medium in Bluetooth. Bluetooth uses a Frequency-Hopping Spread Spectrum (FHSS) technique which takes a narrow band signal and spreads it over a broader portion of the available radio frequency band. A frequency hop transceiver is applied to combat interference and fading. In FHSS, the total available frequency is split into many channels of smaller bandwidth. Transmitter and receiver stay on one of these channels for a certain period and then hop to another channel. The channel usage pattern is called hopping pattern. Bluetooth divides transmitted data into packets, and transmits each packet on one of 79 designated Bluetooth channels. There is a frequency selection module that gives the frequency depending on masters clock. The hopping rate is 1600 hopes/sec. Bluetooth is compatible across a range of different operating systems and devices.

2.2 Bluetooth Baseband

The Baseband is the physical layer of the Bluetooth. It manages physical channels and links. It also provides other services like error correction, data whitening, hop selection and security. The Baseband layer lies on top of the Bluetooth radio layer in the Bluetooth stack. The baseband transceiver applies a time-division duplex (TDD) scheme. Therefore, apart from different hopping frequency (frequency division), the time is also slotted. This section describes the radio frequency specification requirements for a Bluetooth transceiver. These requirements are taken from the Bluetooth Core 4.2 specification.

2.2.1 Bluetooth Baseband Normal Operation

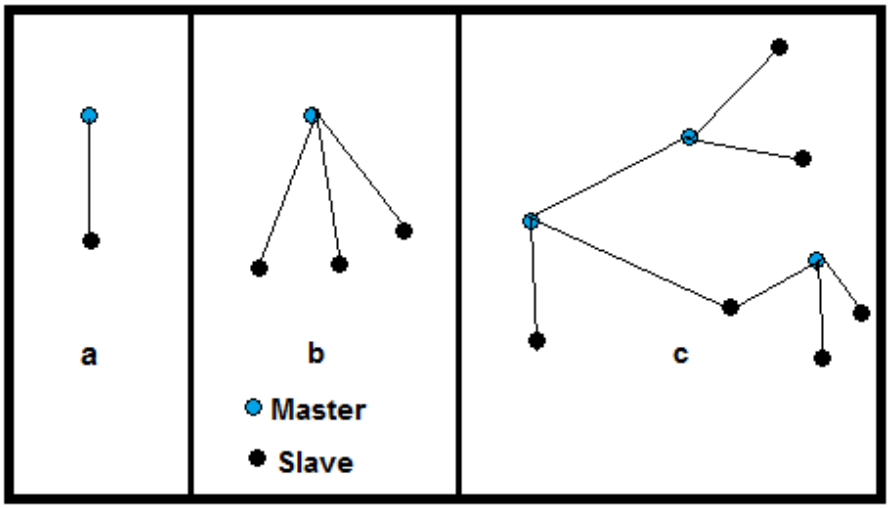

Figure 7. (a) Piconets with a single slave operation, (b) a multi-slave operation and (c) a scatternet operation

The Bluetooth system provides a point-to-point connection or a point to multipoint connection, see (a) and (b) in Figure 7. In a point to point connection, the physical channel is shared between two Bluetooth devices. In a point-to-multipoint connection, the physical channel is shared among several Bluetooth devices. Two or more devices sharing the same physical channel form a piconet. One Bluetooth device acts as the master of the piconet, whereas the other device(s) act as slave(s). Up to seven slaves can be active in the piconet. Many more slaves can remain connected in a parked state. These parked slaves are not active on the channel, but remain synchronized to the master and can become active without using the connection establishment procedure. Both for active and parked slaves, the channel access is controlled by the master. An unlimited number of slaves can receive data on the physical channel carrying the Connectionless Slave Broadcast physical link. [14]
Piconets that have common devices are called a scatternet, see (c) in Figure 7. Each piconet only has a single master, however, slaves can participate in different piconets on a time-division multiplex basis. In addition, a master in one piconet can be a slave in other piconets. Piconets shall not be frequency synchronized and each piconet has its own hopping sequence. [14]

2.2.2 Modes of Operation

Data is transmitted over the air in packets. Two modes are defined:

- Basic Rate
- Enhanced Data Rate

The symbol rate for all modulation modes is 1 Ms/s. The gross air data rate is 1 Mb/s for Basic Rate. [14]

2.2.2.1 Basic Rate Bluetooth Packet

Figure 8. Basic Rate Bluetooth Packet format

In this thesis work, I have designed the baseband Bluetooth for the basic rate bluetooth packet. The general Basic Rate packet format is shown in Figure 8. Each packet consists of 3 entities:

- The access code (72)
- The header (54)
- The payload $(0 - 2790)$

2.2.2.2 Bluetooth Device Address Format

Each Bluetooth device is allocated a unique 48-bit Bluetooth device address. The Lower Address Part (LAP) and Upper Address Part (UAP) form the significant part of the Bluetooth Device Address. The bit pattern in Figure 9 is an example of Bluetooth Device Address. The Bluetooth Device Address may take any values except the 64 reserved LAP values for general and dedicated inquiries. [14]

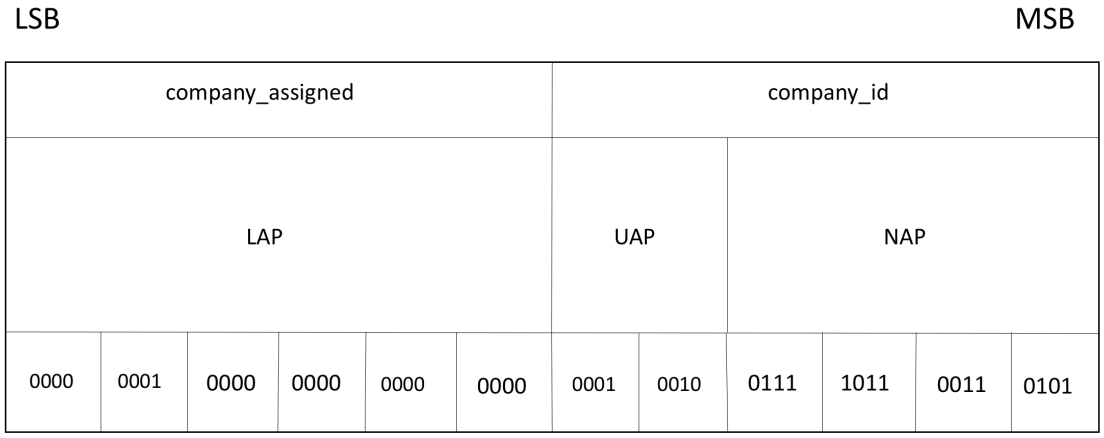

Figure 9. Bluetooth Device Address

2.3 Physical Channels

All Bluetooth physical channels are characterized by the combination of a pseudo-random frequency hopping sequence, the specific slot timing of the transmissions, the access code and packet header encoding. These aspects, together with the range of the transmitters, define the signature of the physical channel. [14]

Frequency hopping is used to change frequency periodically to reduce the effects of interference. Given that the number of RF carriers is limited and that many Bluetooth devices could be operating independently within the same spatial and temporal area, there is a strong likelihood of two independent Bluetooth devices having their transceivers tuned to the same RF carrier, resulting in a physical channel collision. To mitigate the unwanted effects of this collision each transmission on a physical channel starts with an access code that is used as a correlation code by devices tuned to the physical channel. This channel access code is a property of the physical channel. The access code is always present at the start of every transmitted packet. [14]

There are five Bluetooth physical channels defined. Two of these physical channels (the basic piconet channel and adapted piconet channel) are used for communication between connected devices and are associated with a specific piconet. Other physical channels are used for discovering (the inquiry scan channel) and connecting (the page scan channel) Bluetooth devices. A Bluetooth device can only use one of these physical channels at any given time. In order to support multiple concurrent operations, the device uses time division multiplexing between the channels. To connect to a channel, a Bluetooth device is synchronized to the timing, frequency and access code of a physical channel. [14]

2.4 Packet Structure

The packet structure for a basic rate packet used by Bluetooth devices is defined in this section and is shown in figure 8. The bit ordering when defining packets and messages in the Baseband Specification, follows the Little Endian format. Least Significant Bit (LSB) is sent first over the air.

2.4.1 Access Code

Every Bluetooth packet starts with an access code. If a packet header follows, the access code is 72 bits long, otherwise the access code is 68 bits long and is known as a shortened access code. This access code is used for synchronization, DC offset compensation and identification. The access code identifies all packets exchanged on a physical channel: all packets sent in the same physical channel are preceded by the same access code. In the receiver of the device, a sliding correlator correlates against

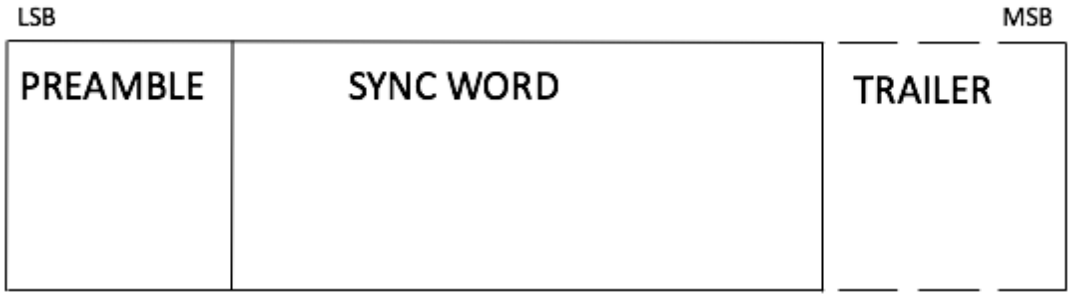

Figure 10. Access Code Structure in Bluetooth

the access code and triggers when a threshold is exceeded. The access code consists of a preamble, a sync word, and possibly a trailer. [14]

2.4.1.1 Preamble

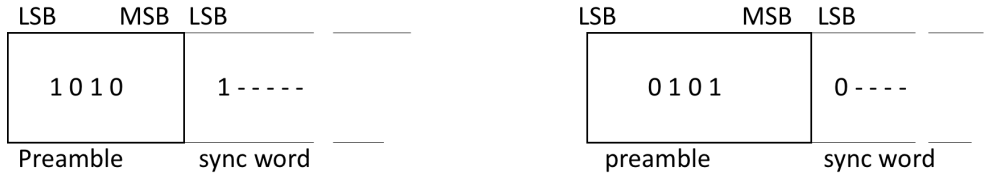

Figure 11. Preamble in Bluetooth

The preamble is a fixed zero-one pattern of 4 symbols. To facilitate DC compensation, a preamble is used. The sequence is either 1010 or 0101 (in transmission order), depending on whether the Least Significant Bit of the following sync word is 1 or 0 respectively. [14]

2.4.1.2 Sync Word

The sync word is a 64−bit code word derived from a 24 bit address (LAP). The proper construction guarantees large Hamming distance between sync words based on different LAPs. In addition, the good auto correlation properties of the sync word improve timing acquisition. The sync words are based on a $(64, 30)$ expurgated block code with an overlay (bit-wise XOR) of a 64 bit full length pseudo-random noise (PN) sequence. The expurgated code guarantees large Hamming distance ($d_{min} = 14$) between sync words based on different addresses. The PN sequence improves the auto correlation properties of the access code. The following steps describe how the sync word shall be generated:

1. Generate information sequence
2. XOR this with the "information covering" part of the PN overlay sequence
3. Generate the codeword
4. XOR the codeword with all 64 bits of the PN overlay sequence.

The information sequence is generated by appending 6 bits to the 24 bit LAP (step 1). The appended bits are 001101 if the MSB of the LAP equals 0. If the MSB of the LAP is 1 the appended bits are 110010. The LAP MSB together with the appended bits constitute a length-seven Barker sequence. The purpose of including a Barker sequence is to further improve the auto correlation properties. In step 2 the information is pre-scrambled by exoring it with the bits $p_{34}.....p_{63}$ of the PN sequence. After generating the codeword (step 3), the complete PN sequence is exored to the codeword (step 4). This step de-scrambles the information part of the codeword. At the same time the parity bits of the codeword are scrambled. Consequently, the

original LAP and Barker sequence are ensured a role as a part of the access code sync word, and the cyclic properties of the underlying code is removed. [14]

2.4.1.3 Trailer

Figure 12. Bluetooth trailer structure when most significant bit of sync word is 0 (a), and when most significant bit of sync word is 1 (b)

The trailer is appended to the sync word as soon as the packet header follows the access code. The trailer is a fixed zero-one pattern of four symbols. The trailer together with the three MSBs of the sync word form a 7-bit pattern of alternating ones and zeroes which can be used for extended DC compensation. The trailer sequence is either 1010 or 0101 (in transmission order) depending on whether the MSB of the sync word is 0 or 1, respectively. [14]

2.4.2 Packet Header

The header contains link control (LC) information and consists of 6 fields as shown in figure 13. The total header, including the Header Error Check (HEC), consists of 18 bits and is encoded with a rate $\frac{1}{3}$ Forward Error Correction (FEC) resulting in a 54-bit header. [14]

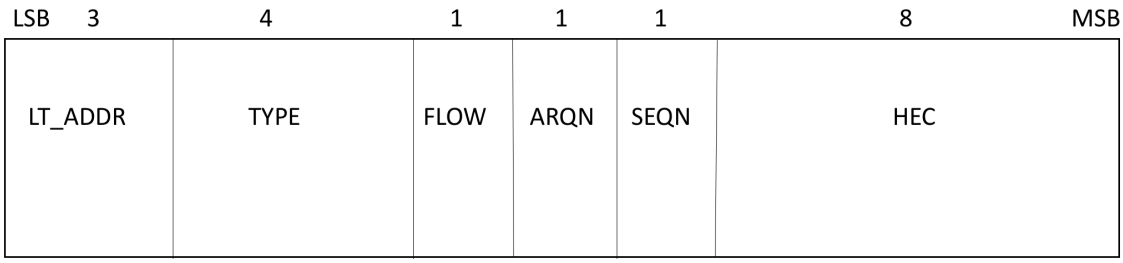

Figure 13. Bluetooth packet header structure

2.4.2.1 LT_ADDR

The 3-bit LT_ADDR field contains the logical transport address for the packet. This field indicates the destination slave or slaves in the case of a broadcast for a packet in a master-to-slave transmission slot and indicates the source slave for a slave-to-master transmission slot. [14]

2.4.2.2 TYPE

Sixteen different types of Bluetooth packets can be distinguished with this field. The 4-bit TYPE code specifies which packet type is used. The TYPE code determines how many slots the current packet will occupy. [14]

2.4.2.3 FLOW

The FLOW bit is set to 0 in my packet structure at transmitter side since I have designed the packet structure for the basic rate Connectionless Slave Broadcast (CSB) logical transport. It is ignored at the receiver end. [14]

2.4.2.4 ARQN

The 1-bit acknowledgment indication ARQN is used to inform the source of a successful transfer of payload data with Cyclic Redundancy Check (CRC), and can be Positively Acknowledged (ACK) or Negatively Acknowledged (NAK). The ARQN bit is not used on the CSB logical transport and shall be set to 0 on transmission and ignored upon receipt. [14]

2.4.2.5 SEQN

The SEQN bit provides a sequential numbering scheme to order the data packet stream. The SEQN bit is not used on the CSB logical transport and shall be set to 0 on transmission and ignored upon receipt. [14]

2.4.2.6 Header Error Check (HEC)

The HEC is a 8 bit word used to check the header integrity. The HEC generator is initialized with the first 8 bits of the upper address part of the master device. After this, HEC is calculated for 10 header bits. Before checking HEC, the receiver initialises the HEC check circuitry with the proper 8 bits. If the HEC doesn't check properly then the entire packet is discarded. [14]

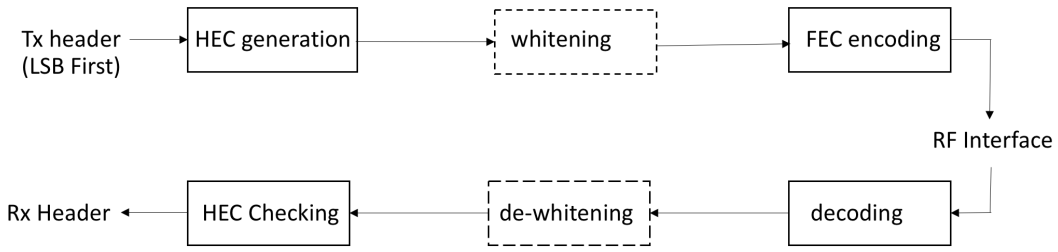

Figure 14. Header Bit Processing in Bluetooth

2.5 Bitstream Processing

Before the payload is sent over the air interface, several bit manipulations are performed in the transmitter to increase reliability and security. An HEC is added to the packet header, the header bits are scrambled with a whitening word, and Forward Error Correction (FEC) coding is applied. These steps are showed in figure 14. In the receiver, the inverse processes are carried out. All header bit processes are mandatory except that whitening and de-whitening shall not be performed on synchronization train packets. Processes not performed on all packet types are indicated by dashed blocks. In addition to the processes carried out for header we can do encryption as well for the payload. [14]

2.5.1 Error Checking

The packet can be checked for errors or wrong delivery using the channel access code, the HEC in the header, and the CRC in the payload. When the packet is received, first the access code is checked. Since the 64-bit sync word in the channel

access code is derived from the 24-bit master LAP, this checks if the LAP is correct, and prevents the receiver from accepting a packet of another piconet. [14]

2.5.1.1 HEC Generation

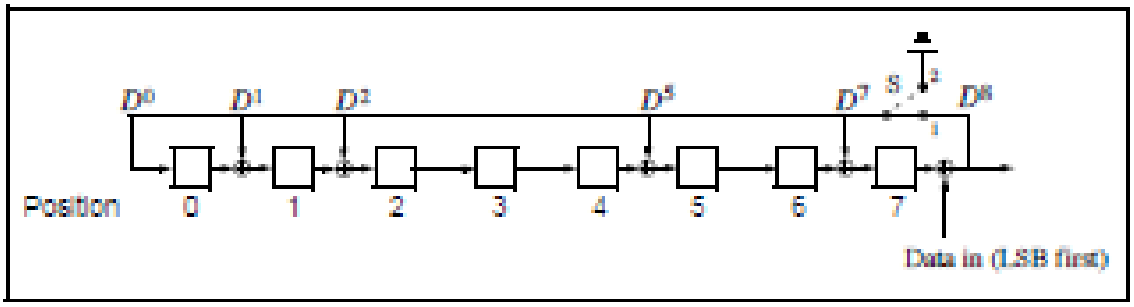

Figure 15. The linear feedback shift register circuit generating the header error check in Bluetooth [14]

The generator polynomial is $g(D) = (D + 1)(D^7 + D^4 + D^3 + D^2 + 1)$. Initially this circuit shall be pre-loaded with the 8-bit UAP such that the LSB of the UAP (denoted UAP_0) goes to the left-most shift register element, and, UAP_7 goes to the right-most element. The initial state of the Linear Feedback Shift Register (LFSR) which generates HEC is shown in figure 16. Then the data is shifted with switch S set in position 1. When the last data bit has been clocked in the LFSR, the switch S shall be set in position 2, and, the HEC can be read out from the register. The LFSR bits shall be read out from right to left. [14]

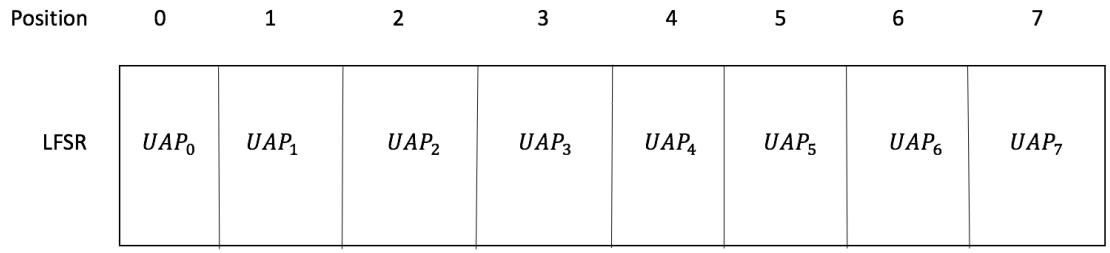

Figure 16. Initial state of the header error check generating circuit in Bluetooth

Figure 17. Linear feedback shift register circuit generating the cyclic redundancy check for payload in Bluetooth [14]

2.5.1.2 CRC Generation

The 16 bit LFSR for the CRC is constructed using the generator polynomial:

$$g(D) = \mathrm{D}^{16} + D^{12} + D^5 + 1 \tag{2.1}$$

The eight left-most bits shall be initially loaded with the 8-bit Upper Address Part (UAP_0 to the left and UAP_7 to the right) while the 8 right-most bits shall be reset to zero. The initial state of the LFSR which generates CRC is shown in figure 18. When the data is shifted in then the switch S is held at position 1. When the last bit has entered the LFSR then after that switch S is set in position 2. The CRC can now be read out from the register. The contents of the register are read from right to left. [14]

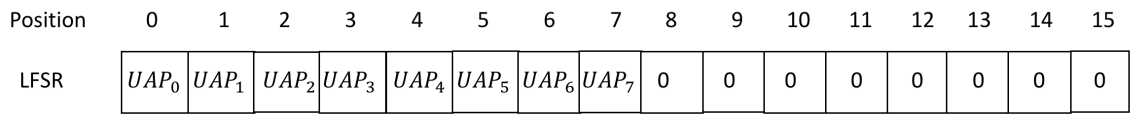

Figure 18. Initial State of the cyclic redundancy check generating circuit

2.5.2 Data Whitening

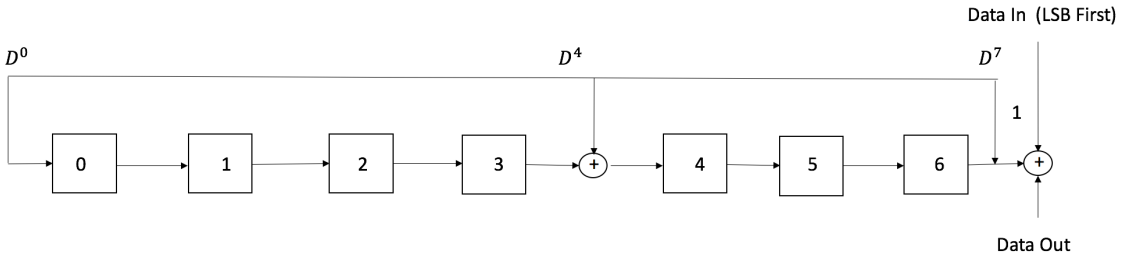

Figure 19. Data whitening linear feedback shift register in Bluetooth

To randomize the data from highly redundant patterns and to minimize DC bias in the packet, before the transmission of the packets, both the header and the payload are scrambled with a data whitening word. The scrambling is performed prior to the Forward Error Correction encoding. At the receiver, the received data of scrambled packets are descrambled using the same whitening word generated in the recipient. The descrambling is performed after Forward Error Correction decoding. The whitening word is generated with the polynomial $g(D) = D^7 + D^4 + 1$. It is subsequently exored with the header and the payload. The whitening word is generated with the LFSR shown in the figure below. There is no re-initialization of the LFSR between data header and payload. [14]

2.6 Block Codes

Block codes are error correction coding scheme that map a fixed number of message symbols to a fixed number of code symbols. Its a memoryless device as it treats each block independently. Each block of k message symbols is encoded into a codeword that consists of n message symbols where k is called the message length, n is called the codeword length, and the code is called an (n,k) code. Examples of block codes are Reed–Solomon codes, Hamming codes, Hadamard codes, Expander codes, Golay codes, and Reed–Muller codes. These examples also belong to the class of linear codes, and hence they are called linear block codes.

Hamming codes are a family of linear error-correcting codes. They were invented by Richard Hamming in 1950. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. For Hamming codes, the codeword length n must have the form 2^m-1, where m is an integer greater than or equal to 3. The message length k must equal n-m. Encoding a message using a generic linear block code requires a generator matrix. The process of encoding a message into an [n,k] linear block code is determined by a K-by-N generator matrix G. A 1-by-k message vector v is encoded into the 1-by-n codeword vector vG. If G has the form [I_k, P] or [P, I_k], where P is some k-by-(n-k) matrix and I_k is the k-by-k identity matrix, G is said to be in standard form.

A parity-check matrix of a linear block code is a matrix which describes the linear relations that the components of a codeword must satisfy. It can be used to decide if a particular vector is a codeword. It can also be used in decoding algorithms.

The parity check matrix, H, for a given code can be derived from its generator matrix (and vice versa).[13] If the generator matrix for an [n,k]-code is in standard

form

$$G = [\mathrm{I}_k|\mathrm{P}] \tag{2.2}$$

then the parity check matrix is given by

$$\mathrm{H}=[\text{-}\mathrm{P}^{\top}|I_{n-k}] \tag{2.3}$$

because

$$G\mathrm{H}^{T}=\mathrm{P}\text{-}\mathrm{P}=0 \tag{2.4}$$

A decoding table tells a decoder how to correct errors that would have corrupted the code during transmission. Hamming codes can correct any single-symbol error in any codeword. I have used the syndrome table technique to detect the location of an error and then correct it. Syndrome table is a decoding table for an error-correcting binary code having codeword length n and message length k. H is an (n-k)-by-n parity-check matrix for the code. T is a 2n-k-by-n binary matrix. The r^{th} row of T is an error pattern for a received binary codeword whose syndrome has decimal integer value r-1. The syndrome of a received codeword is its product with the transpose of the parity-check matrix. [8]

2.7 Error Correction

There are three error correction schemes defined for Bluetooth:

- $\frac{1}{3}$ rate Forward Error Correction (FEC)
- $\frac{2}{3}$ rate FEC
- ARQ scheme for the data

The purpose of the FEC scheme on the data payload is to reduce the number of retransmissions.

2.7.1 FEC Code: Rate $\frac{1}{3}$

This simple 3-times repetition FEC code is used for the header.

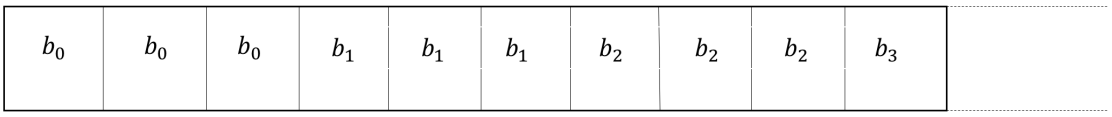

Figure 20. Bit repetition encoding for header bits in Bluetooth

2.7.2 FEC Code: Rate $\frac{2}{3}$

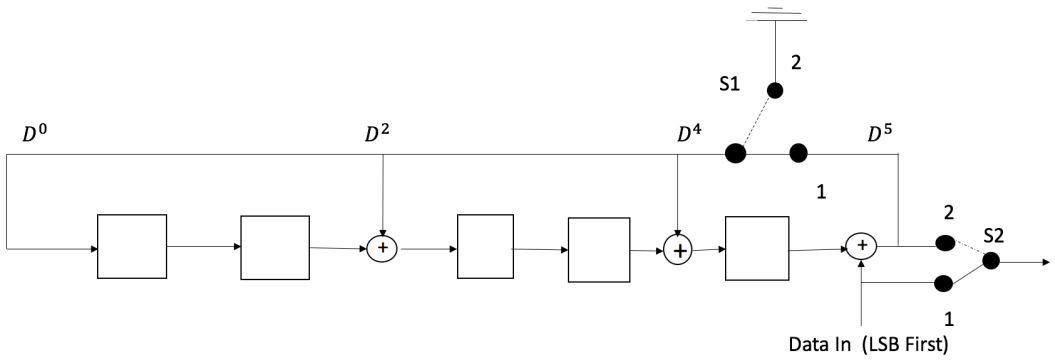

Figure 21. Linear feedback shift register generating the $(15, 10)$ shortened hamming code in Bluetooth

The other FEC scheme used for bluetooth is a (15,10) shortened Hamming code. The generator polynomial used for it is $g(D) = (D + 1)(D^4 + D + 1)$. Initially all register elements are set to zero. The 10 information bits are sequentially fed into the LFSR with the switches S1 and S2 set in position 1. Then, after the final input bit, the switches S1 and S2 are set in position 2, and the five parity bits are shifted out.

The parity bits are appended to the information bits. Subsequently, each block of 10 information bits is encoded into a 15 bit codeword. This code can correct all single errors and detect all double errors in each codeword. [14] The generator matrix which I generated using the above LFSR circuit was obtained by individually sending each of the 10 rows of a 10x10 Identity matrix as an input to the LFSR circuit. Thus I got the parity bits for each row of the generator matrix. At the receiver side while decoding the received data, I have used the same generator matrix to obtain the parity check matrix and subsequently the syndrome table to detect the location of errors.

2.8 Modulation and data transmission

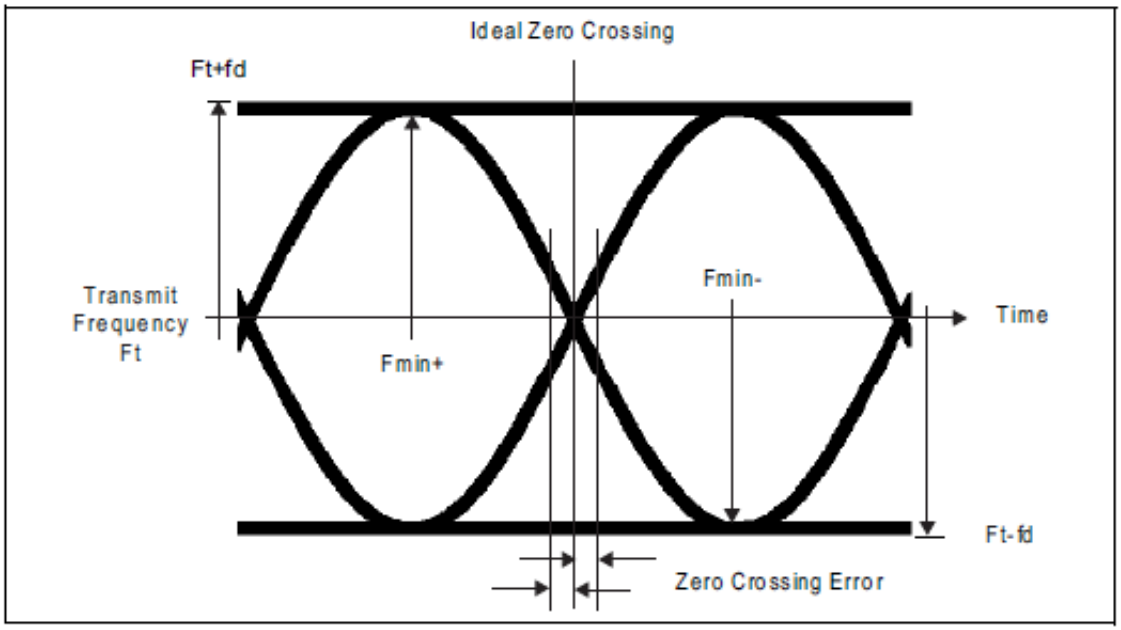

Figure 22. Gaussian frequency shift keying parameters definition [14]

The Modulation technique used is GFSK (Gaussian Frequency Shift Keying) with

a bandwidth bit period product BT=0.5. The Modulation index is between 0.28 and 0.35. A binary one is represented by a positive frequency deviation, and a binary zero is represented by a negative frequency deviation. The symbol timing is less than ±20 ppm. After the modulation, the modulated signal is frequency hopped and upsampled and then sent across the channel. [14]

2.9 FEC Code: Rate $\frac{1}{3}$ Decoder

Repetition decoding is usually done using Majority logic detection. To determine the value of a particular bit, we look at the received copies of the bit in the stream and choose the value that occurs more frequently. In this thesis work I have used a threshold value of 0.5 to differentiate between a 0 and a 1. Since at the transmitter side, I used the three time repetition coding, so at the receiver, I am using a group of three bits at a time and summing it up and dividing by 3. If the value is above 0.5 then the received bit is a 1 else its a 0.

2.10 FEC Code: Rate $\frac{2}{3}$ Decoder

The same generator matrix is used at the receiver side to decode the received payload which was used at transmitter side to encode the payload. I have used the syndrome table technique to correct the errors in the payload by using the parity check matrix. The syndrome of a received codeword is its product with the transpose of the parity-check matrix. The syndrome table to decode the codeword for a given parity check matrix is generated using matlab. The decoding table helps determine

the correction vector. The corrected codeword is the sum (modulo 2) of the correction vector and the received codeword.

2.11 Simulation Results

Below are the simulation results for bluetooth in P2P mode. There is only 1 master and slave respectively. We can see from the simulation output that two Bluetooth nodes are created.

- Simulation output of packets being transferred

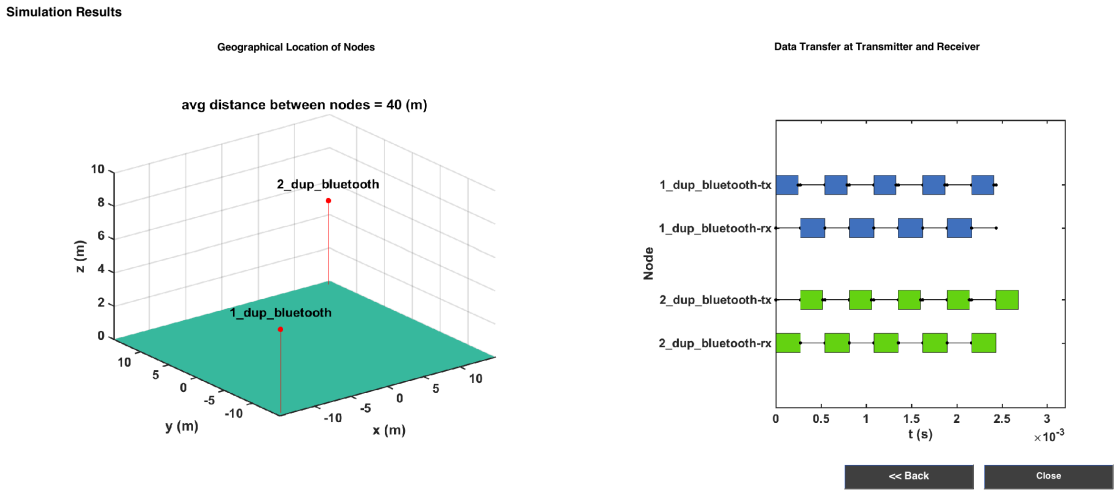

Figure 23. Simulation output of WiscaComm showing geographical location of the two Bluetooth nodes created in the left figure. The figure on right shows the packets being transferred between the Bluetooth nodes. When the first packet is being transferred by 1_dup_bluetooth_tx node then the corresponding packet is being received by 2_dup_bluetooth_rx node. Likewise we can see how the other packets are being transferred between the nodes.

- Transmit spectrum

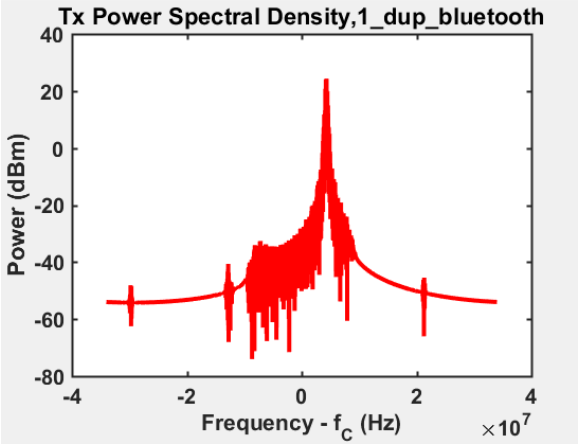

Figure 24. Spectrum at transmitter end for the Bluetooth simulation in peer to peer mode. Bluetooth uses frequency hopping scheme. The value of the peak is magnitude squared of the absolute value of the Fast Fourier Transform (FFT) of the signal and normalized according to the FFT size at that frequency. The side lobe is around 40dB lower than the peak.

- Received Spectrum

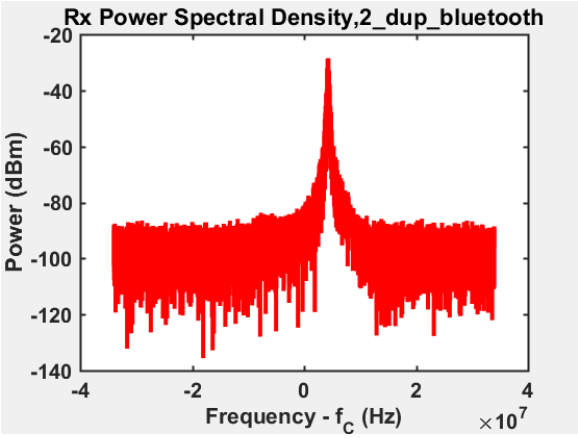

Figure 25. Spectrum at the receiver end for the Bluetooth simulation in peer to peer mode. The peak is centered around 6 Mhz. The side lobe is around 40dB lower than the peak.

- Bit Error Rate Plot for the receiver

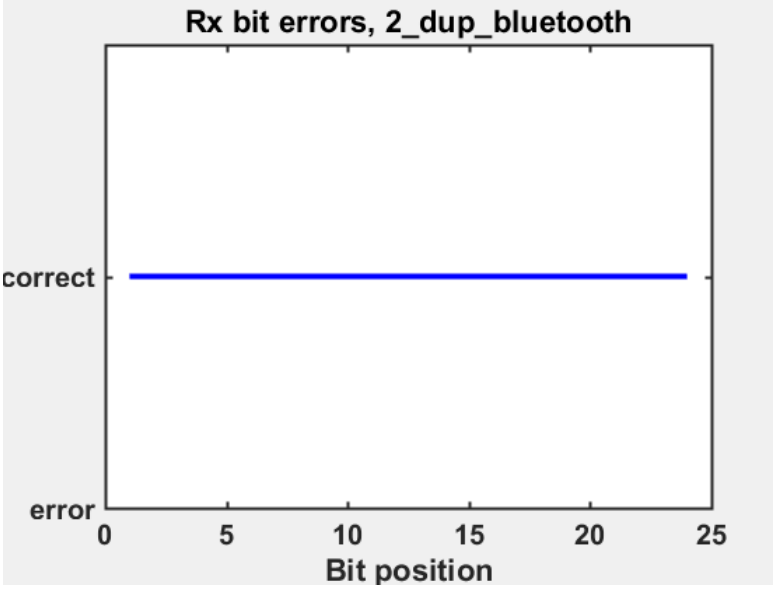

Figure 26. The bit error rate plot at receiver end for the Bluetooth simulation. The graph clearly shows that all the bits are received correctly and without any error.

Chapter 3

CHANNEL CODING

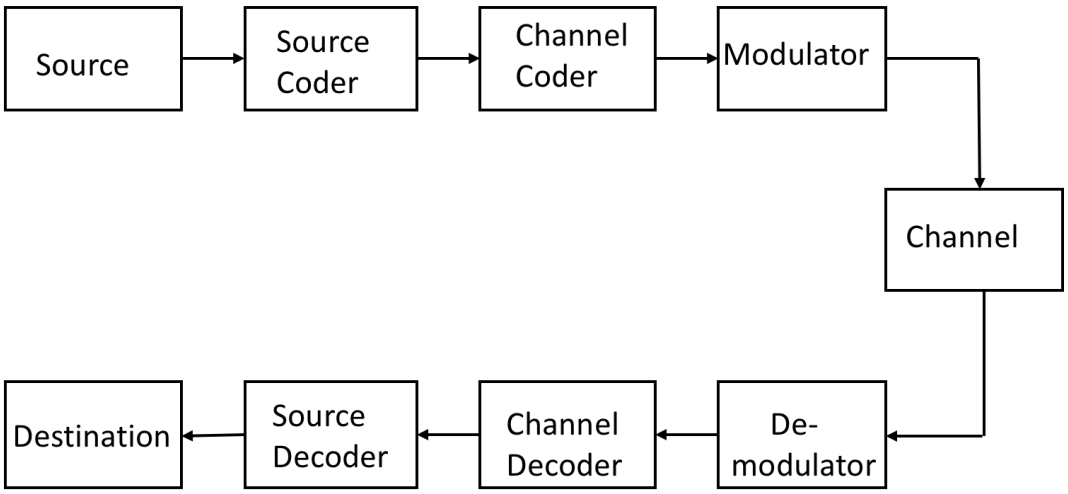

Figure 27. Block Diagram of a basic communication System

In communication systems, the information is represented as a sequence of binary bits. The binary bits are then mapped to analog signal waveforms which are then transmitted over a communication channel. The communication channel can introduce noise and interference and thus corrupt the transmitted signal. At the receiver, the received signal which is channel corrupted is mapped back to binary bits. The received binary information is an estimate of the transmitted binary information. Bit errors may result due to the transmission and the number of bit errors depends on the amount of noise and interference in the communication channel. [4] Channel coding is used in digital communication systems to protect the digital information from noise

and interference and reduce the number of bit errors. Channel coding is done by introducing redundant bits into the transmitted information stream which help in detection and correction of bit errors in the received data stream on the expense of reduction in data rate or an expansion in bandwidth. [4]

3.1 Types of Coding

There are two main types of channel codes, namely:

- Block Codes
- Convolution Codes

A (n,k) block code accept a block of k information bits and produce a block of n coded bits. They can be used to either detect or correct errors. Some of the commonly used block codes are Hamming codes, Golay codes, BCH codes, and Reed Solomon codes.

Convolutional codes are one of the most widely used channel codes in practical communication systems. [4] The encoded bits depend not only on the current k input bits but also on past input bits. Convolution codes convert the entire data stream into one single codeword.

A near channel capacity error correcting code called turbo code was introduced in nineties. Turbo codes are able to transmit information across the channel with arbitrary low (approaching zero) bit error rate. [11] Turbo code is a parallel concatenation of two component convolutional codes separated by a random interleaver. [7] Both constituent decoders use the same trellis structure and algorithm. Turbo codes are widely used in deep space communications, third generation wireless standards, and digital video broadcasting.

3.2 Convolution Codes

This section describes the encoder and decoder structures for convolutional codes. Convolutional codes are linear codes which involve the computation of parity bits from message bits and their transmission. These codes were invented by Peter Elias '44, an MIT EECS faculty member, in the mid-1950s. In a convolutional code, the sender sends only the parity bits. The main decoding strategy for convolutional codes, based on the Viterbi Algorithm, is described. Convolution Codes are used in a variety of systems including:

- Wireless standards (such as 802.11)
- Satellite communications
- Building block of more powerful modern codes, such as turbo codes, which are used in wide-area cellular wireless network standards such as 3G, LTE, and 4G

3.2.1 Convolution Encoder

The encoder uses a sliding window to calculate parity bits by combining various subsets of bits in the window, the windows overlap and slide by 1. The size of the window, in bits, is called the code's constraint length. The longer the constraint length, the larger the number of parity bits that are influenced by any given message bit and a larger constraint length generally implies a greater resilience to bit errors. [2]

If a convolutional code produces r parity bits per window and slides the window forward by one bit at a time, its rate (when calculated over long messages) is $\frac{1}{r}$. The greater the value of r, the higher the resilience of bit errors, but the trade-off is that

a proportionally higher amount of communication bandwidth is devoted to coding overhead. In practice, we would like to pick r and the constraint length to be as small as possible while providing a low enough resulting probability of a bit error. I will use "K" for the constraint length. The encoder looks at K bits at a time and produces r parity bits according to carefully chosen functions that operate over various subsets of the K bits. [2]

Parity equations govern the way in which parity bits are produced from the sequence of message bits. One can view each parity equation as being produced by combining the message bits and a generator polynomial. Each parity equation is a convolution of generator polynomial and message bits; hence the term "convolutional code". [2]

3.2.1.1 Representation of Convolution Encoder

I have used the Matlab Communication system Toolbox System object i.e. comm.ConvolutionalEncoder for describing the structure of the convolution encoder. There are two different representations of convolution encoder namely, generator polynomial and trellis structure respectively. I have used the generator polynomial to describe the structure of the Convolution Encoder.

A polynomial description of a convolutional encoder describes the connections among shift registers and modulo 2 adders. For example, Figure 28 depicts a convolutional encoder that has one input, two outputs, and two shift registers. A convolutional code introduces redundant bits into the data stream through the use of linear shift registers as shown in Figure 28. The code rate r for a convolutional code is defined as

$$r = \frac{k}{n} \tag{3.1}$$

where k is the number of parallel input information bits and n is the number of parallel output encoded bits at one time interval. [4]

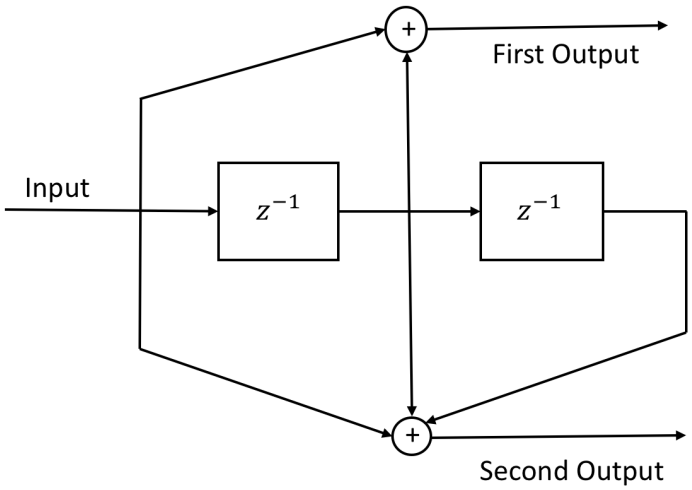

Figure 28. Polynomial description of a convolution code

A polynomial description of a convolutional encoder has either two or three components as follows, depending on whether the encoder is a feedforward or feedback type:

- Constraint Length
- Generator Polynomial
- Feedback Connection Polynomial (for feedback encoders only)

The constraint length of the encoder form a vector whose length is the number of inputs in the encoder diagram. The elements of this vector indicate the number of bits stored in each shift register, including the current input bits. The constraint length K for a convolutional code is defined as

$$K = m + 1 \tag{3.2}$$

where m is the maximum number of stages (memory size) in any shift register. It is a scalar because the encoder has one input stream. For example, for a rate 1/3 convolution code, I have used the following parameters:

Constraint length =7

Generator Polynomial = [133 171 165]

3.2.2 Convolution Decoder: Hard and Soft Decoding

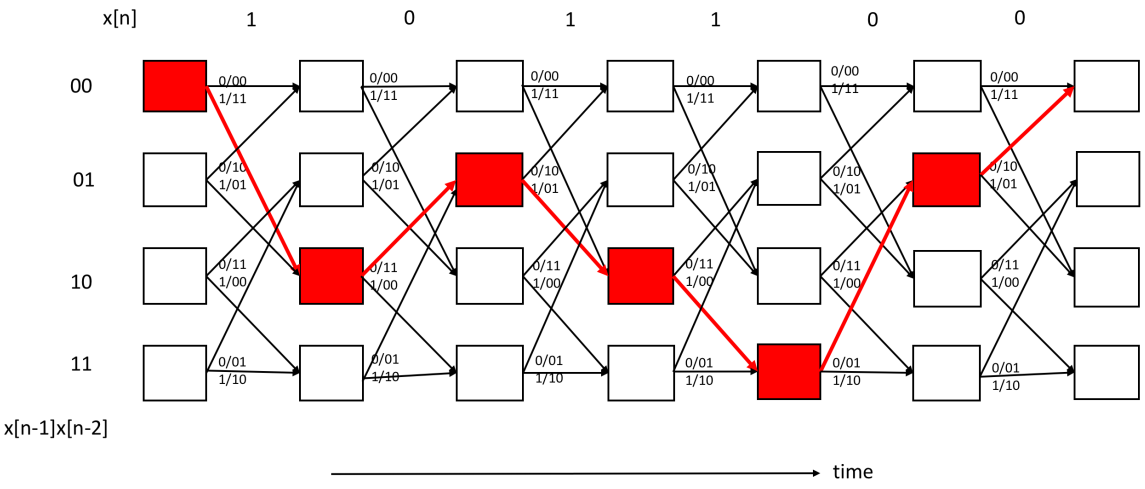

Figure 29. Trellis Diagram

For decoding convolution codes, I have used Viterbi Decoder using the Matlab communication system tool box system object comm.ViterbiDecoder. This decoding method uses a special structure called the trellis as shown in Figure 29. Each column of the trellis has the set of states; each state in a column is connected to two states in

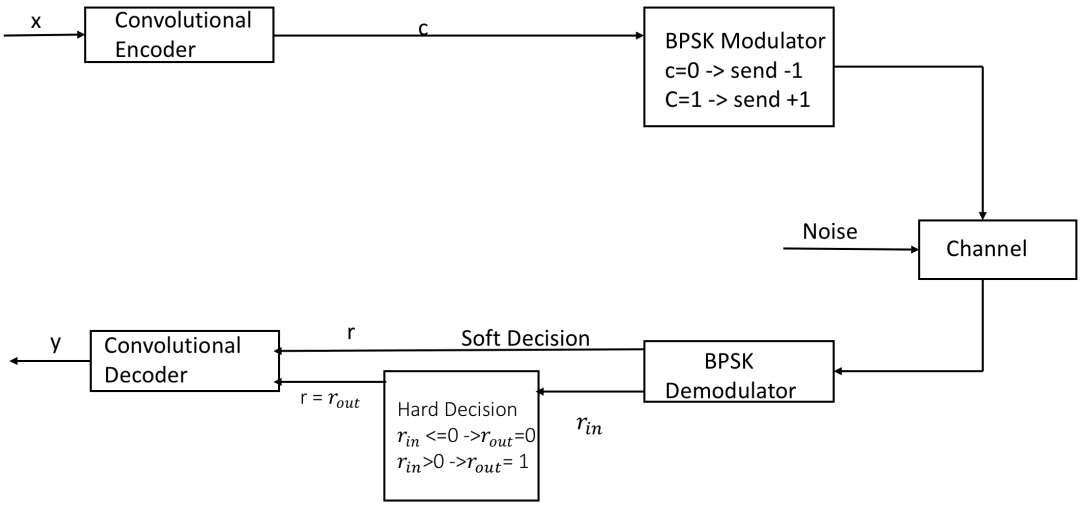

Figure 30. Hard and soft decision decoding

the next column. The top link from each state in a column of the trellis shows what gets transmitted on a 0, while the bottom shows what gets transmitted on a 1. We can think what the decoder needs to do in terms of this trellis. It gets a sequence of parity bits, and needs to determine the best path through the trellis, that is, the sequence of states in the trellis that can explain the observed, and possibly corrupted, sequence of received parity bits. The Viterbi decoder finds a maximum-likelihood path through the trellis. [2]

Hard decision and soft decision decoding refer to the type of quantization used on the received bits. Hard decision decoding uses 1 bit quantization on the received channel values. Soft decision decoding uses multi bit quantization on the received channel values. For the ideal soft-decision decoding (infinite-bit quantization), the received channel values are directly used in the channel decoder. [4] Figure 30 shows hard and soft decision decoding.

3.2.2.1 Hard Decoding Viterbi Algorithm

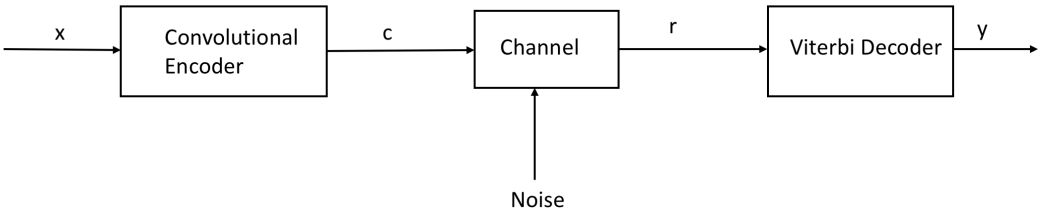

Figure 31. Convolution Code System

The Viterbi algorithm computes a maximum likelihood (ML) estimate on the estimated code sequence **y** from the received sequence **r** such that it maximizes the probability $p(\mathbf{r}|\mathbf{y})$ that sequence **r** is received conditioned on the estimated code sequence **y**. Sequence **y** must be one of the allowable code sequences and cannot be any arbitrary sequence. Figure 31 shows the described system structure. [4] For a rate r convolutional code, the encoder inputs k bits in parallel and outputs n bits in parallel at each time step. The input sequence is denoted as

$$x = (x_0(1), x_0(2), ..., x_0(k), x_1(1), ..., x_1(k), x_{L+m-1}(1), ..., x_{L+m-1}(k)) \tag{3.3}$$

and the coded sequence is denoted as

$$c = (c_0(1), c_0(2), ..., c_0(n), c_1(1), ..., c_1(n), c_{L+m-1}(1), ..., c_{L+m-1}(n)) \tag{3.4}$$

where L denotes the length of input information sequence and m denotes the maximum length of the shift registers. Additional m zero bits are required at the tail of the information sequence to take the convolutional encoder back to the all-zero state. It is required that the encoder start and end at the all-zero state. The subscript denotes

the time index while the superscript denotes the bit within a particular input k-bit or output nbit block. The received and estimated sequences $\mathbf{r}$ and $\mathbf{y}$ can be described similarly as [4]

$$\mathbf{r} = (r_0^1, r_0^2, ..., r_0^n, r_1^1, ..., r_1^n, r_{L+m-1}^1, ..., r_{L+m-1}^n) \tag{3.5}$$

and

$$\mathbf{y} = (y_0^1, y_0^2, ..., y_1^n, y_1^1, ..., y_1^n, y_{L+m-1}^1, ..., y_{L+m-1}^n) \tag{3.6}$$

For Maximum Likelihood (ML) decoding, the Viterbi algorithm selects y to maximize p($\mathbf{r}|\mathbf{y}$). The channel is assumed to be memoryless, and thus the noise process affecting a received bit is independent from the noise process affecting all of the other received bits [Wic95]. From probability theory, the probability of joint, independent events is equivalent to the product of the probabilities of the individual events. [4] Thus,

$$p(\mathbf{r}/\mathbf{y}) = \prod_{i=0}^{L+m-1}[p(r_i^1/y_i^1)p(r_i^2/y_i^2)....p(r_i^n/y_i^n)] \quad [15] \tag{3.7}$$

$$= \prod_{i=0}^{L+m-1}[\prod_{j=1}^{n}[p(r_i^{(j)}/y_i^{(j)})]] \quad [15] \tag{3.8}$$

This equation is called the likelihood function of y given that r is received. The estimate that maximizes p($\mathbf{r}|\mathbf{y}$) also maximizes log p($\mathbf{r}|\mathbf{y}$) because logarithms are monotonically increasing functions. Thus, a log likelihood function can be defined as

$$log(p(\mathbf{r}/\mathbf{y})) = \sum_{i=0}^{L+m-1}[\sum_{j=1}^{n} log(p(r_i^{(j)}/y_i^{(j)}))] \quad [15] \tag{3.9}$$

For an easier manipulation of the summations over the log function, a bit metric is defined. The bit metric is defined as

$$M(r_i^{(j)}/y_i^{(j)}) = a[log[p(r_i^{(j)}/y_i^{(j)})] + b] \quad [15] \tag{3.10}$$

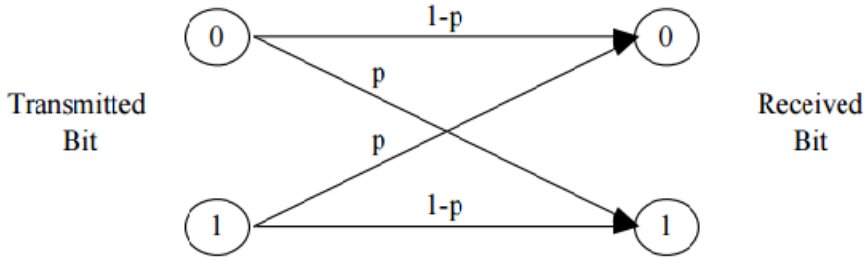

Figure 32. Binary Symmetric channel with p as crossover probability

where a and b are chosen such that the bit metric is a small positive integer.The values a and b are defined for binary symmetric channel (BSC) or hard-decision decoding. Figure 32 shows a BSC.

For BSC, a and b can be chosen in two distinct ways. For the conventional way, they can be chosen as

$$a = \frac{1}{log(p) - log(1-p)} \quad [15] \tag{3.11}$$

and

$$b = -log(1-p) \quad [15] \tag{3.12}$$

The resulting bit metric is then

$$M(r_i^{(j)}/y_i^{(j)}) = \frac{1}{[log(p) - log(1-p)}[log(p(r_i^{(j)}/y_i^{(j)})) - log(1-p)] \tag{3.13}$$

From the BSC model, it is clear that $p(r_i^{(j)}/y_i^{(j)})$ can only take on values p and $1-p$. Table shows the resulting bit metric:

$M(r_i^{(j)}/y_i^{(j)})$	$r_i^{(j)} = 0$	$r_i^{(j)} = 1$
Decoded Bit $y_i^j = 0$	0	1
Decoded Bit $y_i^j = 1$	1	0

Table 1. Conventional Bit Metric Values

This bit metric shows the cost of receiving and decoding bits. For example, if the decoded bit $y_i^{(j)} = 0$ and the received bit $r_i^{(j)} = 1$, then the cost $M(r_i^{(j)}/y_i^{(j)}) = 1$. As

it can be seen, this is related to the Hamming distance and is known as the Hamming distance metric. Thus, the Viterbi algorithm chooses the code sequence **y** through the trellis that has the smallest cost/Hamming distance relative to the received sequence **r**. [4]

Alternatively, a and b can be chosen as

$$a = \frac{1}{log(1-p) - log(p)} \quad [15] \tag{3.14}$$

and

$$b = -log(p) \quad [15] \tag{3.15}$$

The resulting alternative bit metric is then

$$M(r_i^{(j)}/y_i^{(j)}) = \frac{1}{[log(1-p) - log(p)]}[log(p(r_i^{(j)}/y_i^{(j)})) - log(p)] \tag{3.16}$$

Table below shows the resulting alternative bit metric:

$M(r_i^{(j)}/y_i^{(j)})$	$r_i^{(j)} = 0$	$r_i^{(j)} = 1$
Decoded Bit $y_i^j = 0$	1	0
Decoded Bit $y_i^j = 1$	0	1

Table 2. Alternative Bit Metric Values

For this case, the Viterbi algorithm chooses the code sequence **y** through the trellis that has the largest cost/Hamming distance relative to the received sequence **r**. Furthermore, for an arbitrary channel (not necessarily BSC), the values a and b are found on a trial-and error basis to obtain an acceptable bit metric. From the bit metric, a path metric is defined. The path metric is defined as [4]

$$M(\mathbf{r}/\mathbf{y}) = \sum_{i=0}^{L+m-1} \sum_{j=1}^{n} M(r_i^{(j)}/y_i^{(j)}) \; quad[15] \tag{3.17}$$

and indicates the total cost of estimating the received bit sequence $\mathbf{r}$ with the decoded bit sequence $\mathbf{y}$ in the trellis diagram. Furthermore, the k^{th} branch metric is defined as

$$M(\mathbf{r_k}/\mathbf{y_k}) = \sum_{j=1}^{n} [M(r_k^{(j)}/y_k^{(j)})] \quad [15] \tag{3.18}$$

and the k^{th} partial path metric is defined as

$$M^k(\mathbf{r}/\mathbf{y}) = \sum_{i=0}^{k} M(\mathbf{r_i}/\mathbf{y_i}) = \sum_{i=0}^{k} \sum_{j=1}^{n} M(r_i^{(j)}/y_i^{(j)}) \quad [15] \tag{3.19}$$

The k_{th} branch metric indicates the cost of choosing a branch from the trellis diagram. The k_{th} partial path metric indicates the cost of choosing a partially decoded bit sequence $\mathbf{y}$ up to time index k.

The Viterbi algorithm utilizes the trellis diagram to compute the path metrics. Each state (node) in the trellis diagram is assigned a value, the partial path metric. The partial path metric is determined from state $s = 0$ at time $t = 0$ to a particular state $s = k$ at time $t \geq 0$. At each state, the "best" partial path metric is chosen from the paths terminated at that state. The "best" partial path metric may be either the larger or smaller metric, depending whether a and b are chosen conventionally or alternatively. The selected metric represents the survivor path and the remaining metrics represent the non survivor paths. The survivor paths are stored while the non survivor paths are discarded in the trellis diagram. The Viterbi algorithm selects the single survivor path left at the end of the process as the ML path. Trace-back of the ML path on the trellis diagram would then provide the ML decoded sequence. [4]
The hard-decision Viterbi algorithm (HDVA) can be implemented as follows [15] [12] :
$S_{k,t}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_k, t)$.
Step 1:
(a) Initialize time $t = 0$.

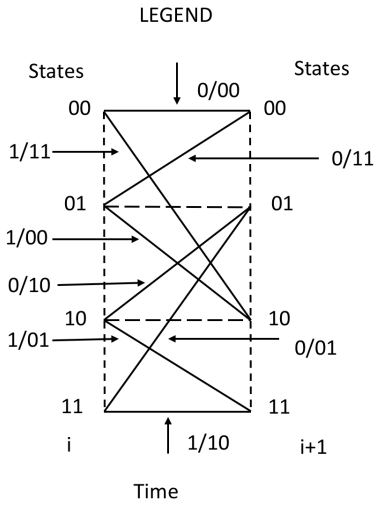

Figure 33. The state transition diagram (trellis legend) of the example convolutional encoder

(b) Initialize $V(S_{0,0}) = 0$ and all other $V(S_{k,t}) = +\infty$

Step 2:

(a) Set time $t = t + 1$.

(b) Compute the partial path metrics for all paths going to state S_k at time t.

First, find the t^{th} branch metric

$$\mathrm{M}^k(\mathbf{r_t}/\mathbf{y_t}) = \textstyle\sum_{\mathbf{j=1}}^{\mathbf{n}} M(r_t^{(j)}/y_t^{(j)}) \tag{3.20}$$

This is calculated from the Hamming distance $\sum_{j=1}^{n} |r_t^j - y_t^j|$. Second, compute the t_{th} partial path metric

$$\mathrm{M}_t(\mathbf{r}/\mathbf{y}) = \textstyle\sum_{i=0}^{t} M(\mathbf{r_i}/\mathbf{y_i}) \tag{3.21}$$

This is calculated from

$$\mathrm{V}(\mathrm{S}_{k,t-1}) + M(\mathbf{r_t}/\mathbf{y_t}) \tag{3.22}$$

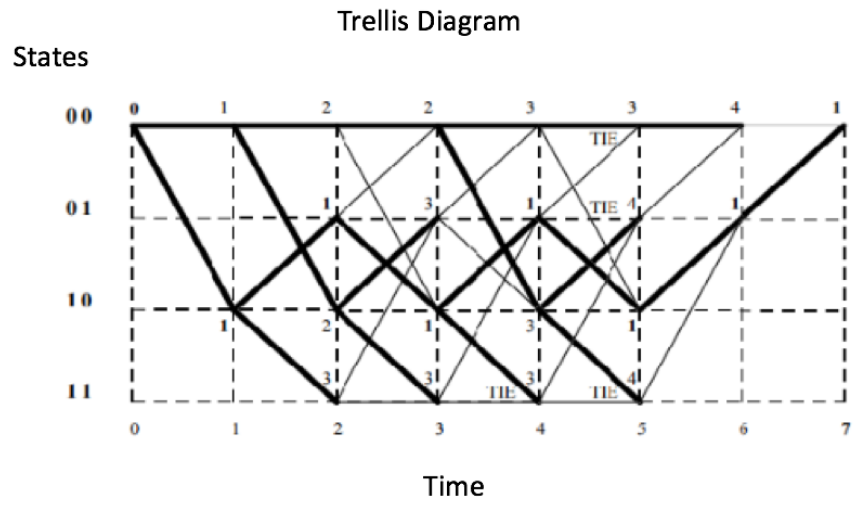

Figure 34. HDVA decoding of the example

Step 3:

(a) Set $V(S_{k,t})$ to the best partial path metric going to state S_k at time t. Conventionally, the best partial path metric is the partial path metric with the smallest value.

(b) If there is a tie for the best partial path metric, then any one of the tied partial path metric may be chosen.

Step 4:

Store the best partial path metric and its associated survivor bit and state paths.

Step 5:

If $t < L + m - 1$, return to Step 2.

The result of the Viterbi algorithm is a unique trellis path that corresponds to the ML codeword. [4]

A simple HDVA decoding example is shown below. The convolutional encoder used is shown in Figure 28. The input sequence is $\mathbf{x}$=1010100, where the last two bits are used to return the encoder to the all-zero state. The coded sequence is $\mathbf{c} =$

$[11, 10, 00, 10, 00, 10, 11]$. However, the received sequence $\mathbf{r} = 10, 10, 00, 10, 00, 10, 11$ has a bit error (underlined). Figure 33 shows the state transition diagram (trellis legend) of the example convolutional encoder.

The state transition diagram shows the estimated information and coded bits along the branches (needed for the decoding process). HDVA decoding chooses the ML path through the trellis as shown in Figure 34. The chosen partial path (accumulated) metric for this example is the smallest Hamming distance and are shown in the figure for every node.

The bold partial path metrics correspond to the ML path. Survivor paths are represented by bold solid lines and competing paths are represented by simple solid lines.For metric "ties", the first branch is always chosen. [4] From the trellis diagram in Figure 2.13, the estimated code sequence is $\mathbf{y} = 11, 10, 00, 10, 00, 10, 11$ which is the code sequence $\mathbf{c}$. Utilizing the state transition diagram in Figure 2.12, the estimated information sequence is $\mathbf{x}' = [1010100]$

3.2.2.2 Soft Decision Viterbi Algorithm

There are two general methods of implementing a soft-decision Viterbi algorithm. The first method (Method 1) uses Euclidean distance metric instead of Hamming distance metric. The received bits used in the Euclidean distance metric are processed by multi-bit quantization. The second method (Method 2) uses a correlation metric where its received bits used in this metric are also processed by multi-bit quantization. [4] I have used the first method.

Soft-Decision Viterbi Algorithm:

In soft-decision decoding, the receiver does not assign a zero or a one (single-bit quantization) to each received bit but uses multi-bit or infinite-bit quantized values. Ideally, the received sequence r is infinite-bit quantized and is used directly in the soft-decision Viterbi decoder. The soft-decision Viterbi algorithm is similar to its hard-decision algorithm except that squared Euclidean distance is used in the metric instead of Hamming distance. [4]

The soft-decision Viterbi algorithm (SDVA1) can be implemented as follows: $S_{k,t}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

Step 1:

(a) Initialize time $t = 0$.

(b) Initialize $V(S_{0,0}) = 0$ and all other $V(S_{k,t}) = +\infty$

Step 2:

(a) Set time $t = t + 1$.

(b) Compute the partial path metrics for all paths going to state S_k at time t. First, find the t^{th} branch metric $M(\mathbf{r_t}/\mathbf{y_t}) = \sum_{j=1}^{n} M(r_t^{(j)}/y_t^{(j)})$. This is calculated from the squared Euclidean distance $\sum_{j=1}^{n}(r_t^{(j)} - y_t^{(j)})^2$. Second, compute the t^{th} partial path metric $M^t(\mathbf{r}/\mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r_i}/\mathbf{y_i})$. This is calculated from $M^t(\mathbf{r}/\mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r_i}/\mathbf{y_i})$. This is calculated from $V(S_{k,t-1}) + M(\mathbf{r_t}|\mathbf{y_t})$

Step 3:

(a) Set $V(S_{k,t})$ to the "best" partial path metric going to the State S_k at time t. Conventionally the "best" partial path metric is the partial path metric with the smallest value.

(b) If there is a tie for the "best" partial path metric, then any one of the tied partial path metric may be chosen.

Step 4:

Store the “best” partial path metric and its associate survivor bit and the state paths.

Step 5:

If t<L+m-1 then return to Step 2. [4]

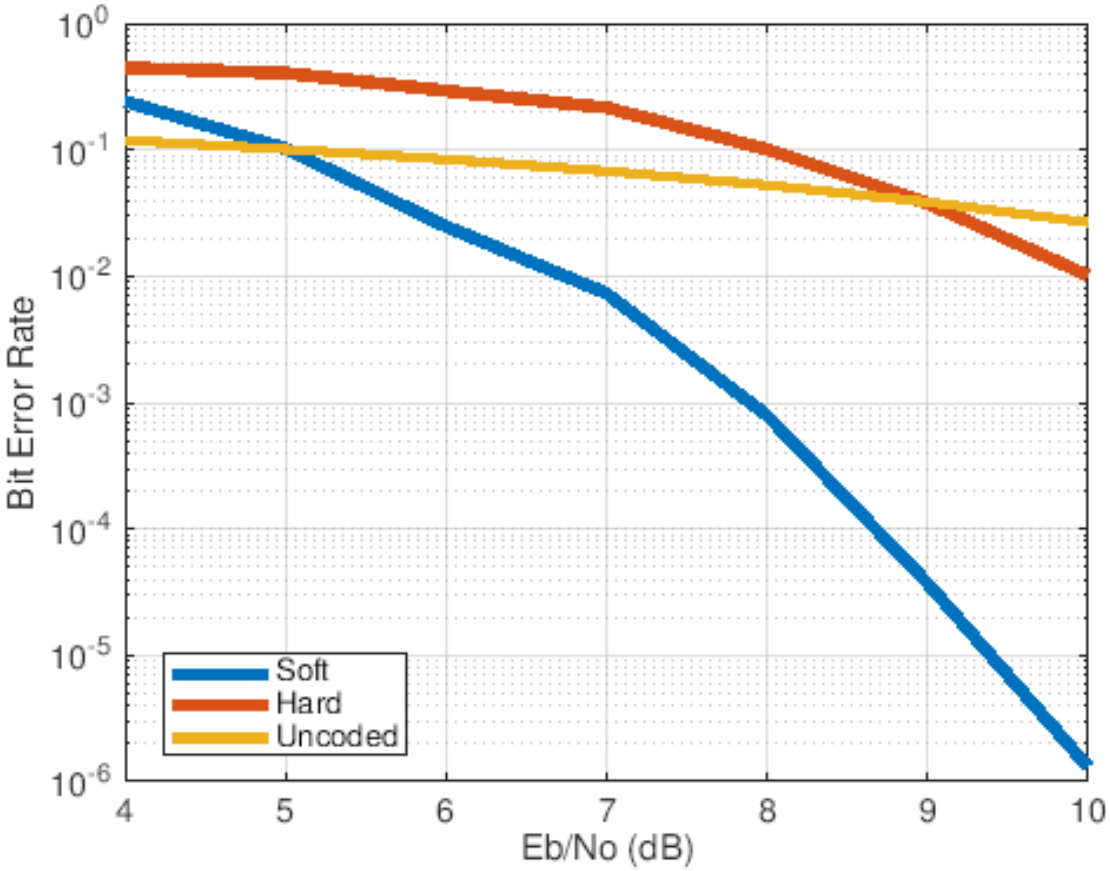

Figure 35. Performance comparison of soft and hard decision viterbi decoder for 64-qam modulation in an AWGN channel for rate 1/2 convolution code.

3.2.3 Performance comparison of Soft and Hard decision Viterbi Algorithm

Here, I have compared the bit error rate (BER) vs signal to noise ratio performance of soft and hard decision viterbi decoder for rate $\frac{1}{2}$ convolution code in additive white gaussian noise (AWGN) channel using 64 QAM modulation. Figure 35 clearly tells that the soft decision decoding produces the best results as it provides coding gain over hard decision decoding.

3.3 Turbo Code Encoder

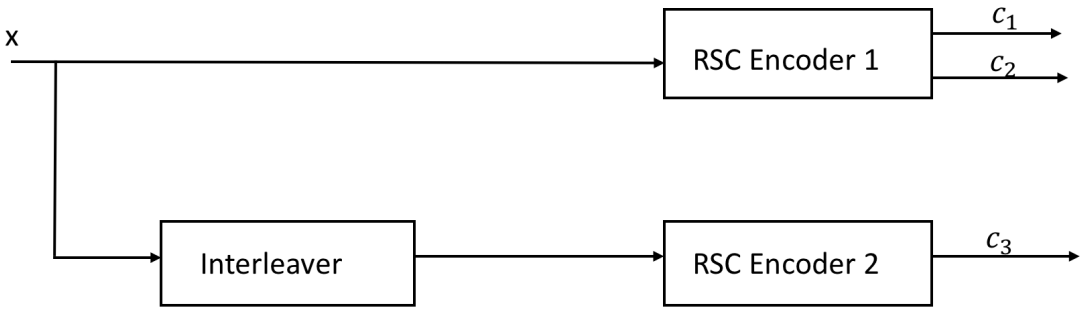

Figure 36. Fundamental turbo code encoder

This section describes the turbo code encoder and its components in detail. The fundamental turbo code encoder is built using two identical recursive systematic convolutional (RSC) codes with parallel concatenation. The turbo code encoder which I have used consists of two identical 8-state recursive systematic convolutional encoders. The first encoder operates directly on the input bit sequence, while the other operates on interleaved input sequences, obtained by interleaving the input bits

over a block length. An RSC encoder is typically $r = \frac{1}{2}$ and is termed a component encoder. The two component encoders are separated by an interleaver. Only one of the systematic outputs from the two component encoders is used, because the systematic output from the other component encoder is just a permuted version of the chosen systematic output. The output from turbo encoder consists of appended termination bits at the end of the encoded data bits. Figure 36 shows the fundamental turbo code encoder. The first RSC encoder outputs the systematic $\mathbf{c_1}$ and recursive convolutional $\mathbf{c_2}$ sequences while the second RSC encoder discards its systematic sequence and only outputs the recursive convolutional $\mathbf{c_3}$ sequence. [4]

3.3.1 Recursive Systematic Convolutional (RSC) Encoder

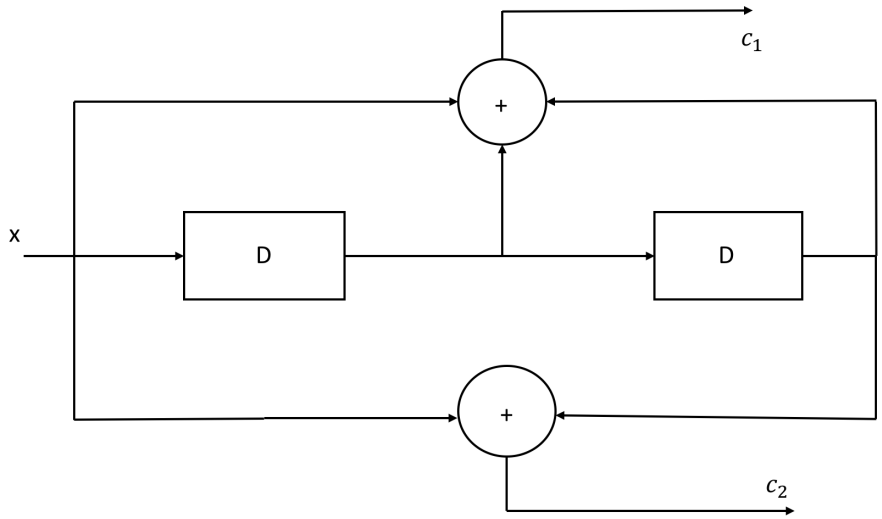

Figure 37. Conventional convolutional encoder with $r = 1/2$ and $K = 3$

The recursive systematic convolutional (RSC) encoder is obtained from the nonrecursive nonsystematic (conventional) convolutional encoder by feeding back one of its

encoded outputs to its input. Figure 37 shows a conventional convolutional encoder. [4]

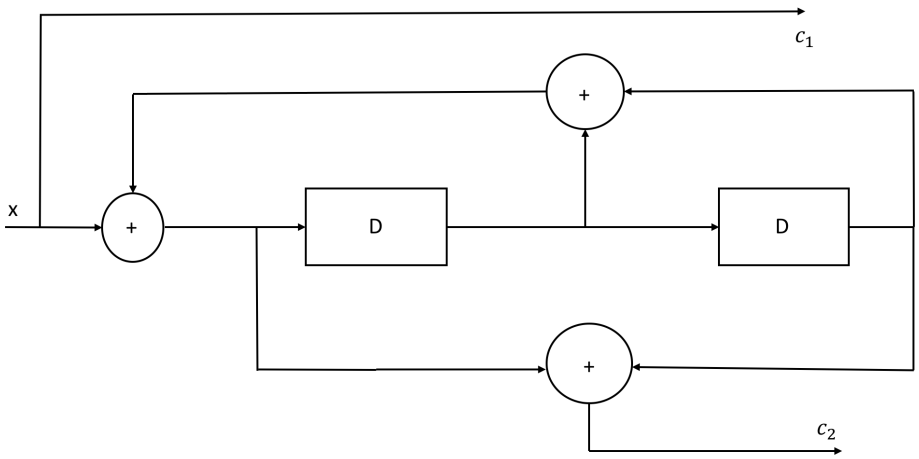

Figure 38. The RSC encoder with $r = 1/2$ and $K = 3$

The conventional convolutional encoder is represented by the generator sequences g1 = [1 1 1] and g2 =[1 0 1] and can be equivalently represented in a more compact form as $G = [g1, g2]$. The RSC encoder of this conventional convolutional encoder is represented as $G = [1, g2, g1]$ where the first output (represented by g1) is fed back to the input. In the above representation, 1 denotes the systematic output, g2 denotes the feed forward output, and g1 is the feedback to the input of the RSC encoder. Figure 38 shows the resulting RSC encoder.

3.3.2 Interleaver

Coding techniques such as convolutional codes are suitable for channels with random errors like binary symmetric channel or AWGN channel. But there are many

channels where errors occur continuously. A burst of errors produces large errors in code words. So, there appears a need of strong correction capability. In order deal with these bursty channels, inter-leaver is introduced. Inter-leaver works by taking an input sequence and permuting it randomly or according to a prescribed method. Inter-leaver proves to be very effective in dealing with bursty channels by permuting the data at the receiver side. There are various types of inter-leavers like block inter-leaver and convolutional inter-leaver. In turbo codes inter-leaver that permutes the data randomly is preferred. [1]

3.4 Soft Input for Turbo Decoder

The input data to a turbo decoder is soft demodulated binary data. Two soft-decision algorithms are available: exact log-likelihood ratio (LLR) and approximate LLR. Exact LLR provides the greatest accuracy but is slower, while approximate LLR is less accurate but more efficient. I have used approximate LLR input to the turbo decoder.

Exact Log Likelihood Ratio Algorithm:

The log-likelihood ratio (LLR) is the logarithm of the ratio of probabilities of a 0 bit being transmitted versus a 1 bit being transmitted for a received signal. The LLR for a bit, b, is defined as:

$$L(b) = log\frac{Pr(b = 0|r = (x, y))}{Pr(b = 1|r = (x, y))} \quad [6] \tag{3.23}$$

where,

r is received signal with coordinates (x, y) and b is transmitted bit (one of the K bits in an M-ary symbol, assuming all M symbols are equally probable) [6]

Approximate Log likelihood Ratio Algorithm:

Approximate LLR is computed by using only the nearest constellation point to the received signal with a 0 (or 1) at that bit position, rather than all the constellation points as done in exact LLR. It is defined in as [6]:

$$L(b) = -\frac{1}{\sigma^2}(\min_{s \in S_o}((x - S_x)^2 + (y - S_y)^2)) - \min_{s \in S_1}((x - S_x)^2 + (y - S_y)^2)) \quad [6] \quad (3.24)$$

where,

b is transmitted bit (one of the K bits in an M-ary symbol, assuming all M symbols are equally probable)

S_o is ideal symbols or constellation points with bit 0, at the given bit position

S_1 is ideal symbols or constellation points with bit 1, at the given bit position

S_x is In-phase coordinate of ideal symbol or constellation point

S_y is Quadrature coordinate of ideal symbol or constellation point

σ^2 is noise variance of baseband signal

3.5 Turbo Decoder

The Turbo Decoder block decodes the input signal using a parallel concatenated decoding scheme. The iterative decoding scheme uses the a posteriori probability (APP) decoder as the constituent decoder, an interleaver, and a de-interleaver. The two constituent decoders use the same trellis structure and decoding algorithm.

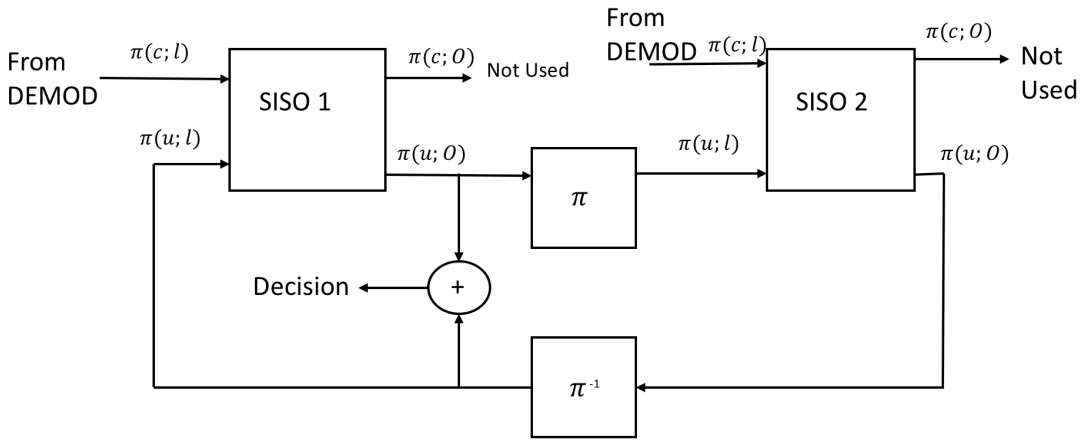

Figure 39. Block diagram of iterative turbo decoding

3.5.1 Iterative Turbo Decoding

The block diagram in Figure 39 illustrates that the APP decoders (labeled as SISO modules in the Figure) output an updated sequence of log-likelihoods of the encoder input bits, $\pi(u;O)$. This sequence is based on the received sequence of log-likelihoods of the channel (coded) bits, $\pi(c;I)$, and code parameters. The decoder block iteratively updates these likelihoods for a fixed number of decoding iterations and then outputs the decision bits. The interleaver (π) that the decoder uses is identical to the one the encoder uses. The de-interleaver (π-1) performs the inverse permutation with respect to the interleaver. The decoder does not assume knowledge of the tail bits and excludes these bits from the iterations. [9]

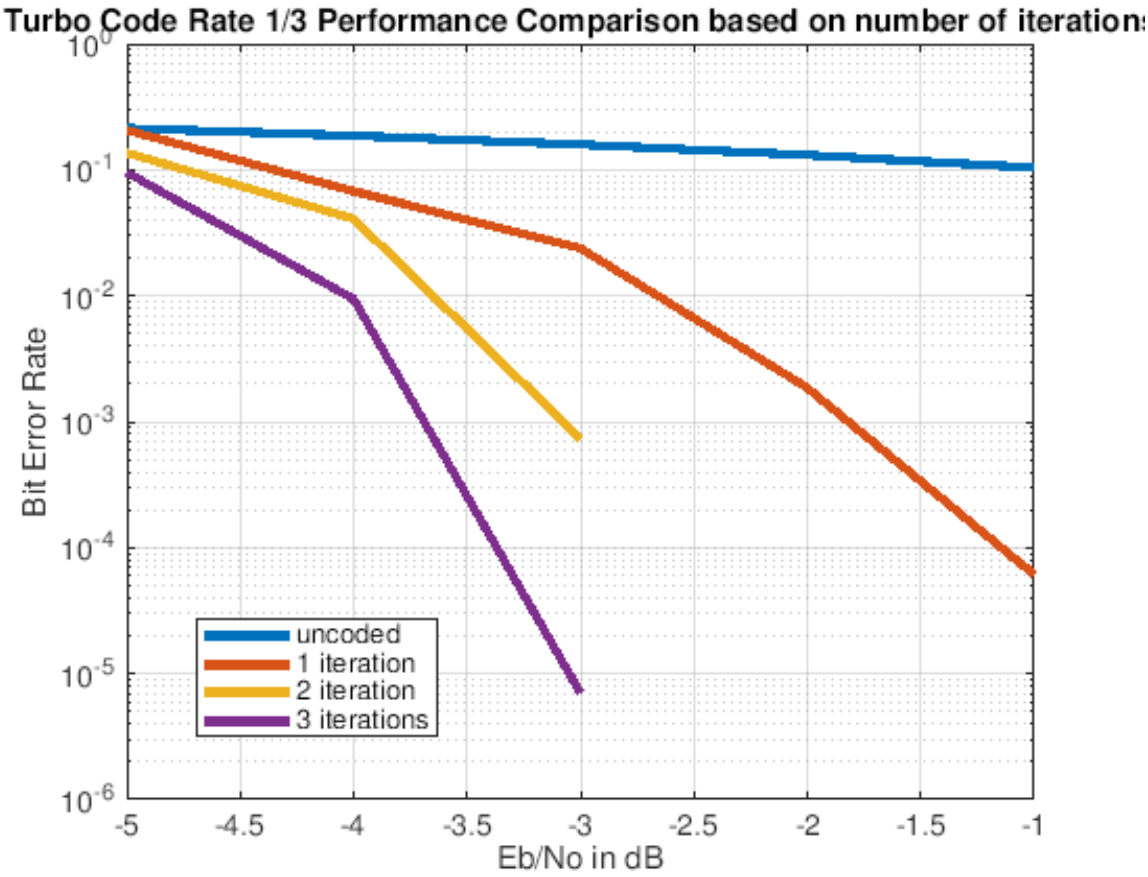

Figure 40. Turbo code performance as a function of number of decoding iterations

3.5.2 Performance comparison of turbo codes based on number of decoding iterations

Figure 40 shows that as the decoder performs more number of iterations, performance improves. However with each subsequent iteration, improvement follows a law of diminishing returns. The corresponding increase in the coding gain with an increase in the number of iterations is not proportionate. It causes more amount of decoding delay and complexity than the coding gain and hence it is not justified.

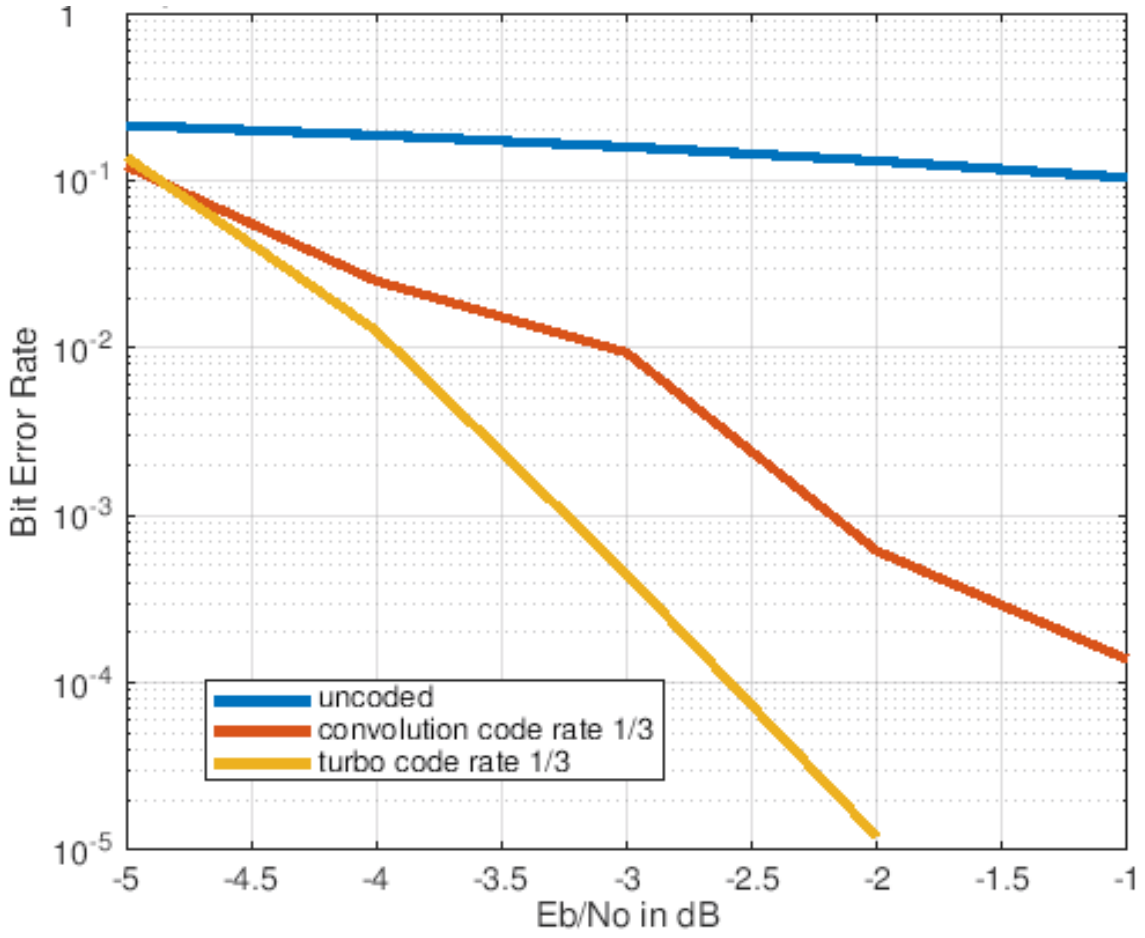

Figure 41. Performance comparison of rate $\frac{1}{3}$ turbo and convolution code for BPSK in AWGN channel

Figure 41 shows that turbo codes perform better than convolution codes. For a bit error rate of 10^{-3}, there is a coding gain of 1dB if we use turbo codes over convolution codes. The parallel structure of the turbo codes and the use of an interleaver provide this coding gain as they help in producing high weight code words.

Chapter 4

INVESTIGATING THE IMPLEMENTATION OF ORTHOGONAL FREQUENCY DIVISON MULTIPLEXING SUBCARRIER DENSITY

4.1 Introduction

Wireless applications demand high data rates. Wireless channels are very unpredictable. For high data rate communications, it is not an easy task to deal with wireless channels. Multi-carrier transmissions can be used for combating the hostility of wireless channels and providing high data rate communications. Orthogonal Frequency Division Multiplexing (OFDM) is a special form of multi-carrier transmission which promises a high user data rate transmission capability at a reasonable complexity and precision.

At high data rates, it is somewhat impossible to recover the transmitted data with a simple receiver because of the significant channel distortion. A very complex receiver structure is needed which will use computationally expensive equalization and channel estimation algorithms to correctly estimate the channel. OFDM can drastically simplify the equalization problem by turning the frequency-selective channel into a flat channel. A simple one-tap equalizer is needed to estimate the channel and recover the data. [5]

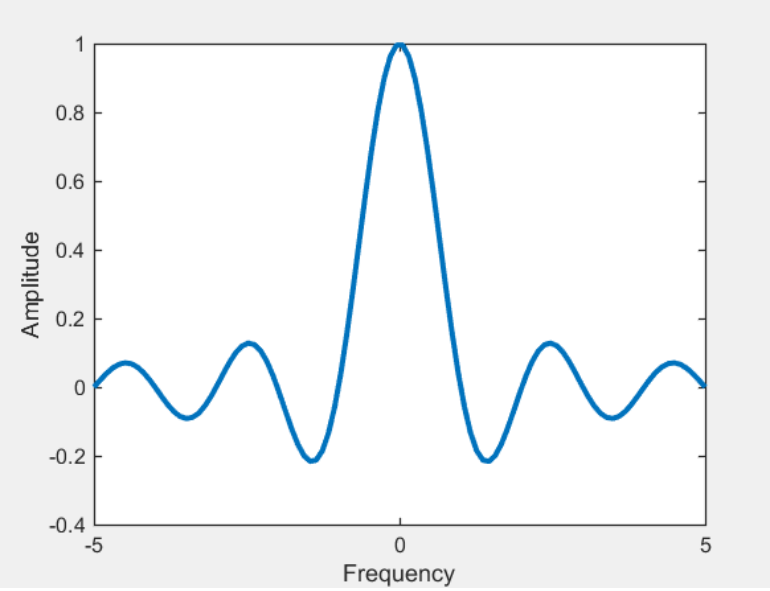

Figure 42. A single carrier transmission is shown here. One single carrier occupies the entire transmission bandwidth B. Every T seconds one symbol is sent over it

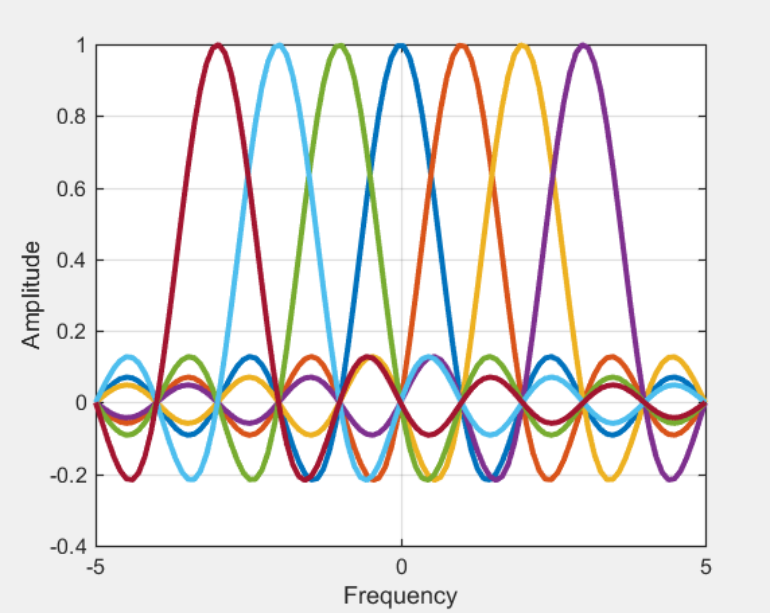

Figure 43. An OFDM symbol is shown here. There are 7 subcarriers in this OFDM symbol occupying the bandwidth B. Each subcarrier transmits one symbol in parallel. In OFDM, these subcarriers are overlapping orthogonally. It is a more spectrally efficient technique than single carrier.

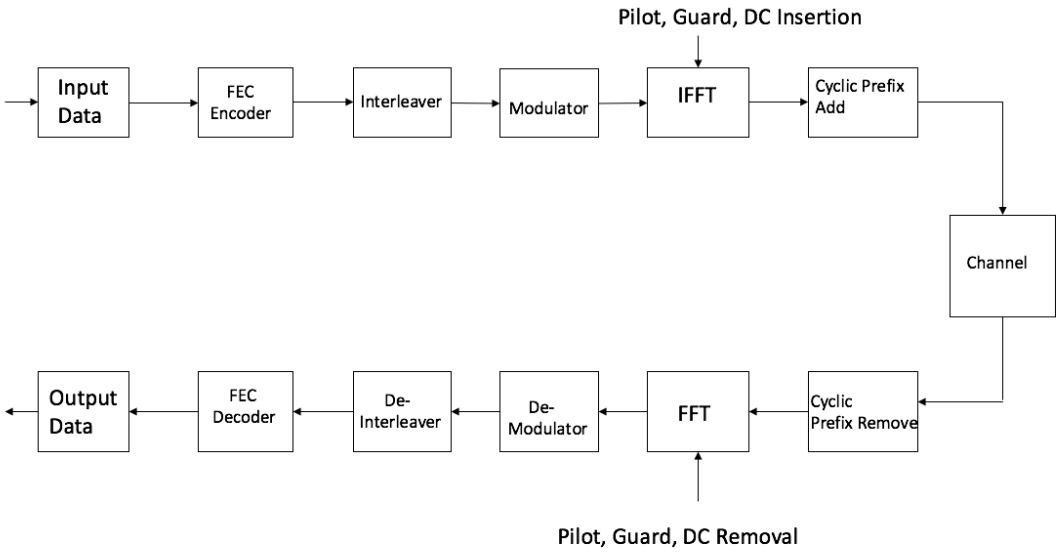

Figure 44. Block Diagram of an OFDM Transmitter and Receiver

4.2 Block Diagram of an OFDM System

Figure. 46 shows the block diagram of a basic OFDM system. At the transmitter side as we can see that the bits are first encoded using the desired error correction coding scheme and then converted to parallel streams. The parallel streams are then modulated with the modulation scheme required and then pilots are added which are used for channel estimation. After pilot tone insertion, the frequency domain signal is converted into time domain using Inverse Fast Fourier transform (IFFT). Then the cyclic prefix (CP) is added to the OFDM symbol which helps in mitigating the inter symbol interference. After that the parallel streams are again converted into serial and then the baseband signal is converted into pass band and upsampled and transmitted. At receiver side, the received data is down sampled and then the same steps are done in the opposite order to recover the transmitted data.

4.3 OFDM System Requirements

OFDM systems depend on four system requirements:

- Available bandwidth: Bandwidth is the scarce resource. Hence the main deciding factor in the system design should be the available bandwidth for operation. The amount of bandwidth play a significant role in determining number of sub carriers, because with a large bandwidth, we can easily fit in a large number of sub carriers with reasonable guard space. [5]
- Required bit rate: The overall system should be able to support the data rate required by the users. For example, to support broadband wireless multimedia communication, the system should operate at more than 10 Mega bits per symbol at least. [5]
- Tolerable delay spread: Tolerable delay spread depends on the user environment. Measurements show that indoor environment experiences maximum delay spread of few hundreds of nanoseconds at most, whereas outdoor environment can experience up to 10 s. So the length of Cyclic Prefix should be determined according to the tolerable delay spread. [5]
- Doppler values: Users on a high-speed vehicle will experience higher Doppler shift, whereas pedestrians will experience smaller Doppler shift. These considerations must be taken into account. [5]

4.4 Work Done and Motivation

In this part of my thesis work, I have worked on wideband OFDM or wider Fast Fourier Transform (FFT) for an OFDM symbol. As explained above in the section

for requirements of an OFDM System, the bandwidth is scarce resource.

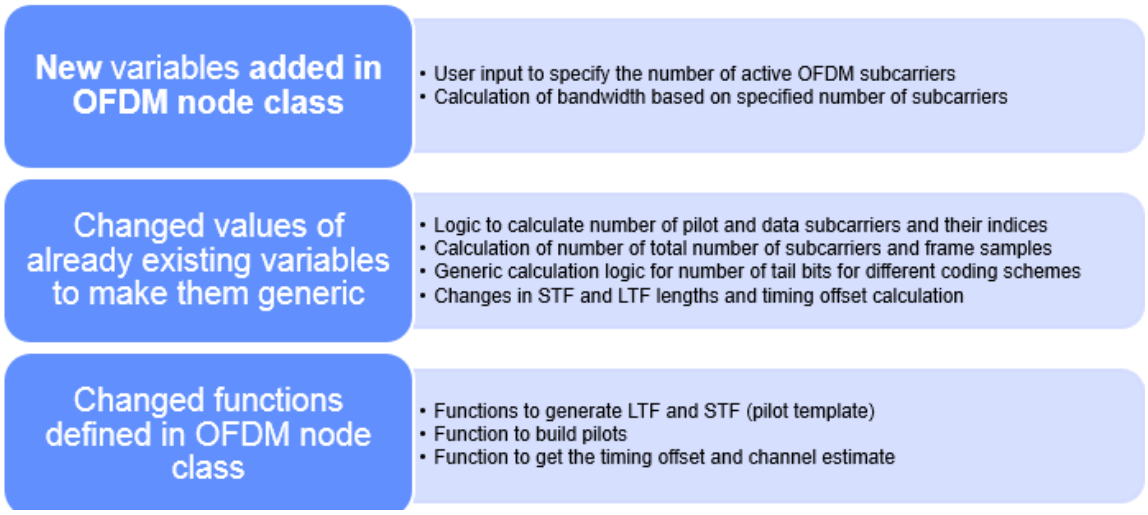

Figure 45. Different radios are implemented in the form of classes in WiscaComm. In order to support programmable subcarrier density, a lot of parameters had to be added and changed in OFDM node class. The figure depicts the changes done in OFDM node class.

So the number of sub carriers should be chosen according to the available bandwidth. Earlier, the OFDM symbol was designed according to the 802.11 standard specifications. In order to simulate it for outdoor environments which have high delay spread and high data rate applications, we required a generic OFDM symbol structure with programmable number of subcarriers. We wanted to simulate OFDM for different scenarios and thus varying FFT lengths were required. Increasing number of subcarriers reduce the datarate via each subcarrier, which ensures that the relative amount of dispersion in time caused by multipath delay is decreased. But when there are large numbers of subcarriers, the synchronization at the receiver side will be extremely difficult. Hence the optimum number of subcarriers can be increased to 4096 for which the performance is reasonable. Also another factor which should be taken in account is the subcarrier spacing. Subcarrier spacing must be kept at a level so

that synchronization is achievable. Subcarrier spacing largely depends on available bandwidth and the required number of subchannels.

4.5 Simulation Results

Below is a simulation output for the Peer to Peer mode in a sub urban setting. The range is 1000 meters. Based on this configuration, the PDK suggested OFDM scheme with Binary Phase Shift Key (BPSK) modulation and rate $\frac{1}{3}$ convolution code. Figure 46 shows the geographical location of the ofdm nodes and the packets being transferred between two nodes.

- Node Location and Packets being Transferred

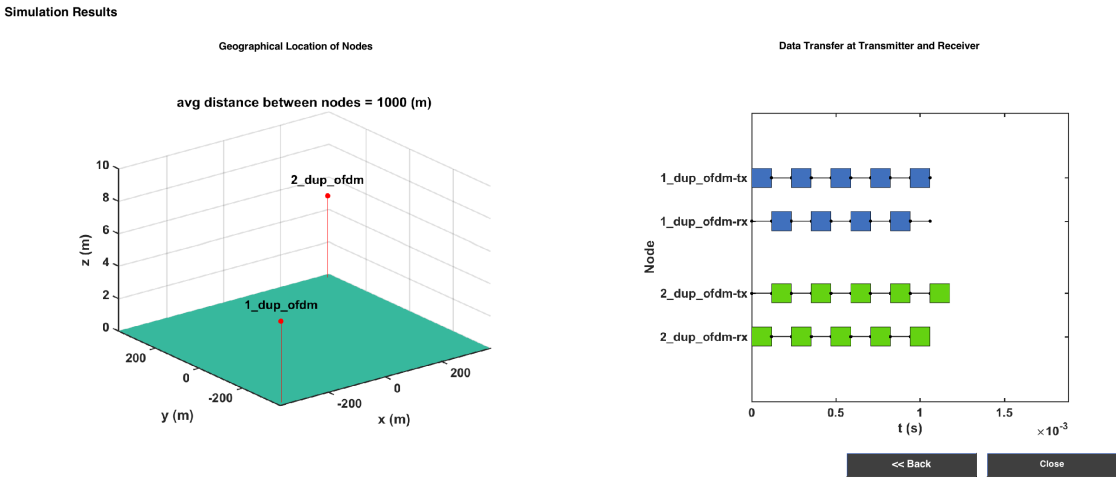

Figure 46. The left figure shows the geographical location of OFDM nodes created. The figure on right shows the packets being transmitted between the OFDM nodes with time.

- Transmit Spectrum

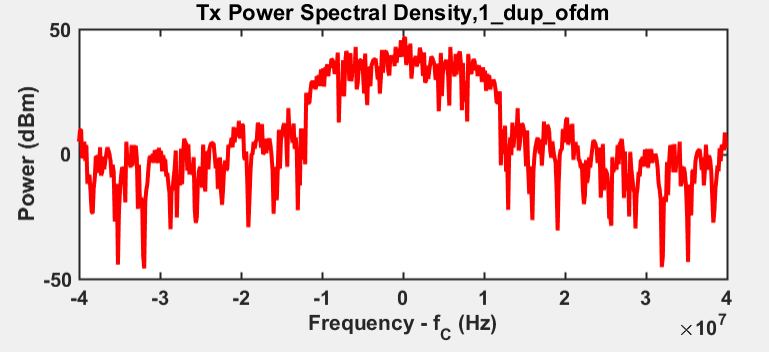

Figure 47. WiscaComm simulation output showing the OFDM transmit spectrum for a 20 MHz channel. The simulation environment consists of P2P topology in sub-urban setting for a range of 1000 meters. The PDK suggested OFDM with BPSK modulation and rate 1/3 convolution code.

- Received Spectrum

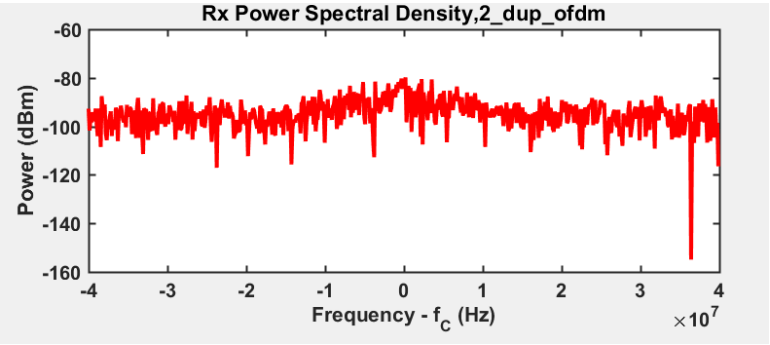

Figure 48. Output from a WiscaComm simulation which shows the received spectrum for a 20MHz channel. The simulation environment consists of P2P topology in sub-urban setting for a range of 1000 meters. The PDK suggested OFDM with BPSK modulation and rate 1/3 convolution code.

- Bit Error Plot at receiver Side

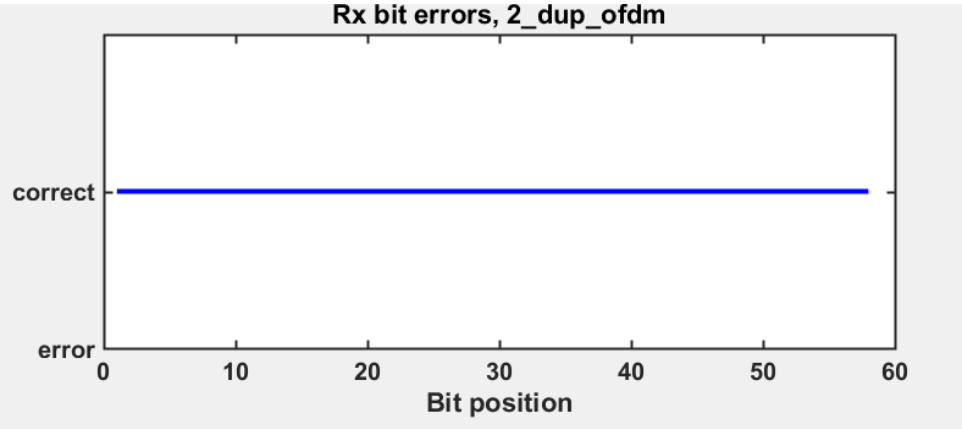

Figure 49. Output from a WiscaComm simulation which shows the bit error rate at receiver end. The simulation environment consists of P2P topology in sub-urban setting for a range of 1000 meters. The PDK suggested OFDM with BPSK modulation and rate 1/3 convolution code. In this case, all the transmitted bits are received correctly and without any error.

Chapter 5

FUTURE WORK

WiscaComm currently doesn't support Multiple Input Multiple Output (MIMO) technique. It can be used to provide increased link capacity and spectral efficiency combined with improved link reliability. Wireless medium suffers from unstable channel conditions e.g. signal fading due to distance, frame collision from simultaneous transmissions, and interference from other sources. Rate adaptation is the determination of the optimal data transmission rate most appropriate for current wireless channel conditions. So, there is a need for rate adaptation for the radios supported by WiscaComm. Equalization and signal acquisition can also be implemented for the radios. A high transmit power may lead to interference in adjacent channels and it also affects the battery life of the device especially its a very important parameter for hand held devices like mobile phones. If we are transmitting with too much power then the battery will drain faster. So there is a need of power control for radios in WiscaComm. We can also implement LDPC Codes in WiscaComm. Also, there is a need for unit and system level testing to check the product robustness.

REFERENCES

[1] Nabeel Arshad and Abdul Basit. "Implementation and analysis of convolutional codes using MATLAB". In: *International journal of multidisciplinary sciences and engineering* 3.8 (2012), pp. 9–12.

[2] Hari Balakrishnan and George Verghese. *Introduction to EECS II: Digital Communication Systems*. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-02-introduction-to-eecs-ii-digital-communication-systems-fall-2012/readings/MIT6_02F12_chap07.pdf.

[3] Vamsi Reddy Chagari. *Development and implementation of physical layer kernels for wireless communication protocols*. Arizona State University, 2016.

[4] Fu-hua Huang. *Evaluation of soft output decoding for turbo codes*. Virginia Tech, 1997.

[5] Nicola Marchetti et al. "OFDM: Principles and challenges". In: *New directions in wireless communications research* (2009), pp. 29–62.

[6] Mathworks. *Digital Modulation*. URL: https://www.mathworks.com/help/comm/ug/digital-modulation.html.

[7] Mathworks. *Parallel Concatenated Convolutional Coding: Turbo Codes*. URL: https://www.mathworks.com/help/comm/examples/parallel-concatenated-convolutional-coding-turbo-codes.html.

[8] Mathworks. *Syndtable*. URL: https://www.mathworks.com/help/comm/ref/syndtable.html.

[9] Mathworks. *Turbo Decoder*. URL: https://www.mathworks.com/help/comm/ref/turbodecoder.html?s_tid=doc_ta.

[10] John Pardini. *Software Defined Radios*. URL: http://holyfamily.lucidiweb.com/isym540/week2/ExecutiveBriefing2textjohnpardini.pdf.

[11] J. G. Proakis. *Digital Communications, McGraw-Hill*. 3rd ed. 1995.

[12] T. S. Rappaport. *Wireless Communications Principles and Practice, New Jersey, Prentice-Hall,* 1996.

[13] Steven Roman. *Coding and Information Theory, GTM, 134, Springer-Verlag, ISBN 0-387-97812-7*. 1992.

[14] Bluetooth SIG. *Bluetooth Specification version 4.2*. December 2014.

[15] S. B. Wicker. *Error Control Systems for Digital communication and Storage, New Jersey, Prentice-Hall*. 1995.

[16] Wikipedia. *Bluetooth*. URL: https://en.wikipedia.org/wiki/Bluetooth.