Implementation of Tail-Biting Convolutional Codes

by

Deepthi Ancha

Problem Report submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Daryl S. Reynolds, Ph.D.
Natalia A. Schmid, Ph.D.
Matthew C. Valenti, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2008

Keywords: convolutional codes, tailbiting , circular decoding

Abstract

Implementation of Tail-Biting Convolutional Codes

by

Deepthi Ancha
Master of Science in Electrical Engineering

West Virginia University

Matthew C. Valenti, Ph.D., Chair

Error control coding plays an important role in present digital communication systems. Many types of error control codes exist to overcome the errors induced by noise in the transmitting channel. This report emphasizes convolutional codes in general, and the codes described in the IEEE 802.16 (WiMAX) standard in particular. In conventional convolutional codes, a rate loss occurs due to the transmission of extra bits through the channel in order to start and end in the same state. For such systems, bandwidth and transmission time are not efficiently used. In order to overcome the disadvantages of convolutional codes tail-biting convolutional coding is introduced. In a tail-biting convolutional code, the starting and ending states of the encoder are set to be identical, thereby alleviating the need for a tail.

The objective of this report is to implement and test simulation software capable of encoding and decoding tail-biting convolutional codes. The encoder is designed to assure that the starting and ending states of the encoder are the same. The decoder is a modified version of the classic Viterbi algorithm, but modifications proposed in are applied to handle the unknown, but identical, starting and stopping states. The software was developed within the Coded Modulation Library (CML), an open-source library of error control coding functions written by students and faculty of WVU. Existing functions for convolutional encoding and Viterbi decoding were modified to handle tail-biting codes. An extensive set of simulations were run and used to generate bit error rate (BER) curves.

Acknowledgements

I would like to convey my sincere thanks to committee chair and advisor, Dr. Matthew C. Valenti,for his valuable suggestions which helped me lot to keep going through the problem report. I am very grateful to him for his patience and esteemed guidance. I would also like to thank Dr. Daryl Reynolds and Dr. Natalia Schmid for being my committee members and for their time.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The IEEE 802.16 standard or WiMAX (worldwide interoperability for microwave access) is currently in the process of being adopted for broadband wireless access technology. The main objective of WiMAX is to provide high data rate communication. For applications requiring high and stable output, WiMAX provides better performance than the traditional wireless communication standards [1]. WiMAX provides wireless transmissions using various transmission modes like fixed, portable or mobile non line of sight service from base station to subscriber station. Fixed WiMAX provides cost effective point to point and point to multipoint solutions.

The basic protocol layer diagram of OSI and IEEE 802.16 standard is shown in Fig. 1.1. From the seven layers of the OSI (open system interconnection) only two layers are defined in WiMAX. The two layers are physical (PHY) and the data link layer. The physical layer provides the physical connection required to send and receive data between two communicating sources. On the other hand, the data link layer encodes and decodes the data packets. It provides the TCP/IP protocol knowledge of the reliability of the underlying channel and handles the errors occurred in the physical layer [2]. The data link layer is divided into two sublayers called logical link control layer and media access (MAC) layer. The MAC layer comprises of three sublayers: The convergence sublayer (CS), the common part sublayer (CPS), and the security sublayer. Frame synchronization, flow control, error checking are

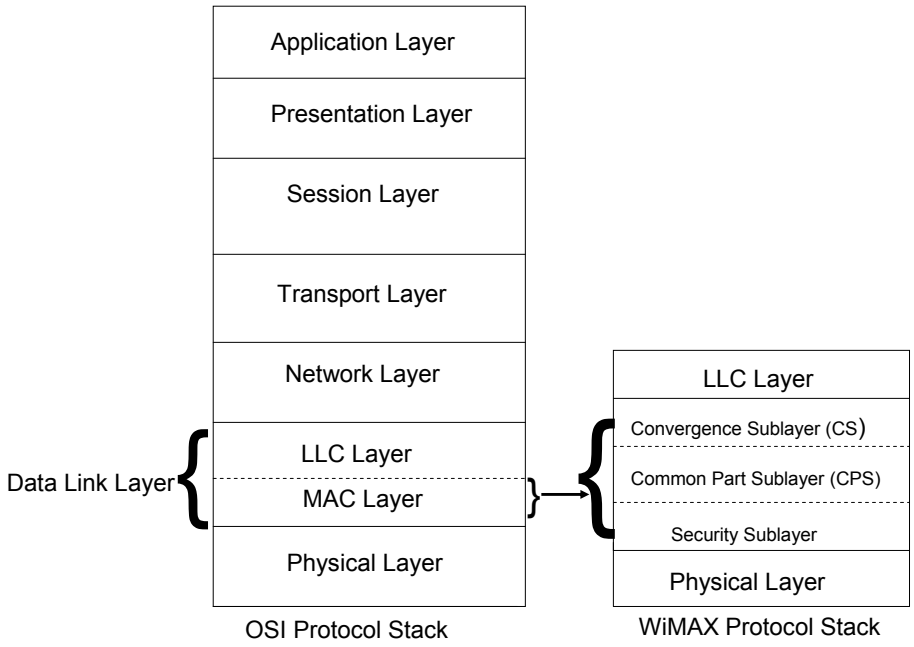

Figure 1.1: The layers of the OSI and WiMAX protocol stacks

done by the logical link control layer.

The MAC layer establishes and maintains the connection required to receive and transmit data. It controls the multiple accessing, scheduling of the data [3]. A basic WiMAX model consists of a transmitter, receiver and channel. The transmitter performs channel coding and modulation, while the receiver performs demodulation and decoding. The channel is generally AWGN, but may also be corrupted by fading and interference. Channel coding is important because it provides the potential for error free transmission across the channel. In order to provide reliable transmission between two communicating sources physical layer uses forward error correction code (FEC) [4]. An alternative to FEC is automatic repeat request (ARQ), which allows erroneous packets to be retransmitted. ARQ is effective in channels that are relatively noise-free, but does not work well in the presence of severe noise because too many retransmissions are required. To overcome this problem and allow ARQ to work in noisy wireless channels, the physical layer could use *hybrid ARQ* which is a combination

of FEC and ARQ.

1.2 Codes in 802.16 Standard

The four types of codes in the IEEE 802.16 standard are

1. Tail-biting Convolutional Code
2. Block Turbo Code
3. Convolutional Turbo Code
4. Low Density Parity Check code

1.2.1 Tail-biting Convolutional code

A convolutional code is generated by a finite-state machine which has states that depend on the current and past inputs. Normally, the encoder starts and ends in the all-zeros state. A *tail* of zeros can be used to bring the encoder back to the all-zeros state, and the decoder is more effective when it knows the starting and ending states. In zero tailing method the encoder's states are assumed to start in all zero's state. Once all the input information sequence is passed through the encoder, it is not in all-zeros state. In order to get back the encoder into all zeros state, zeros equal to m i.e number of memory elements are fed into the encoder. This results in a fractional rate loss. Tail-biting convolutional codes overcome the rate loss caused by the encoding of convolutional codes by zero tailing. With a tail-biting convolutional code, the memory elements are initialized by the last m input bits of the input sequence in order to be sure that encoder ends in the same state.

1.2.2 Block Turbo Code

A block turbo code is the product of two simple codes. In the WiMAX standard, the constituent codes are either binary extended Hamming codes or single parity check codes. k_x information bits of the row are encoded into n_x by the (n_x, k_x) block code specified in the standard. Similarly k_y information bits in the column are encoded into n_y by the (n_y, k_y)

specified block code. The total number of information bits is given as $k = k_x \times k_y$, the block size of the code is $n = n_x \times n_y$ and the rate is $R = R_x \times R_y$ where $R_x = k_x/n_x$ and $R_y = k_y/n_y$. Transmission of the block codes over the channel occurs in linear fashion, where the first row bits from left to right are transmitted first followed by the second row and so on. The block turbo codes can be resized by removing symbols to match a required packet size. The advantage of block turbo codes is that it provides maximum information transfer rate over a noisy channel [3].

1.2.3 Convolutional Turbo Code

Convolutional turbo code is another type of turbo code which uses convolutional methods. It uses a pair of duo-binary circular recursive systematic convolutional codes. In the encoding process the first encoder is initialized with zero and the information sequence is fed in normal order to determine the state of the encoder. A tail-biting encoder is used, so after the final encoder state is determined, the initial encoder state is initialized. After initialization the sequence is encoded. Let the resulting output of the first encoder be $\mathbf{c}_1$. The second encoder operates in a similar manner, though its input is an interleaved version of the information. The output from the second encoder is $\mathbf{c}_2$. The final output is the multiplexed sequence of $\mathbf{c}_1$ and $\mathbf{c}_2$. Puncturing of CTC is done by selectively deleting the parity check bits [3].

1.2.4 Low Density Parity Check Codes

LDPC codes are a class of linear block codes. LDPC codes can be represented in matrix form and graphical form. For a $n \times m$ dimension parity-check matrix to be of low density, the number of ones in rows and columns should be less than n and m respectively. LDPC codes are represented graphically with the help of Tanner graphs.

There are two types of LDPC codes, namely regular and irregular LDPC codes. LDPC codes are regular if the number of ones in every column W_c is the same and the number of ones in every row W_r is constant and $W_r = W_c(n/m)$. If the number of ones is not the same in the rows and columns then it is called irregular LDPC code. The advantage of LDPC code to provide information rate near Shannon limit is true only for large block lengths. However,

large block length results in large parity-check matrices and large generator matrices which increases the complexity [5].

1.3 Objective

The objective of this report is to implement and test simulation software capable of encoding and decoding tail-biting convolutional codes. The encoder is designed to assure that the starting and ending states of the encoder are the same. The decoder is a modified version of the classic Viterbi algorithm, but modifications proposed in [6] are applied to handle the unknown, but identical, starting and stopping states. The software was developed within the Coded Modulation Library (CML), an open-source library of error control coding functions written by students and faculty of WVU. Existing functions for convolutional encoding and Viterbi decoding were modified to handle tail-biting codes. An extensive set of simulations were run and used to generate bit error rate (BER) curves for all of the tail-biting convolutional code configurations listed in the IEEE 802.16 standard.

1.4 Report Structure

Wireless communications plays an important role in the modern world. WiMAX is one of the emerging standard for broadband wireless access. The importance of error control coding, the four codes in the IEEE 802.16 standard, and the objective of the report are mentioned in Chapter 1. Chapter 2 provides a review of basic concepts associated with error control coding and a brief survey of error correcting codes, focusing on block and convolutional codes. In Chapter 3 convolutional encoding, its representations and tail-biting convolutional encoding are presented. Chapter 4 deals with the decoding of convolutional code which include hard decision, soft decision Viterbi algorithm along with circular decoding method for decoding tail-biting convolutional code. Finally, Chapter 5 provides the results obtained from simulation of tail-biting convolutional coding and the conclusion of the report.

Chapter 2

Types of Error Correcting Codes

The demand for efficient and reliable digital transmission has been increasing over the past few years due to the development of high speed, large scale data networks for exchanging, storing and processing of digital information. The main objective of the design to support such communication is to control errors for reliable reproduction of data [7]. When the communication medium has unacceptable high bit error rate and low signal to noise ratio, error coding method is used to provide reliable digital data transmission and storage. In error coding rather than transmitting the information sequence as available it is encoded with some extra bits at the source. The encoding is done by a channel encoder which converts the information sequence into discrete encoded sequence called codeword. The codeword is then transmitted and the channel decoder at the receiver decodes it to retrieve the original information. If errors are introduced while transmission then most likely it would be detected at the receiver, as the codeword would be transformed into an invalid information sequence.

2.1 Error Detection

Error detection is the ability to detect if the received codeword $\mathbf{r}$ is a valid codeword by checking whether $\mathbf{r} \in \mathcal{C}$ where $\mathcal{C}$ is the set containing all the valid codewords that can be produced by an encoder. d_{min} is defined as the minimum Hamming distance between all distinct pairs of codewords. As the closest pair of codewords are d_{min} apart there exists some error pattern of weight d_{min} that transforms the transmitted valid codeword into another

valid codeword making it not possible to detect the error [8].

$$\mathbf{r} \quad = \quad \mathbf{c_i} + \mathbf{e} = \mathbf{c}_j \tag{2.1}$$

Such errors are undetectable errors. But there are no error patterns of weight less than d_{min} that are undetectable. The maximum error detection capability of the code is d_{min}-1. All error patterns with Hamming weight

$$W_h(e) \leq d_{min} - 1 \tag{2.2}$$

are detectable.

2.2 Error Correction

Error correction is the ability to reconstruct the original data without any errors. A Hamming sphere of radius l centered at a vector $\mathbf{v}$ is the set of all binary vectors within distance l from $\mathbf{v}$.

$$S_l(\mathbf{v}) \quad = \quad \{\mathbf{u} \epsilon \mathbf{v}_n : d(\mathbf{u}, \mathbf{v} \leq l)\} \tag{2.3}$$

Let t be the maximum error weight $w_h(e)$ for which errors can be corrected always, $\mathbf{v}_i$ and $\mathbf{v}_j$ be the distinct codewords. t is the largest radius of non overlapping Hamming spheres around all codewords. The maximum error correcting capability is [8]

$$t \quad = \quad \max_{\mathbf{v}_i, \mathbf{v}_j \epsilon \mathcal{C}} \{l : S_l(\mathbf{v}_i) \cap S_l(\mathbf{v}_j) = \phi, \mathbf{v}_i \neq \mathbf{v}_j\} \tag{2.4}$$

The maximum number of error patterns that can be corrected are

$$t \quad = \quad \lfloor \frac{d_{min} - 1}{2} \rfloor \tag{2.5}$$

Detection and correction can be combined. In order for the errors to be corrected they must be detected first. The idea is to use a part of the redundancy to correct small number of errors while using the remaining redundancy bits to detect large number of errors. Let t_c be the maximum number of correctable errors and t_d be the maximum number of detectable errors and

$$t_d \geq t_c. \tag{2.6}$$

Then the condition

$$t_c + t_d < d_{min}. \tag{2.7}$$

should be satisfied by t_c and t_d.

2.3 Types of Error Control Strategies

2.3.1 Automatic Repeat Request

ARQ used when information can be sent in either direction and transmitter acts as receiver and receiver as transmitter. It uses error detection method, acknowledgements, negative acknowledgements and time outs to achieve reliable transmission. Two commonly used ARQ protocols are *stop-and-wait ARQ* and *continuous ARQ*.

2.3.2 Stop and Wait ARQ

With the stop-and-wait protocol the transmitter sends a code word to receiver and waits for the acknowledgement, positive or negative sent by the receiver. If positive acknowledgement (ACK) is received by the transmitter then it means no errors have been detected and the transmitter sends the next codeword. If negative acknowledgment (NACK) is received then the transmitter resends the same codeword until it is received correctly.

2.3.3 Continuous ARQ

With the continuous ARQ protocol, the transmitter sends the code words continuously and receives the acknowledgments continuously. When an NACK is received the transmitter may resend the codeword with error along with the codewords that follow it. This is called go-back-N ARQ. Other type is selective-repeat ARQ where the transmitter resends only those code words with negative acknowledgement. But it requires more logic and buffering [7].

2.3.4 Forward Error Correction

Forward error correction (FEC) is used when the transmission is strictly from transmitter to receiver i.e one directional. In this case the transmitter applies one or more error detecting methods that makes it easy for the receiver to detect where exactly the error occurred and the errors are automatically corrected with the help of the error correcting codes employed. Examples are magnetic tape storage system, deep space communication systems. Also most of the coded systems used today employ some form of FEC even if the channel is not strictly one way [7].

2.4 Types of Forward Error Correcting Codes

Channel coding is used to overcome noise and interference and to reduce the number of bit errors for the protection of digital data being transmitted. Error correction or detection codes designed for the given error control strategy play a vital role in deciding the performance of an error handling system. There are two different types of codes in common use. They are block codes and convolutional codes. Block codes divide the message into separate blocks of same length, encode it and then transmit it. On the other hand, convolutional codes encode entire message sequence as one long codeword which are then transmitted by dividing it into blocks of fixed length.

2.4.1 Block Codes

Block codes can be used either to detect or correct errors. The encoder divides the information sequence into blocks, where each block contains k information bits each. Each block is represented by the binary k-tuple called a message. There are total of 2^k possible different messages. The encoder converts the k bit long messages into n bit codeword where $k < n$. When $n > k$, $n - k$ redundant bits can be added to the k information bits to form n bit codeword. These redundant bits help in overcoming the channel noise. Since there are 2^k possible different messages, corresponding to each message we have 2^k different possible codewords. Block codes is set of 2^k codewords commonly referred as (n, k) block

codes. The code rate is defined as the number of information bits entering the encoder per transmitted code bit i.e. $R = \frac{k}{n}$. The encoder is memoryless as the n codeword output bits depend only on the corresponding k input message bits and hence can be implemented using combinational logic [7]. Various types of block codes exist like Hamming codes, cyclic codes, BCH codes and Reed-Solomon codes.

2.4.2 Convolutional Codes

Block codes have certain disadvantages such as the requirement to receive the entire codeword prior to completion of decoding. Furthermore, standard decoders for block codes work on hard decision but not on soft which is needed to attain the performance bounds by Shannon. To overcome the drawbacks of block codes, convolutional codes were introduced [8]. Convolutional code is a type of error correcting code which accepts blocks of k-bit length codes and produces an encoded sequence of length n. In this case each encoded block n depends not only on the present k bit messages but also on m previous message blocks. Hence the encoder is said to have a memory order m. It is denoted as (n, k, m) convolutional code with n output, k input linear sequential circuit with memory m. In order to achieve low error probability m must be large enough. The ratio $R = \frac{k}{n}$ is called the *code rate* which is a measure of the efficiency of the code. Each output bit depends on the current input and m past inputs. The term *constraint length* is used to denote the total number of bits each output depends on. The constraint length is denoted by K, where $K = m + 1$ when there is one input to the encoder.

Convolutional codes have a simple structure. Convolutional codes are used to improve the performance of radio, mobile phones, satellite links, and deep space communications where bandwidth is not limited. Convolutional encoding is done by convolving the input sequence $\mathbf{u}$ with the generator sequence $\mathbf{g}$ to obtain the output $\mathbf{v}$. The generator sequences may be found by applying an impulse (1 followed by all zeros) to the encoder input.

$$\mathbf{v} = \mathbf{u} * \mathbf{g} \tag{2.8}$$

Several decoding techniques are available for decoding of convolutional codes. First, sequential decoding was proposed as the decoding technique for convolutional codes. Two

types of sequential decoding algorithms are Fano algorithm and Stack algorithm. The sequential decoding was later overcome by new decoding technique called threshold decoding. Threshold decoding was simpler than sequential but also less efficient. Later maximum-likelihood decoding was proposed by Viterbi known as Viterbi decoding technique which was easy to implement with small number of memory elements. Viterbi decoding along with improved sequential decoding lead convolutional codes to be applied to deep space and satellite communications. Viterbi decoding complexity increases exponentially with the memory order m. Codes with longer constraint lengths are usually decoded with sequential decoding as complexity of sequential decoding increases only slightly with the increase of constraint length.

Convolutional codes are most conveniently encoded when the rate is of the form $1/n$. However, many applications require higher rates such as 2/3. Higher rates may be achieved through a process called *puncturing*. Puncturing is done by deleting some of the bits in the encoded output. Deletion of bits is done according to a puncturing matrix. For example consider a rate 1/2 convolutional code. The puncturing and serialization patterns used to obtain different code rates from the basic 1/2 code rate are listed in Table 2.1, where X and Y are the outputs of the encoder. In the table, 1 represents that bit is transmitted and 0 represents that bit is deleted. At the receiver, the decoder inserts dummy bits, setting the log-likelihood ratio to zero by not effecting the decoding metric. The punctured codes have high code rates hence provide high coding gain without requiring a complex decoder.

	Code Rates			
Rate	1/2	2/3	3/4	5/6
d_{free}	10	6	5	4
X_1	1	10	101	10101
Y_1	1	11	110	11010
XY	X_1Y_1	$X_1Y_1Y_2$	$X_1Y_1Y_2X_3$	$X_1Y_1Y_2X_3Y_4X_5$

Table 2.1: Puncturing Patterns[3]

Chapter 3

Encoding of Convolutional Codes

3.1 Convolutional Encoder

A convolutional encoder can be defined as a machine which accepts binary input messages in the form of k-tuples and produces output codewords in the form of n-tuples. The output codewords not only depend on the input at that time but also on the previous inputs, which implies that convolutional encoders have memory. An (n, k, m) convolutional encoder is represented as rate $R = k/n$ where k is number of input streams, n is number of output streams and m is the number of memory elements. The constraint length of an encoder is the number of incoming bits and the number of bits in the memory that effect the generation of n output bits. It is defined as sum of number of input stream and number of memory elements. Typically if k=1 then the constraint length would be m+1. In convolutional encoding, first all the memory elements are set to zero. Then k input message bits are sent which generates the k code bits. The encoder state is returned to all zero's state by feeding m zero's in, which results in the generation of remaining m code bits. Consider a (2, 1, 3) convolutional encoder. The code rate is 1/2 with 1 input, where each input is coded into 2 outputs generated from the 2 modulo adders. The generator polynomials specify which bits need to be added to obtain the output codeword. Since we have two outputs, we will have two generator sequences corresponding to each output.

For example, let the generator sequences be $g_1 = [1, 1, 0, 1]$, $g_2 = [1, 1, 1, 1]$. For the specified generator sequences, a (2,1,3) convolutional encoder would be as shown in Fig. 3.1.

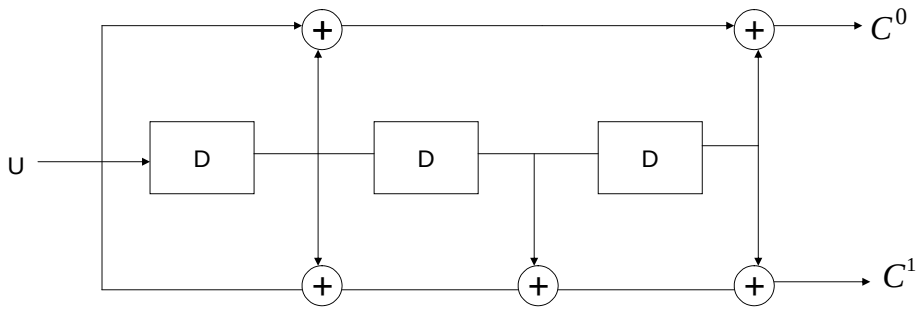

Figure 3.1: A (2,1,3) Convolutional Encoder

The two outputs of the convolutional encoder can be obtained by convolving the input bits with the generator sequences since the convolutional encoder is a linear system.

$$\mathbf{c}_1 = \mathbf{u} * \mathbf{g}_0 = (c_0^1, c_1^1, c_2^1, \ldots) \tag{3.1}$$

$$\mathbf{c}_2 = \mathbf{u} * \mathbf{g}_1 = (c_0^2, c_1^2, c_2^2, \ldots) \tag{3.2}$$

where the discrete convolution is performed using modulo-2 arithmetic.

The two encoded output sequences are then multiplexed into one codeword sequence which is transmitted across the channel. The codeword is

$$\mathbf{c} = (c_0^1 c_0^2, c_1^1 c_1^2, \ldots) \tag{3.3}$$

consider an input sequence $\mathbf{u} = [1011]$ Then the two outputs would be

$$\mathbf{c}_0 = [1011] * [1101] = [1111111] \tag{3.4}$$

$$\mathbf{c}_1 = [1011] * [1111] = [1101001] \tag{3.5}$$

$$\mathbf{c} = [11, 11, 10, 11, 10, 10, 11] \tag{3.6}$$

3.2 Encoder Representations

3.2.1 State Diagram

A convolutional encoder is a linear sequential circuit comprising a shift register. The contents of the shift register are called the state of encoder. Each element of the shift register can have any of the two values i.e either 0 or 1. If m is the number of elements in the convolutional encoder's shift register, then it would have 2^m different possible states. At any time the encoder resides in one of these possible states. A transition between states is caused by each new input bit. A state diagram provides information about the input causing transition between states, output, initial and final states. In a state diagram, the state of the encoder are represented by circles. The path between states is labeled as u/c, where u

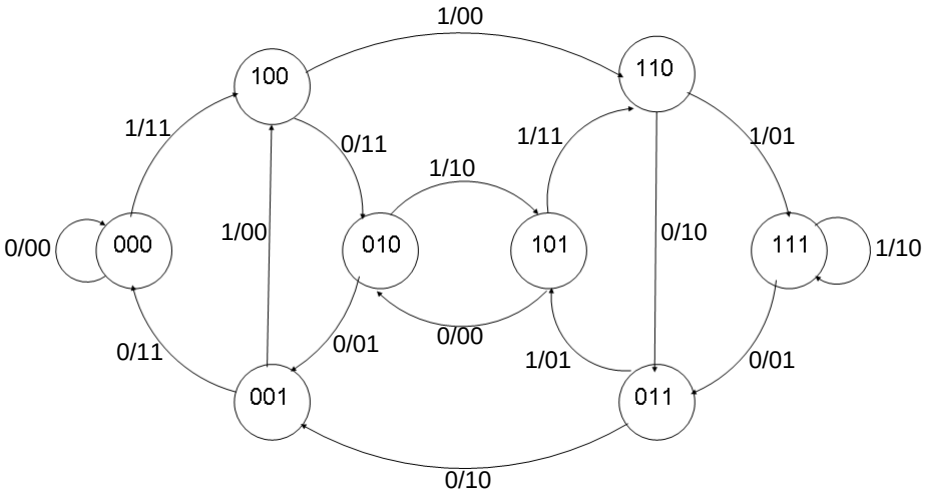

Figure 3.2: State diagram for a (2,1,3) convolutional encoder

represents the input information bit and c represents the output encoded bit or set of bits. Usually convolutional encoding starts from all-zeros state [9].

For the (2,1,3) convolutional encoder stated above, the state diagram is shown in Fig. 3.2. For the input sequence **u**=[1011] starting from all zeros state goes through the states $S = (000, 100, 010, 101, 110)$. Then by feeding m zero inputs it goes through states $S = (011, 001, 000)$ resulting in the output codeword **c**=(11,11,10,11,10,10,11).

3.2.2 Tree Diagram

Instead of directly jumping from one state to another as in state diagram, in tree diagram we go step by step depending on whether the input bit is either 1 or 0. The starting state is assumed to be all zeros. All possible information and encoded sequences for the convolutional encoder is given by the tree diagram. If input information bit 0 is received we go up and if input bit 1 is received we go downwards the tree. Bits on the branches of the tree are

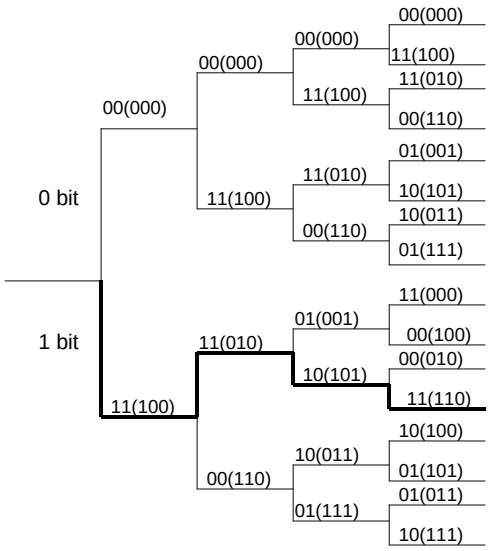

Figure 3.3: Tree diagram for a (2,1,3) convolutional code

output bits and resulting output state of the convolutional encoder is specified inside the parenthesis. Any input information sequence follows path through the tree diagram from left to right. Consider the input sequence $\mathbf{u} = [1011]$. The sequence is traced along the branches of the tree shown in solid lines which gives k coded bits. For remaining m coded bits jump back to the 110 state and input trace along the tree for m zero input bits thus producing final n codewords [9]. Tree diagram for the above (2,1,3) convolutional encoder is shown in Fig. 3.3.

3.2.3 Trellis Diagram

Trellis diagram is the most common representation of the convolutional encoder. It is similar to a state diagram. Linear time sequencing of events is represented by trellis which makes it more opted than tree and state diagrams despite of being messy. In trellis diagram discrete time is represented along the x-axis and states along the y-axis. Transition occurs

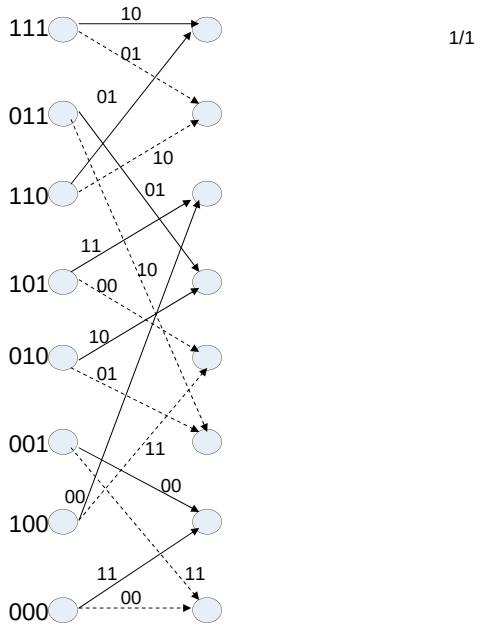

Figure 3.4: Trellis Section for a (2,1,3) Encoder

horizontally through trellis with time. A trellis diagram is drawn by writing all possible 2^m states along the y-axis. At each state we have only two options, either input 0 or 1. Depending on the input each state is connected to other state with the output bits written on the connections between two states. Dashed lines represent 0 input bit and solid lines represent input bit 1. As each period repeats the transitions we can draw it for as many periods as we want. The trellis diagram is unique for each code. The trellis section for the (2,1,3) convolutional encoder in Fig. 3.1 is shown in Fig. 3.4. The trellis diagram, encoding the input sequence **u**=[1011] is as shown in Fig. 3.5. From Fig. 3.5 we can see that the encoded output codeword sequence is **c**=[11,11,10,11,10,10,11].

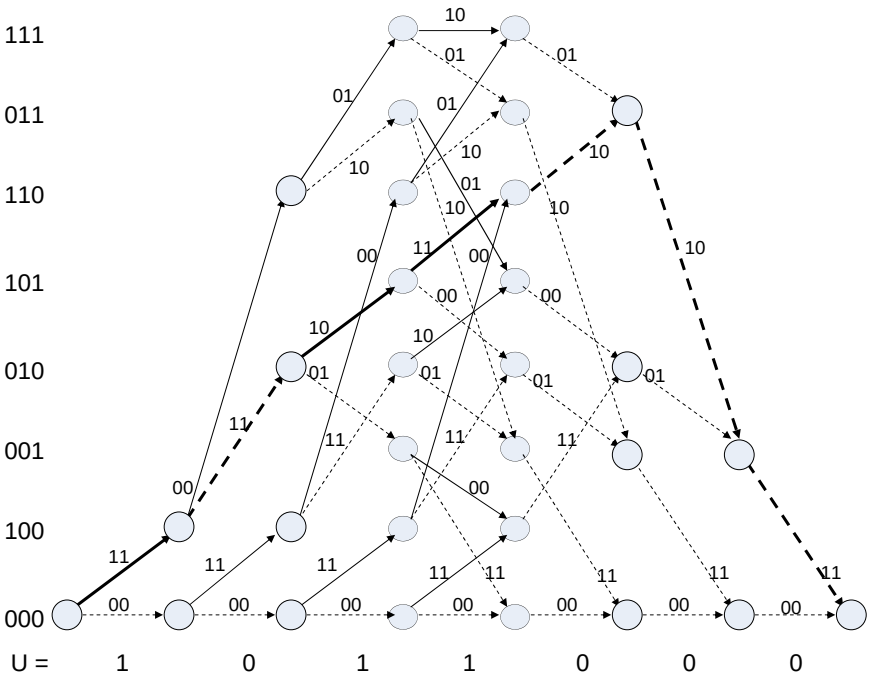

Figure 3.5: Trellis Diagram

3.3 Tail-biting Encoding

The convolutional encoder is a finite state machine. Since convolutional codes are strictly not block codes, the decoder operates on continuous stream of encoded data which is split into traceback lengths for further processing. Splitting the data into packets would be convenient, where each packet contains many traceback lengths of data. Usually for conventional codes it is assumed that the encoder starts in the all-zeros state. When the encoding is done, the encoder's end state may be a state other than all-zeros. Conventional codes will use zero padding to force the trellis to all zero state. Consider an (n, k, m) encoder, then m zeros will be padded for each of the k input sequences of length L producing $n(L + m)$ output bits. Then the effective rate i.e average number of input bits carried by an output bit is given as [10]

$$R_{eff} = \frac{KL}{n(L+m)} = R\frac{L}{L+m} \tag{3.7}$$

$\frac{L}{L+m}$ is the fractional rate loss. Hence the disadvantage is that extra bits need to be transmitted at the cost of extra transmission time, and as a consequence a larger $\mathcal{E}_b/N_0$ is required for a given probability of error. The advantage of termination is that it is easy to implement and does not effect the error correction capability of the convolutional code.

Tail-biting encoding tries to overcome the need to transmit extra termination bits. In tail-biting encoding the encoder's m shift registers are initialized with the last m bits of the input information sequence. Output obtained during the initialization is not transmitted across the channel. This method ensures that the encoder start and end state are same [11].

Consider a (2,1,3) convolutional encoder specified in section 3.1. The trellis section and trellis diagram for encoding the input sequence $\mathbf{u} = [1011]$ are shown in Fig. 3.6 and Fig. 3.7 respectively. The encoded sequence obtained from the tail-biting encoding process is $\mathbf{c} = [01, 01, 01, 11]$.

The advantage of tail-biting encoding is that code rate remains same and the error correction properties of convolutional code are not significantly affected. The disadvantages are that the complexity is increased and over the trellis the decoding latency is increased.

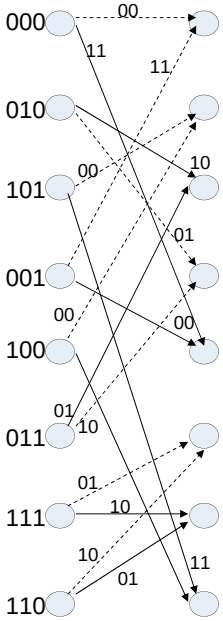

Figure 3.6: Tailbiting Trellis section for a (2,1,3) convolutional code

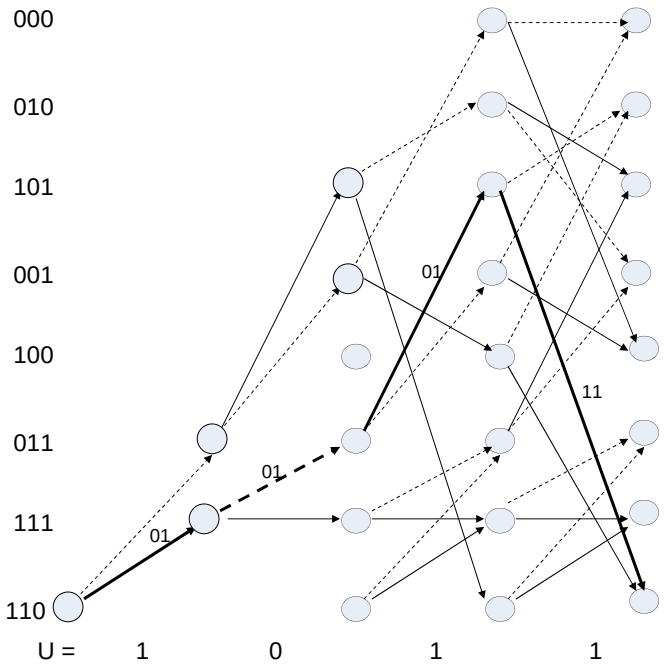

Figure 3.7: Tailbiting Trellis diagram for a (2,1,3) Encoder

Chapter 4

Decoding of Convolutional Codes

4.1 Covolutional Decoding

The output of a convolutional encoder is a sequence of code bits which are sent over a noisy channel. The goal of a *maximal-likelihood* decoder is to determine which bits were sent, given the noisy received sequence. There are several ways to perform decoding. A brute-force method of decoding is to compare the received sequence against all possible transmitted sequences and pick the most likely. However, this type of decoding is too complex for all but the shortest sequences.

The complexity of decoding can be significantly reduced by exploiting the fact that the convolutional encoder has a finite memory. Decoding could progress on the tree or the trellis used to represent the encoder. For instance, a decoder could try to find the most likely branch in the code tree by exploring different paths. If a particular path through the tree is improbable, the decoder could give up on that path and try a different one. The idea of trying different branches of the code tree is the main idea behind sequential decoding. While sequential decoding has a much lower complexity than brute-force decoding, it does not guarantee a maximal-likelihood solution.

Another approach to decoding is to represent the code in terms of a trellis and try to find the most likely path through the trellis. Because of the finite memory of the encoder, each state in the trellis will have many different paths going through it. At each state, only the most likely partial path going into needs to be retained. All paths other than the most

likely path can be discarded, because they will never result in a path that is more likely. The idea of using the trellis structure to eliminate unlikely paths is the main idea behind the Viterbi algorithm. Like sequential decoding, the Viterbi algorithm reduces complexity relative to brute-force decoding. However, unlike sequential decoding, the Viterbi algorithm is guaranteed to find the maximal-likelihood solution.

4.2 Hard Decision Viterbi Algoritm

The Viterbi decoding algorithm mainly depends on the structure called trellis diagram which is expansion of the state diagram in time i.e. each time unit is represented with a separate state diagram.

Write all possible 2^m states along the y-axis and for one period of time show the transitions from one state to another depending on the input bits 0 or 1. Input bit 0 is represented by a dashed line and 1 is represented by a solid line. Then draw the trellis diagram which is just repetition of the compact trellis section for n time periods. The Viterbi algorithm is as follows: Label each branch of the trellis with a branch metric, i.e. the Hamming distance between the received codeword and code bit associated with the trellis.

$$\gamma = d_h(r, c) \tag{4.1}$$

Where r is the received bits for this trellis section, c is the code bits associated with the trellis branch.

Starting with time $t = 0$, compute the partial path metric and label each state with it. Partial metric is computed by adding metric of the connecting survivor at the preceding time unit with the branch metric entering that state. It is computed for all paths entering the state in the remaining time units.

$$\alpha = \sum_{i=1}^{t-1} \gamma_i \tag{4.2}$$

If there are two paths entering a state, then choose one with small metric and disregard the other one, i.e. perform add select operation at each state and choose one with small metric,

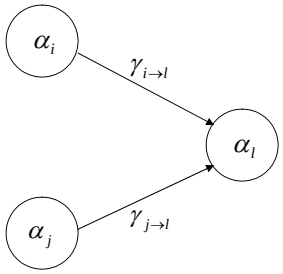

Figure 4.1: Add Compare Select

$$\alpha_l \quad = \quad \min[(\alpha_i + \gamma_{i\to l}), (\alpha_j + \gamma_{j\to l})] \tag{4.3}$$

where $\gamma_{i\to l}$ and $\gamma_{j\to l}$ are the metrics associated with the branches connecting i and l or j and l respectively. During traceback from the last time unit to first, the maximum likelihood path will be the surviving path which determines the corresponding $\hat{u}$ for this path [8].

Consider (2,1,3) convolutional code stated in chapter 3 with length of the codeword $n = 7$ received codeword and generator sequence $g_1 = [1, 1, 0, 1]$, $g_2 = [1, 1, 1, 1]$ then the received codeword is c=[11,11,10,11,10,10,11]. Trellis diagram is as shown in Fig. 1.2, from which the information sequence is estimated.

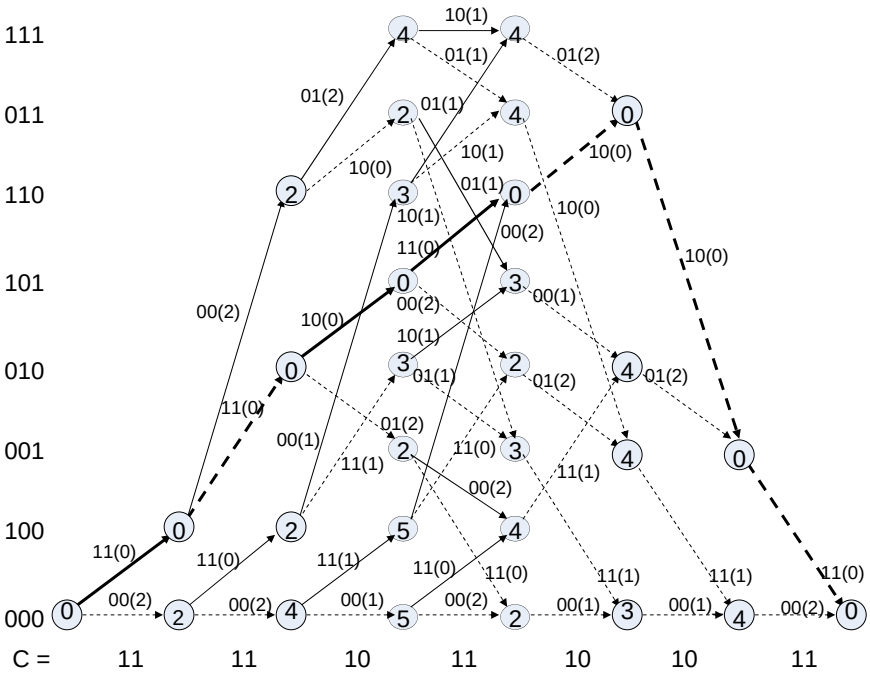

Figure 4.2: Hard Decision Viterbi decoding for a (2,1,3) convolutional code

4.3 Decoding of Tail-biting Codes

In tail-biting codes, the encoder starts and ends in the same state, but the state remains unknown to the decoder. The advantage of tail-biting convolutional codes is that overhead of the framed code with tail can be avoided. A Viterbi decoder with a known starting and ending state can be achieved by making number of known information bits terminate the convolutionally encoded sequences. Disadvantage of this method is requirement of overhead due to which there is a fractional rate loss. Consider an (n, k, m) convolutional code with rate $R = k/n$, number of memory elements m. Then when N is the length of the encoded codeword, the rate is

$$R = \frac{k}{n}\frac{N-m}{N} \tag{4.4}$$

When $N \gg m$ i.e. for long blocks, the rate loss is small.

To overcome the fractional rate loss, encoding and decoding is performed without a tail [12]. One of the simple decoding algorithm for tail-biting convolutional codes achieving maximum likelihood performance is circular decoding [6]. It is assumed that the trellis is circular and each received codeword is considered to be a path around circular trellis of L stages, where L is the number of information bits. It works by traveling through the trellis circle more than once without stopping. Circular decoding is a simple and effective method for decoding a tail-biting code. It achieves maximum likelihood decoding performance as the number of cycles approaches infinity. Performance of decoding depends on the length of the trellis. The decoder is not limited to number of cycles while decoding and must satisfy bound specifying the number of errors needed to be corrected [13].

Consider an (n, k, m) tail-biting convolutional code with rate $1/n$, constraint length K. Let the length of the input information bits be L and length of codeword be $N = nL$ which is transmitted over additive white Gaussian channel. Let the minimum distance of the codeword be d_{min} and the free distance be d_{free}. When not in tail-biting mode, $d_{free} \geq d_{min}$. Circular decoding uses the Viterbi decoding trellis of length $H + L + T$ where H, T are the head decoding length and tail decoding length respectively. The starting nT bits of a received sequence are added at the end and the last nH bits are appended at the beginning

of the received sequence. Consider a received sequence $\mathbf{r}$ of length n [6].

$$\mathbf{r} = (r_1, r_2, \ldots, r_n) \tag{4.5}$$

Adding the last nH bits i.e. $(r_1, r_2, \ldots, r_{nH})$ at first and adding the first nT bits $(r_{N-nT+1}, \ldots, r_n)$ at the end of the received sequence changes it to the form

$$\mathbf{r} = (r_{N-nH+1}, \ldots, r_N, r_1, r_2, \ldots, r_N, r_1, r_2, \ldots, r_{nT}) \tag{4.6}$$

The resulting codeword is then decoded with the help of the Viterbi decoding trellis of length $L+H+T$ by initializing all of the initial trellis states with the same metric (indicating equally-likely starting states), and performing the traceback from state with best metric at the end of the trellis. From the obtained decoded sequence discard the first H bits and the last T bits to obtain the final decoded sequence of length L [6].

Once the decoding by the circular Viterbi algorithm is done correctly, it can decode the next codewords by using less trellis length which can be known as depth. The decoding depth can be defined as the depth at which it is more likely that the surviving paths merge. The depth increases for long codes [13].

4.4 Soft Decision Decoding

Soft decision decoding uses multi-bit quantization on the received channel values unlike hard decision Viterbi decoding which uses single bit quantization. The ideal soft decision decoder which uses the infinite bit quantization has the received channel values stored directly in the channel encoder [9]. Consider an (n, k, m) convolutional encoder with length of the information sequence being L. Let the input information sequence be $u = (u_0, u_1, \ldots, u_{kL-1})$, received sequence be $\mathbf{r} = (r_0, r_1, \ldots, r_{N-1})$ and the encoded codeword be $c = (c_0, c_1, \ldots, c_{N-1})$. The decoder needs to generate the estimated sequence $\hat{\mathbf{c}}$ from the received sequence $\mathbf{r}$. A maximum likelihood decoder chooses the codeword $\mathbf{c}$ which maximizes the log-likelihood function $\log P(\mathbf{r}|\mathbf{c})$.

$$P(\mathbf{r}|\mathbf{c}) = \prod_{i=0}^{n-1} P(r_i|c_i) \tag{4.7}$$

Since channel is memoryless

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\epsilon C} P(\mathbf{r}|\mathbf{c}) \tag{4.8}$$

Normalize it by all zeros codeword

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\epsilon C} \frac{P(\mathbf{r}|\mathbf{c})}{P(\mathbf{r}|0)} = \arg\max_{\mathbf{c}\epsilon C} \prod_{i=0}^{n-1} \frac{P(r_i|\mathbf{c}=c_i)}{P(r_i|\mathbf{c}=0)} \tag{4.9}$$

Take log monotonic function

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\epsilon C} \log \prod_{i=0}^{n-1} \frac{P(r_i|\mathbf{c}=c_i)}{P(r_i|\mathbf{c}=0)} \tag{4.10}$$

$$\mathbf{c} = \sum_{i=0}^{n-1} \log \frac{P(r_i|(\mathbf{c}=c_i)}{P(r_i|(\mathbf{c}=c_i))} \tag{4.11}$$

If c_i=0

$$\log \frac{P(r_i|\mathbf{c}=0)}{P(r_i|\mathbf{c}=0)} = 0 \tag{4.12}$$

If c_i=1

$$\log \frac{P(r_i|\mathbf{c}=1)}{P(r_i|\mathbf{c}=0)} \triangleq \lambda_i \tag{4.13}$$

Where λ_i is the log-likelihood ratio which if positive, **c**=1 is more likely if negative, **c**=0 is more likely

Equation used for soft decision decoding is [8]

$$\mathbf{c} = \arg\max_{\mathbf{c}\epsilon C} \sum_{i=0}^{n-1} c_i\lambda_i. \tag{4.14}$$

Soft-decision decoding is performed using the Viterbi algorithm. The algorithm is just like the hard-decision algorithm described in Section 4.2 except that the branch metric γ is now:

$$\gamma = \sum_{i=1}^{n} c_i\lambda_i \tag{4.15}$$

where c_i is the i^{th} code bit associated with the branch.

Chapter 5

Results and Conclusion

5.1 Results

The results obtained from the simulations are presented here. The simulations were run until there were 200 independent errors at each signal-to-noise ratio. Convolutional codes are encoded and decoded by using tail-biting method. A convolutional encoder of rate 1/2 with constraint length K=7, generator matrices $g_1 = 171_{oct}$=[1 1 1 1 0 0 1], $g_2 = 133_{oct}$=[1 0 1 1 0 1 1] is used [3]. The encoder diagram is shown in Fig. 5.1.

Code rates higher than 1/2 were achieved using the puncturing patterns given in Table. 2.1. Bit error rate (BER), frame error rate (FER) of QPSK, 16-QAM, 64-QAM are plotted and compared for different code rates, depth, frame size. Table 5.1 provides the modulations, lengths and code rates considered for the simulation [3].

BER of QPSK with constant rate, frame size and varying depth is shown in Fig. 5.2 and FER of QPSK with constant rate, frame size and varying depth is shown in Fig. 5.3. It can be seen from the two plots that as depth increases error rate decreases. In Fig. 5.4 and Fig. 5.5 behavior of BER and FER of QPSK for constant depth, rate and varying frame size is shown. It can be observed from Fig. 5.5 that as frame size increases frame error rate increases. BER, FER of 16-QAM,64-QAM modulations for varying frame size are shown in Fig. 5.6 and Fig. 5.7 respectively. By comparing the curves it can be seen that performance of 16-QAM is better than 64-QAM. BER, FER of QPSK, 16-QAM, 64-QAM modulations are plotted and compared for varying frame size in Fig. 5.8 and Fig. 5.9. By comparison

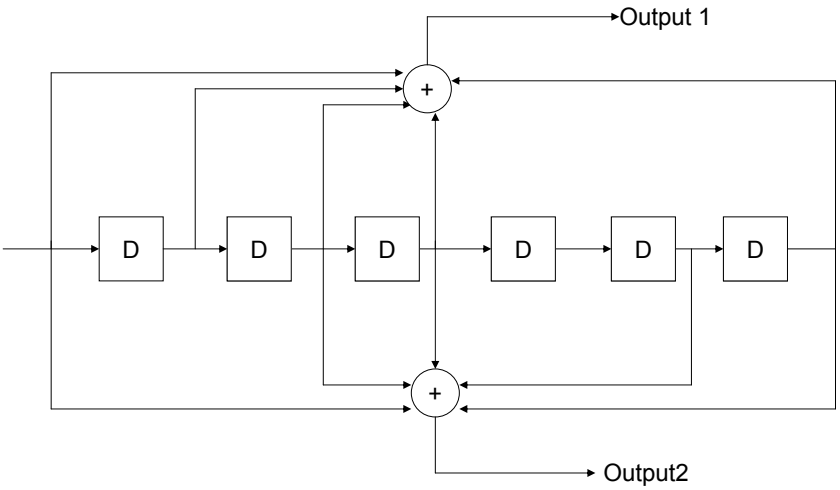

Figure 5.1: Rate 1/2 convolutional encoder from the IEEE 802.16 standard

	QPSK		16 QAM		64 QAM		
Encoding rate	R=1/2	R=3/4	R=1/2	R=3/4	R=1/2	R=2/3	R=3/4
Data payload (bytes)	6						
		9					
	12		12				
	18	18		18	18		
	24		24			24	
		27					27
	30						
	36	36	36	36	36		

Table 5.1: Payload length, code rate, and modulation combinations that were simulated. The coding scenarios are as defined in the IEEE 802.16 standard [3]

it can be seen that QPSK performance is better than 16-QAM and 16-QAM performance is better than 64-QAM.

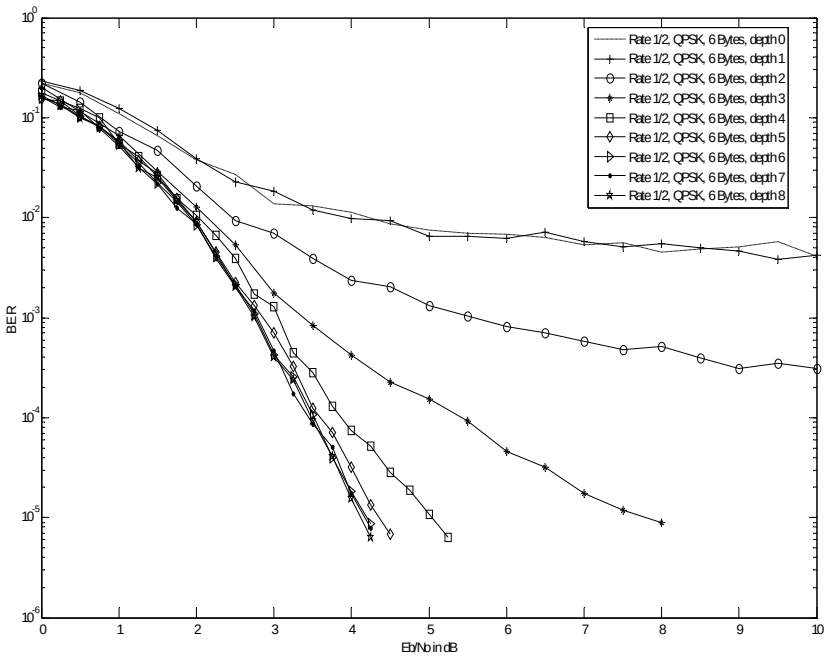

Figure 5.2: Bit Error Rate of QPSK for varying depths

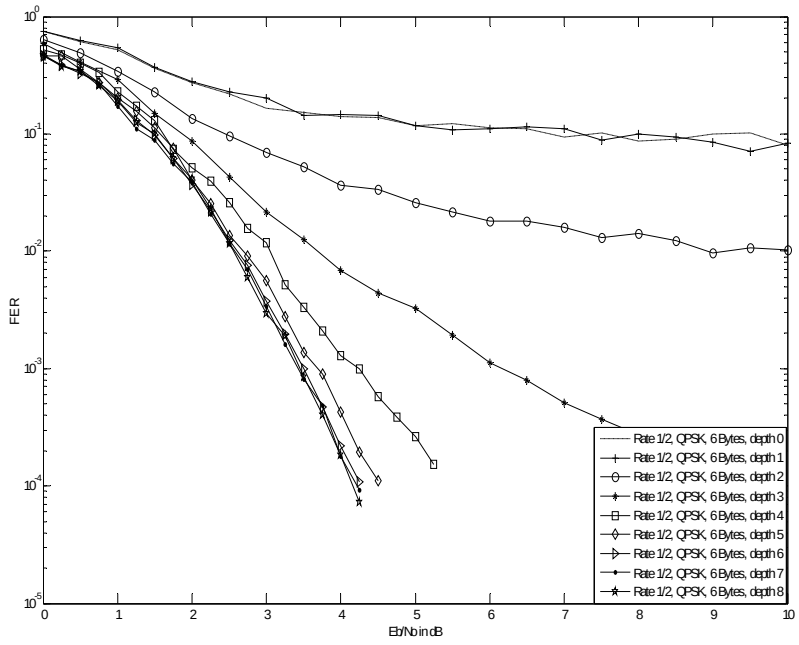

Figure 5.3: Frame Error Rate of QPSK for varying depths

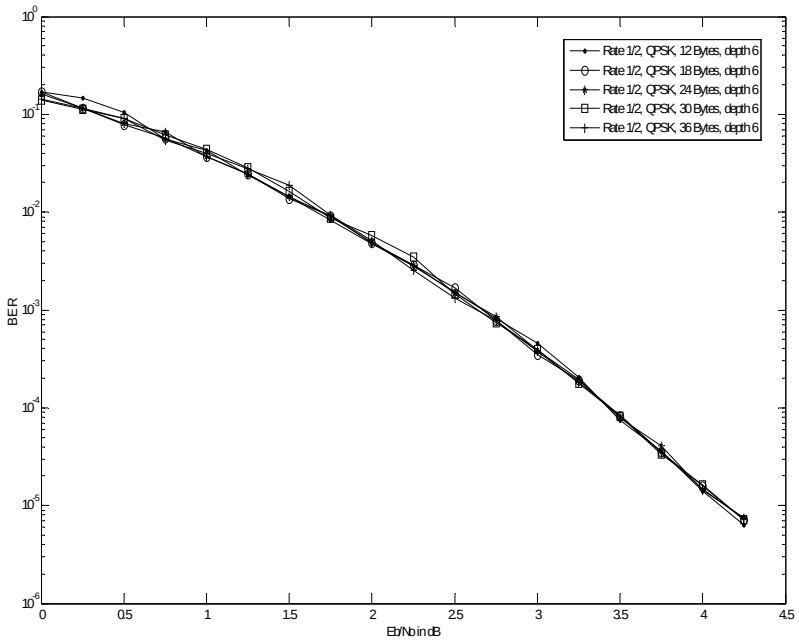

Figure 5.4: Bit Error Rate of QPSK for varying frame size

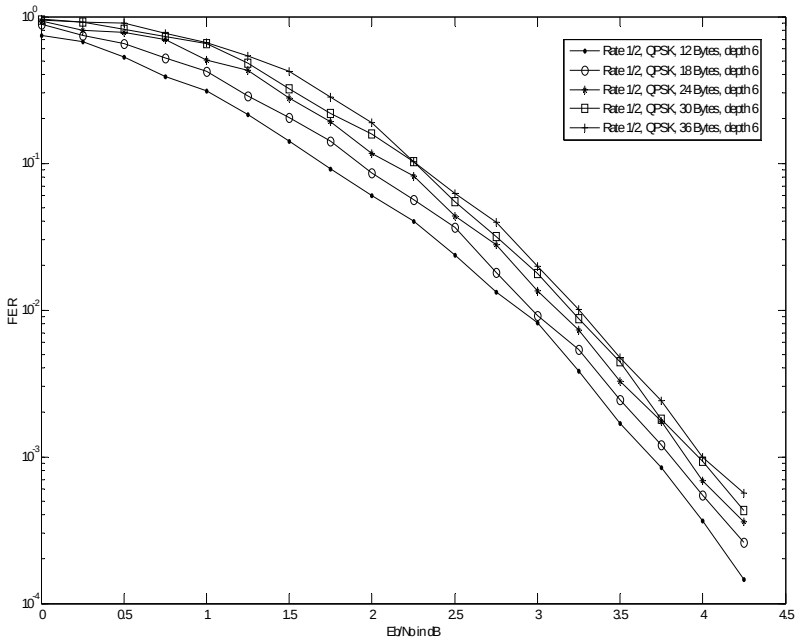

Figure 5.5: Frame Error Rate of QPSK for varying frame size

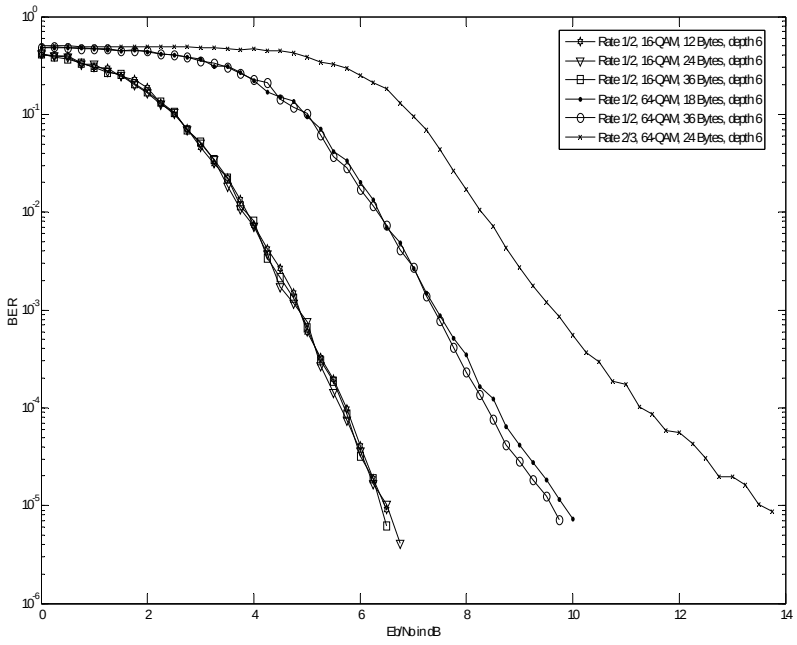

Figure 5.6: Bit Error Rate of QAM for varying frame size

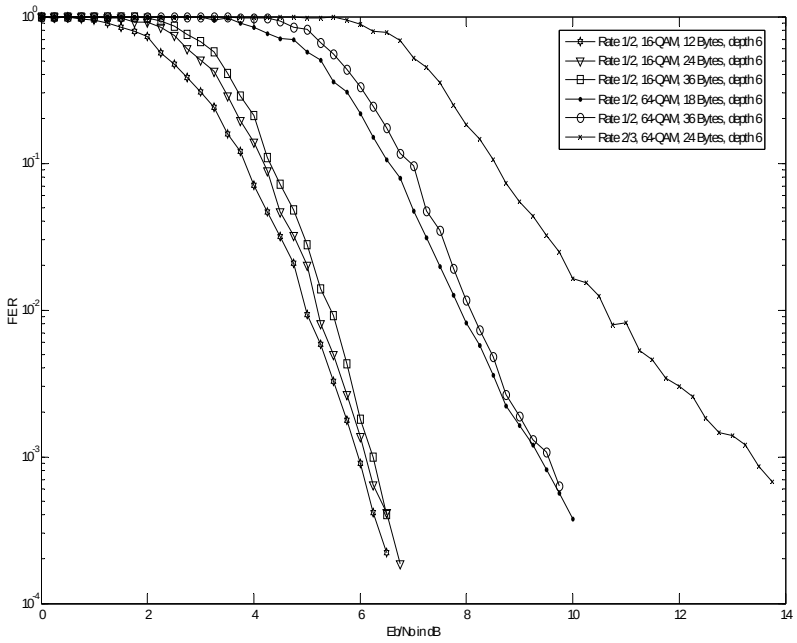

Figure 5.7: Frame Error Rate of QPSK for varying frame size

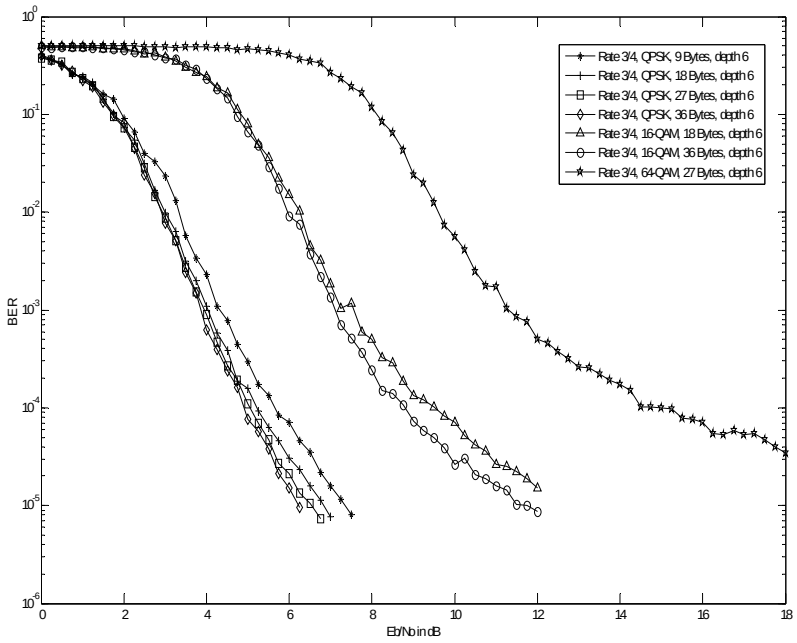

Figure 5.8: Bit Error Rate of QAM, QPSK for varying frame size

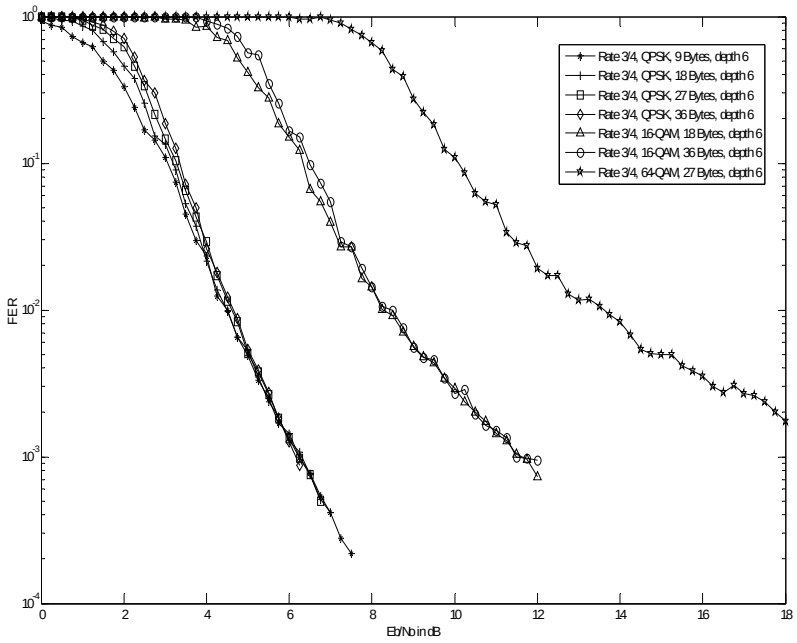

Figure 5.9: Frame Error Rate of QAM, QPSK for varying frame size

5.2 Conclusion

Error control coding plays a vital role in communications. Various types of error control codes exist of which tail-biting convolutional code is significant. In this report we discuss encoding, decoding techniques of conventional convolutional, tail-biting convolutional codes and reason for opting tail-biting codes. Simulations of QPSK, 16-QAM, 64-QAM modulations are performed by varying depth, code rate, frame size and BER, FER curves are plotted for better understanding and comparison.

References

[1] Jin Jin and Bacochun Li, "Adaptive random network coding in wimax," in *Proc. IEEE Int. Conf. on Commun.*, 2008, pp. 2576–2580.

[2] Loutfi Nuaymi, *WiMAX:Technology for wireless broadband access*, John Wiley and Sons, 2007.

[3] IEEE computer society, IEEE microwave theory, and techniques society, "Air interface for fixed broadband wireless access systems," *IEEE Trans. Commun.*, pp. 303–316, Oct. 2004.

[4] Muhammad nadeem Khan and Sabir Ghauri, "The wimax 802.16e physical layer model," in *Proc. Int. Conf. on Wireless, Mobile and Multimedia networks.*, Jan. 2008, pp. 117–120.

[5] Bernhard M.J Leiner, "Ldpc codes- a brief tutorial," pp. 1–9, Apr. 2005.

[6] Wonjin Sung, "Minimum decoding trellis lengths for tail biting convolutional codes," *Electronics Letters*, pp. 643–645, Mar. 2000.

[7] Shu Lin and JR. Daniel J. Costello, *Error control coding Fundamentals and applications*, Prentice Hall, 1983.

[8] Mathew valenti, "Coding theory," Course Notes, WVU, Spring 2008.

[9] Charan Langton, "Coding and decoding with convolutional codes," *Complex to Real*, pp. 1–29, 1999.

[10] Yunghsiang S. Han, "Introduction to binary convolutional codes," Course Notes, National Taipei University, Fall 2004.

[11] Howard H. Ma and Jack K Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun.*, pp. 104–111, Feb. 1986.

[12] R. V. Cox and C-E. W. Sundberg, "A circular viterbi algorithm for decoding tailbiting convolutional codes," in *Proc. IEEE Veh. Tech. Conf.*, may 1993, pp. 104–107.

[13] John B. Anderson and Stephen M. Hladik, "An optimal circular viterbi decoder for the bounded distance criterion," *IEEE Trans. Commun.*, pp. 1736–1742, Nov. 2002.

[14] Matthew C. Valenti, "The evolution of error control coding," Course Notes, WVU, Spring. 2007.

[15] Ajay Dholakia, *Introduction to convolutional codes with applications*, springer, 1994.

[16] Matthew C. valenti, *Iterative coded solutions modulated library*, May. 2008.