ABSTRACT

The project here presents the essence of error free transfer of message between two nodes.To make the communication error free Error Correcting Codes (ECC) are used, which is achieved by using encoder and decoder pair(codec). Thus the work here is to concentrate on channel coding techniques.Channel encoders are designed in such way that the message being transferred will be robust and tolerant to the errors caused in channel due to channel abnormalities. And then channel decoder is designed to decode the error free message at the receiver.

Errors caused in the channel are because of AWGN (Additive White Gaussian Noise), multipath fading and interference. So, to measure the errors BER(Bit Error Rate)is used, which is dependent on SNR(Signal to Noise Ratio). Errors may be burst errors or single bit errors,but they make the message redundant.

ECC provides the techniques to detect and correct the errors at the receiver.The techniques are FEC(Forward Error Correction) and REC(Reverse Error Correction). This project considers FEC for implementing the channel coding techniques to overcome channel abnormalities. FEC techniques include Block codes,convolutional codes,Cyclic codes and so on. In first phase of the project channel coding tech-

niques are tested for their performance with respect to BER, by simulinting their models in Simulink and BERTool. And in next phase channel coding techniques are simulated in Xilinx and ModelSim, by preparing the verilog codes nad these codes are made to run on FPGA,to measure the delay,number of gates and other factors to implement on hardware.

ACKNOWLEGEMENT

A Project work is team effort of all individuals and it cannot be accomplished by an individual all by him. We are grateful to the Almighty and a number of individuals whose professional guidance, assistance and encouragement have made it a pleasure endeavor to undertake this project.

We take this opportunity to express our profound gratitude to our principal **Dr. T. N. Sreenivasa** for his constant support and encouragement.

We are grateful to our Head of the Department **Prof. N. V. Uma Reddy**. Her timely guidance and knowledge impartment has made this project a reality.

We are honoured to be guided by **Prof. K. Jayaraman** without whose guidance the project would have been just a dream.We also thank him his kind and technical support for completion of our project.

We are grateful to be guided by **Mr. Chetan Adhikary. Y**, Assistant Professor, Department of ECE, AMCEC. We also thank him for his unfailing encouragement and suggestions given to us in the course of our project.

We are also indebted to all the staff members of the Department of Electronics and Communication Engineering for their constant support and encouragement.

MANOJ TM

SHREEDHAR SAI KRISHNA EV

TUSHARA R

Contents

List of Figures

Chapter 1

Introduction

In recent years, there has been an increasing demand for efficient and reliable digital data transmission and storage systems. This demand has been accelerated by the emergence of large-scale, high-speed data networks for the exchange, processing, and storage of digital information in the military, governmental, and private spheres. A merging of communications and computer technology is required in the design of these systems. A major concern of the designer is the control of so that reliable reproduction of data can be obtained.In 1948, Shannon demonstrated in a landmark paper that, by proper encoding of the information, errors induced by a noisy channel or storage medium can be reduced to any desired level without sacrificing the rate of information transmission or storage. Since Shannon's work, a great deal of effort has been expended on the problem of devising efficient encoding and decoding methods for error control in a noisy environment. Recent developments have contributed toward achieving the reliability required by today's high-speed digital systems, and the use of coding for error control has become an integral part in the design of modern communication systems and dig-

ital computers.The transmission and storage of digital information have much in common. They both transfer data from an information source to a destination. A typical transmission system may be represented by the block diagram shown in figure **??**.

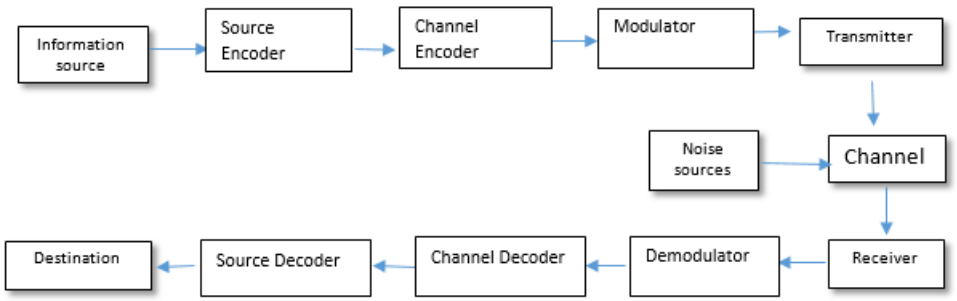

Figure 1.1: Basic communication System

1.1 Information source

The objective of any communication system is to convey information from one point to the other. The information is a very generic word signifying at the abstract level anything intended for communication, which may include some thoughts, news, feeling, visual seen and so on. The information source converts this information into a physical quantity. This physical manifestation of the information is termed as message signal. The message signal also usually will be in non-electrical form. For electrical communication process, first we need to convert the message signal to the electrical form, which is achieved using a suitable transducer.

1.2 Source Encoder

The source encoder is ideally designed so that the number of bits per unit time required to represent the source output is minimized, and the source output can be reconstructed from the information sequence without ambiguity.

1.3 Channel Encoder

The channel encoder transforms the information sequence into a discrete encoded sequence called a code word. In most cases code word is a binary sequence, although in some applications non binary codes have been used. Channel encoding provides better BER i.e. it provides techniques to detect and correct the errors in received signal, encryption for data being transmitted.

1.4 Modulator

The signals are transformed into the form suitable for transmission through the channel. This transformation process is modulation. Modulation may be analog or digital but the purpose is same. This improves the span of signals through the transmission link. ASK, FSK, PSK are the common modulation techniques employed in digital communication.

1.5 Transmitter

The objective of the transmitter block is to collect the incoming message signal and modify in a suitable fashion such that, it can be transmitted via the chosen channel to the receiving point. The functionality of the transmitter block is mainly decided by the type or nature of the channel chosen for communication.

1.6 Channel

Channel is the physical medium which connects the transmitter with that of the receiver. The physical medium includes copper wire, coaxial cable, fibre optic cable, wave guide and free space or atmosphere. The nature of modification of message signal in the transmitter block is based on the choice of the communication channel. This is because the message signal should smoothly travel through the channel with least opposition so that maximum information can be delivered to the receiver. The message signal in the modified form travels through the channel to reach the entry point of the receiver.

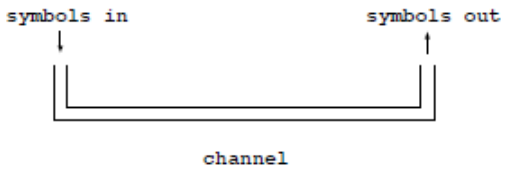

Figure 1.2: Typical channel

1.6.1 Channel Characteristics

- Symbol type: analog waveform, bits, packets.
- Capacity: bandwidth, data rate, packet rate.
- Delay: fixed or variable.
- Fidelity: signal to noise, bit error rate, packet error rate.
- Cost: per attachment, per call, for capacity consumed.
- Reliability
- Security: privacy, unaffordable.
- Order preserving: always, almost always, and usually.
- Connectivity: point to point, 1 to many, many to many.

1.6.2 Noise

Noise is often described as the limiting factor in communication systems: indeed if there was no noise there would be virtually no problem in communications. Noise is a general term which is used to describe an unwanted signal which affects a wanted signal. These unwanted signals arise from a variety of sources which may be considered in one of two main categories:

1. Interference, usually from a human source (man-made)
2. Naturally occurring random noise.

Thermal noise is always presents in electronic systems. Shot noise is more or less significant depending upon the specific devices used for example as FET with an insulated gate avoids junction shot noise. As noted in the preceding discussion, all transistors generate other types of 'non-white' noise which may or may not be significant depending on the specific device and application. Of all these types of noise source, white noise is generally assumed to be the most significant and system analysis is based on the assumption of thermal noise. This assumption is reasonably valid for radio systems which operates at frequencies where non-white noise is greatly reduced and which have low noise 'front ends' which, as shall be discussed, contribute most of the internal (circuit) noise in a receiver system. At radio frequencies the sky noise contribution is significant and is also (usually) taken into account. The (S/N) at various stages in a communication system gives an indication of system quality and performance in terms of error rate in digital data communication systems and 'fidelity' in case of analogue communication systems. (Obviously, the larger the (S/N),the better the system will be).

Noise, which accompanies the signal is usually considered to be additive (in terms of powers) and it's often described as Additive White Gaussian Noise, AWGN, noise. Noise and signals may also be multiplicative and in some systems at some levels of (S/N),this may be more significant than AWGN. In order to evaluate noise various mathematical models and techniques have to be used, particularly concepts from statistics and probability theory, the major starting point being that random noise is assumed to have a Gaussian or Normal distribution.

Noise Sources

1.6.3 Additive White Gaussian Noise

Noise in Communication Systems is often assumed to be Additive White Gaussian Noise (AWGN). **Additive**

Noise is usually additive in that it adds to the information bearing signal. A model of the received signal with additive noise is shown in **??**.

The signal (information bearing) is at its weakest (most vulnerable) at the receiver input. Noise at the other points (e.g. Receiver) can also be referred to the input.

The noise is uncorrelated with the signal, i.e. independent of the signal and we may state, for average powers

$OutputPower = SignalPower + NoisePower$

$output\,power = (S+N)$

White

As we have stated noise is assumed to have a uniform noise power spectral density, given that the noise is not band limited by some filter bandwidth.

We have denoted noise power spectral density by $p0(f)$.

White noise = $p0(f)$ = Constant

Also Noise power = $p0Bn$

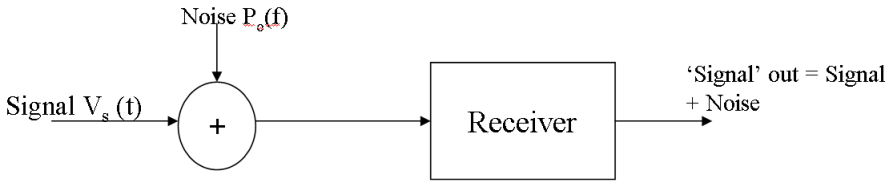

Figure 1.3: Equivalent to AWGN channel

Gaussian

We generally assume that noise voltage amplitudes have a Gaussian or Normal distribution.

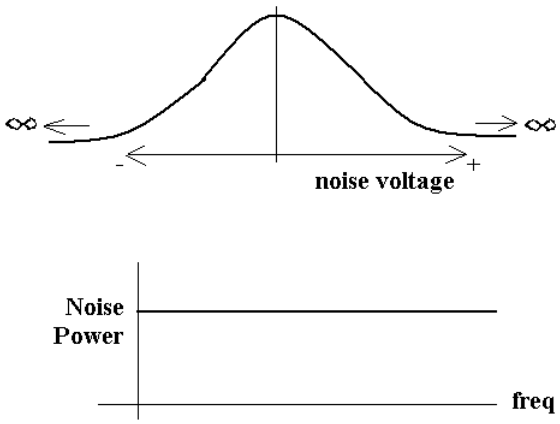

Figure 1.4: Gaussian Noise

1.6.4 Rayleigh Fading Distribution

In mobile radio channels, the Rayleigh distribution is commonly used to describe the statistical time varying nature of the received envelop of a flat fading signal, or the envelop of an individual multipath component.it is well known that the envelop of the sum of two quadrature Gaussian noise signals obeys Rayleigh distribution.

1.6.5 Ricean Fading Distribution

When there is a dominant stationary signal component present, such as line-of-sight propagation path, the small-scale fading envelop distribution is Ricean. In such a situation, random multipath components arriving at different angles are super imposed on a stationary dominate signal. At the output of an envelope detector, this has the effect of adding a dc component to the random multipath.

1.6.6 Bit Error Rate

In digital transmission, the number of bit errors is the number of received bits of a data stream over a communication channel that have been altered due to noise, interference, distortion or bit synchronization errors.

In a communication system, the receiver side BER may be affected by transmission channel noise, interference, distortion, bit synchronization problems, attenuation, wireless multipath fading, etc.

The BER may be improved by choosing a strong signal strength (unless this causes cross-talk and more bit errors), by choosing a slow and robust modulation scheme or line coding scheme, and by applying channel coding schemes such as redundant forward error correction codes. A worst-case scenario is a completely random channel, where noise totally dominates over the useful signal. This results in a transmission BER of 50 percent. In a noisy channel, the BER is often expressed as a function of the normalized carrier-to-noise ratio measure denoted $Eb/N0$.

In the case of QPSK modulation and AWGN channel, the BER as function of the Eb/N0 is given by:

$$BER = (1/2)erfc[(Eb/N0)^{1/2}]$$

To enhance the reliability of message during transmission of information carrying symbols through a communication channel, use error control encoder-decoder pair (also known as a codec).

1.7 Receiver

The receiver block receives the incoming modified version of the message signal from the channel and processes it to recreate the original form of the message signal. There are great variety of receivers in the communication systems, depending on the processing required to recreate the original message signal and also the final presentation of the message to the destination. The purpose of receiver and form of the output display influence its construction as much as the type of modulation system used.

1.8 Demodulation

The demodulator performs reverse operation than that is performed by modulator at transmitter side. Thus here the carrier signal is removed and original signal is extracted. Demodulation may coherent type or incoherent.

1.9 Channel Decoder

Channel decoder decrypts the data encrypted by channel encoder at transmitter side. This block helps in detection and correction of errors that are caused due to the perturbations in the channel.

1.10 Source Decoder

Source decoder retrieves the data that is being encoded by the source encoder. It converts the signal suitable for the applications used at the destination.

1.11 Destination

The destination is the final block in the communication system which receives the message signal and processes it to comprehend the information present in it. Usually, the humans will be the destination block or the displays.

Chapter 2

Error Control and Coding

Error correcting coding includes encoder and decoder pair in order to detect and correct the errors. Unlike wired digital networks, wireless digital networks are much more prone to bit errors. Packets of bits that are received are more likely to be damaged and considered unusable in a packetized system. Error detection and correction mechanisms are vital and numerous techniques exist for reducing the effect of bit-errors and trying to ensure that the receiver eventually gets an error free version of the packet. The major techniques used are error detection with Automatic Repeat Request (ARQ), Forward Error Correction (FEC) and hybrid forms of ARQ and FEC (H-ARQ).

The major categories of activities on error control coding can broadly be identified as the following:

1. To find codes with good structural properties and good asymptotic error performance.

2. To devise efficient encoding and decoding strategies for the codes .

3. To explore the applicability of good coding schemes in various digital transmission and storage systems and to evaluate their performance.

Forward Error Correction (FEC) is the method of transmitting error correction information along with the message. At the receiver, this error correction information is used to correct any bit-errors that may have occurred during transmission. The improved performance comes at the cost of introducing a considerable amount of redundancy in the transmitted code. There are various FEC codes in use today for the purpose of error correction. Most codes fall into either of two major categories: block codes and convolutional codes. Block codes work with fixed length blocks of code. Convolutional codes deal with data sequentially (i.e. taken a few bits at a time) with the output depending on both the present input as well as previous inputs. In terms of implementation, block codes become very complex as their length increases and are therefore harder to implement.

The design of error correcting codes and their corresponding decoders is usually done in isolation. The code is often designed first with the goal of minimizing the gap from Shannon capacity and attaining the target error probability. To reflect the concerns of implementation, the code is usually chosen from a family of codes that can be decoded with low complexity. On the implementation side,decoders are carefully designed for the chosen code with the goal of consuming low power while achieving the required decoding throughput.

2.1 Forward Error Correction

Forward Error Correction is a method used to improve channel capacity by introducing redundant data into the message. This redundant data allows the receiver to detect and correct errors without the need for retransmission of the message.

Forward Error Correction proves advantageous in noisy channels when a large number of retransmissions would normally be required before a packet is received without error. It is also used in cases where no backward channel exists from the receiver to the transmitter.

A complex algorithm or function is used to encode the message with redundant data. The process of adding redundant data to the message is called channel coding. This encoded message may or may not contain the original information in an unmodified form. Systematic codes are those that have a portion of the output directly resembling the input and Non-systematic codes are those that do not. It was earlier believed that as some degree of noise was present in all communication channels, it would not be possible to have error free communications. This belief was proved wrong by Claude Shannon in 1948. In his paper titled -A Mathematical Theory of Communication, Shannon proved that channel noise limits its transmission rate and not the error probability. According to his theory, every communication channel has a capacity C (measured in bits per second), and as long as the transmission rate, R (measured in bits per second), is less than C, it is possible to design an error-free communications system using error control codes. The now famous Shannon-Hartley theorem, describes how this channel capacity

can be calculated. However, Shannon did not describe how such codes may be developed. This led to a wide spread effort to develop codes that would produce the very small error probability as predicted by Shannon. There were two major classes of codes that were developed, namely block codes and convolutional codes.

2.1.1 Block Codes

Block codes are described using two integers k and n, and a generator matrix or polynomial. The integer k is the number of data bits in the input to the block encoder. The integer n is the total number of bits in the generated code word. Also, each n bit code word is uniquely determined by the k bit input data. Another parameter used to describe is its weight. This is defined as the number of non-zero elements in the code word.

In general, each code word has its own weight. If all the M code words have equal weight it is said to be fixed-weight code. Hamming Codes and Cyclic Redundancy Checks are two widely used examples of block codes.

2.1.2 Hamming Codes

A commonly known linear Block Code is the Hamming code. Hamming codes can detect and correct a single bit-error in a block of data. In these codes, every bit is included in a unique set of parity bits. The presence and location of a single parity bit-error can be determined by analysing parities of combinations of received bits to produce a table of parities each of which corresponds to a particular bit-error combination. This table of errors is known as the error syndrome. While

Hamming codes are easy to implement, a problem arises if more than one bit in the received message is erroneous. Hence, there is a need for more robust error detection and correction schemes that can detect and correct multiple errors in a transmitted message.

2.1.3 Cyclic Redundancy Codes

Cyclic Codes are linear block codes that can be expressed by the following mathematical property. If C = [c n-1 cn-2 c1 c0] is a code word of a cyclic code, then [c n-2 cn-3 c0 cn-1], which is obtained by cyclically shifting all the elements to the left, is also a code word. In other words, every cyclic shift of a code word results in another code word. This cyclic structure is very useful in encoding and decoding operations because it is very easy to implement in hardware. A cyclic redundancy check or CRC is a very common form of cyclic code which is used for error detection purposes in communication systems. At the transmitter, a function is used to calculate a value for the CRC check bits based on the data to be transmitted. These check bits are transmitted along with the data to the receiver. The receiver performs the same calculation on the received data and compares it with the CRC check bits that it has received. If they match, it is considered that no bit errors have occurred during transmission. Using different kinds of generator polynomials, it is possible to use CRCs To detect different kinds of errors such as all single bit-errors, all double bit errors, any odd number of errors, or any burst error of length less than a particular value. Due to these properties, the CRC check is a very useful form of error detection. The IEEE 802.11 standard for CRC check polynomial is the CRC-32.

2.1.4 Convolutional Codes

Convolutional codes are codes that are generated sequentially by passing the information sequence through a linear finite-state shift register. A convolutional code is described using three parameters k, n and K. The integer k represents the number of input bits for each shift of the register. The integer n represents the number of output bits generated at each shift of the register. K is an integer known as constraint length, which represents the number of k bit stages present in the encoding shift register.

Each possible combination of shift registers together forms a possible state of the encoder. For a code of constraint length K, there exist 2K-1 possible states. Since convolutional codes are processed sequentially, the encoding process can start producing encoded bits as soon as a few bits have been processed and then carry on producing bits for as long as required. Similarly, the decoding process can start as soon as a few bits have been received. In other words, this means is that it is not necessary to wait for the entire data to be received before decoding is started. This makes it ideal in situations where the data to be transmitted is very long and possibly even endless, e.g.: phone conversations.

2.2 Automatic Repeat Request

Automatic Repeat request or ARQ is a method in which the receiver sends back a positive acknowledgement if no errors are detected in the received message. In order to do this, the transmitter sends a Cyclic Redundancy Check or CRC along with the message. The CRC check bits are calculated based on the data to be

transmitted. At the receiver, the CRC is calculated again using the received bits. If the calculated CRC bits match those received, the data received is considered accurate and an acknowledgement is sent back to the transmitter. The sender waits for this acknowledgement. If it does not receive an acknowledgement (ACK) within a predefined time, or if it receives a negative acknowledgement (NAK), it retransmits the message .This retransmission is done either until it receives an ACK or until it exceeds a specified number of retransmissions.

This method has a number of drawbacks. Firstly, transmission of a whole message takes much longer as the sender has to keep waiting for acknowledgements from the receiver. Secondly, due to this delay, it is not possible to have practical, real-time, two-way communications. There are a few simple variations to the standard Stop-and-Wait ARQ such as Go-back-N ARQ, selective repeat ARQ. The huge drawback however, is that the ARQ method may require a large number of retransmissions to get the correct packet, especially if the medium is noisy. Hence the delay in getting messages across maybe excessive.

2.3 Viterbi Coding

Viterbi algorithm is the best error correction method used currently in communication systems. It is trade-off between complexity of hardware and power consumption. The Viterbi Algorithm (VA) was first proposed as a solution to the decoding of convolutional codes by Andrew J. Viterbi in 1967. Decoding Mechanism:

There are two main mechanisms, by which Viterbi decoding may be carried out

namely, the Register Exchange mechanism and the Trace back mechanism. Register exchange mechanisms, store the partially decoded output sequence along the path. The advantage of this approach is that it eliminates the need for trace back and hence reduces latency. However at each stage, the contents of each register needs to be copied to the next stage. This makes the hardware complex and more energy consuming than the trace back mechanism. Trace back mechanisms use a single bit to indicate whether the survivor branch came from the upper or lower path. This information is used to trace back the surviving path from the final state to the initial state. This path can then be used to obtain the decoded sequence. Trace back mechanisms prove to be less energy consuming and will hence be the approach followed in this project.

Chapter 3

Hamming Codes

Richard Hamming, a colleague of Shannon's at Bell Laboratories, found a need for error correction in his work on computers. Parity checking was already being used to detect errors in the calculations of the relay-based computers of the day, and Hamming realized that a more sophisticated pattern of parity checking allowed the correction of single errors along with the detection of double errors.

The codes that Hamming devised, the single-error-correcting binary Hamming codes and their single-error-correcting, double-error-detecting extended versions marked the beginning of coding theory. These codes remain important to this day, for theoretical and practical reasons as well as historical.

3.1 Fundamentals of Hamming codes

Binary block code: A collection of code words. Each code word is a fixed length pattern of 0s and 1s.

- k: The number of message bits in each code word.

- r: The number of check bits in each code word.
- n: The block length of the code. n = k + r
- Information Rate of a code: The ratio k /n
- Hamming distance between code words: The number of corresponding positions in which the code words differ.
- Minimum distance of a code: The minimum of the Hamming distances between all pairs of code words.
- Hamming weight of a vector: Number of non-zero components of the vector.
- Result: A linear code can correct all error patterns of Hamming weight t or less if and only if it has a Minimum distance at least 2t + 1. It can detect all error patterns of weight d or less if and only if it has a minimum distance at least d+1.

For any r, construct a binary $rx2r-1$ matrix H such that each nonzero binary r-tuple occurs exactly once as a column of H. Any code with such a check matrix H is a binary Hamming code of redundancy r, denoted Hamr(2). Thus the [7; 4] code is a Hamming code Ham3(2). Each binary Hamming code has minimum weight and distance 3, since as before there are no columns 0 and no pair of identical columns. Different codes and check matrices may be selected to suit different purposes.

3.2 Hamming Encoding

In a code where each code word contains several message bits and several check bits, each check bit must be some function of the message bits. In the Hamming code each check bit is taken to be a mod 2 sum of a subset of the message bits. Assume that the rate of the code is $4/7$, so that for every four information symbols transmitted, there are three check symbols introduced in the code word. Call these the parity check symbols. It is customary to index the bits from left to right beginning with 0.

Let $c = (c0, c1, c6)$ denote the vector representing a seven bit code word, with the first four bits being information bits. If c0, c1, c2 and c3 are the information symbols, let us define the Check symbols c4, c5, c6 as follows:

$c4 = c0 + c1 + c2(mod2)$

$c5 = c0 + c1 + c3(mod2)$

$c6 = c0 + c1 + c3(mod2)$

Thus, if the message bits are 1010, then the code word is 1010011. After the message sequence is encoded into the code word c, the code word is transmitted across the noisy channel. The channel adds to the code word, an error pattern e = (e0, e1, e6) to yield the received pattern r = cue. Thus, for example, if the bit with index 4 of the code word above is garbled by the channel, the error pattern is 0000100, and the received pattern is 1010111. The decoder then has to estimate the original message from the garbled code word.

Let us rewrite the equations above as

$c0 + c1 + c2 + c4 = 0(mod2)$

$c0+c1+c3+c5=0(mod2)$

$c0+c2+c3+c6=0(mod2)$

Every code word satisfies these equations. Therefore if $c=(c0,c1,c2,c3,c4,c5,c6)$ is a code word, then the matrix equation below is just a restatement of the three equations above.

The first matrix on the left hand side is called the parity check matrix H. Thus every code word c satisfies the equation Therefore another way of describing the code is by specifying its parity check matrix H. Note that the seven columns of the parity check matrix are the seven distinct non zero combinations of three bits.

3.3 Syndrome Decoding

The syndrome, reveals the pattern of parity check failures on the received pattern.Given the received pattern r, the decoder must eventually decide what the transmitted code word was. If the decoder is able to find the error pattern e, then the code word is c = r + e. To estimate e, it first forms the product HrT = HcT + HeT = HeT . This product is called the syndrome, and reveals the pattern of parity check failures on the received pattern. Decoding in the Hamming code then consists of the following three steps:

1. Form the syndrome HrT from the received vector r.

2. If the syndrome is the all zero vector, assume no errors have occurred. If it is not, then find out which column of H the syndrome matches. If the

column index is i, then the estimated error pattern ł is the vector with a 1 in the ith position and zeros everywhere else.

3. Form c = r +ł as the decoders estimate of the transmitted code word.

The parity check matrix of a code provides us with a means of determining a lower bound on the minimum distance of the code, without either having to examine all pairs of code words or looking at syndromes. The Minimum Distance of a Code the last result in the table displayed indicates that since this Hamming code can correct all single errors, the minimum distance of the code must be at least three.

If u and v are vectors, then the distance between them is the Hamming weight of u + v (where + is a bitwise operation), which is also a code word by the linearity condition. Thus for a linear code, the minimum distance is the minimum Hamming weight of a non-zero code word.

A code is normally characterized by three parameters. These are the length n, the number of information symbols k, and the minimum distance d. A code with these parameters is called a (n, k, d) code.

Binary single error correcting Hamming codes can be defined for any length 2m 1. The parity check matrix of such a code will have m rows and 2m 1 columns consisting of all non-zero binary combinations of m bits. By analogy with the codes of length 7 described above, we can deduce that all these codes have minimum distance three. Thus one can define Hamming codes with parameters (15, 11, 3), (31, 26, 3), (63, 57, 3) and so on. One can check that the information

rate goes up as the length of the code increases, for the same error correcting capability.

The Hamming codes have the beautiful geometric property that the spheres containing all vectors at distance 1 from code words, will be non-intersecting. In fact, the spheres cover the whole space, that is, they together contain all the vectors in the space. This generally does not happen for all codes. The Hamming codes happen to belong to a very exclusive class of codes called perfect codes.

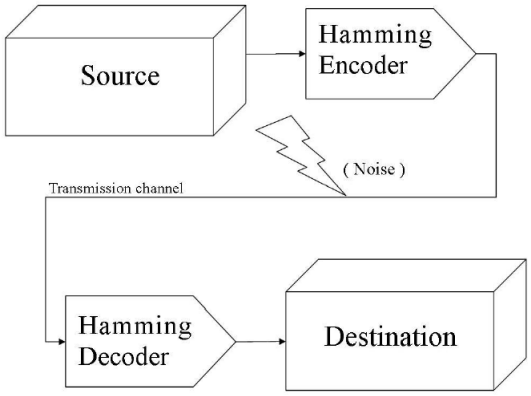

Figure 3.1: Basic Hamming System

Chapter 4

Convolutional Codes

Convolutional codes are commonly described using two parameters: the code rate and the constraint length. The code rate, k/n, is expressed as a ratio of the number of bits into the convolutional encoder (k) to the number of channel symbols output by the convolutional encoder (n) in a given encoder cycle. The constraint length parameter, K, denotes the ”length” of the convolutional encoder, i.e. how many k-bit stages are available to feed the combinatorial logic that produces the output symbols. Closely related to K is the parameter m, which indicates how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the convolutional encoder. The m parameter can be thought of as the memory length of the encoder.

Convolutional codes are widely used as channel codes in practical communication systems for error correction. The encoded bits depend on the current k input bits and a few past input bits. The main decoding strategy for convolutional codes is based on the widely used Viterbi algorithm. As a result of the wide acceptance of convolutional codes, there have been several approaches to modify and extend

this basic coding scheme. Trellis coded modulation (TCM) and turbo codes are two such examples. In TCM, redundancy is added by combining coding and modulation into a single operation. This is achieved without any reduction in data rate or expansion in bandwidth as required by only error correcting coding schemes.

4.1 Convolutional Encoder

A simple convolutional encoder is shown in figure 4.1. The information bits are fed in small groups of k-bits at a time to a shift register. The output encoded bits are obtained by modulo-2 addition (EXCLUSIVE-OR operation) of the input information bits and the contents of the shift registers which are a few previous information bits.

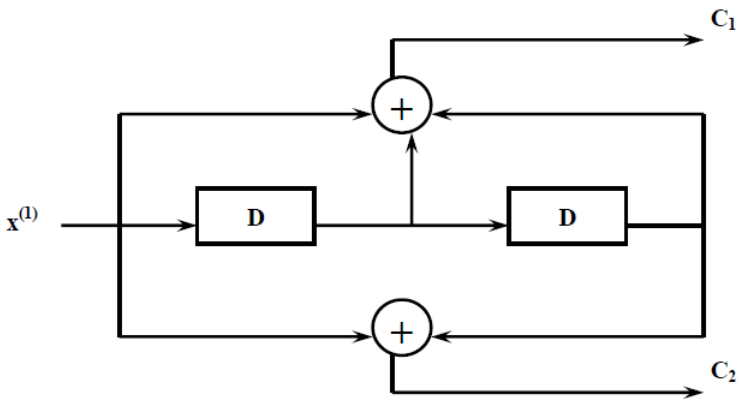

Figure 4.1: Convolution Encoder of code rate $1/2$

If the encoder generates a group of n encoded bits per group of k information bits, the code rate R is commonly defined as R = k/n. In Fig. 6.35.1, k = 1 and n

= 2. The number, K of elements in the shift register which decides for how many code words one information bit will affect the encoder output, is known as the constraint length of the code. For the present example, K = 3. The shift register of the encoder is initialized to all-zero-state before encoding operation starts. It is easy to verify that encoded sequence is 00 11 10 00 01 .for an input message sequence of 01011.

The operation of a convolutional encoder can be explained in several but equivalent ways such as, by a) state diagram representation, b) tree diagram representation and c) trellis diagram representation.

4.1.1 state Diagram Representation

A convolutional encoder may be defined as a finite state machine. Contents of the rightmost (K-1) shift register stages define the states of the encoder. So, the encoder has four states. The transition of an encoder from one state to another, as caused by input bits, is depicted in the state diagram.Figure **??** shows the state diagram of the encoder. A new input bit causes a transition from one state to another. The path information between the states, denoted as b/c1c2, represents input information bit b and the corresponding output bits (c1c2). Again, it is not difficult to verify from the state diagram that an input information sequence b = (1011) generates an encoded sequence c = (11, 10, 00, 01).

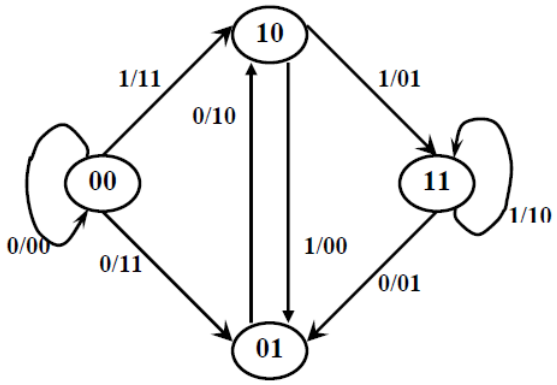

Figure 4.2: State Diagaram

4.1.2 Tree Diagram Representation

The tree diagram representation shows all possible information and encoded sequences for the convolutional encoder. Figure 4.3 Shows the tree diagram for the encoder in Figure 4.1. The encoded bits are labelled on the branches of the tree. Given an input sequence, the encoded sequence can be directly read from the tree. As an example, an input sequence (1011) results in the encoded sequence (11, 10, 00, and 01).

4.1.3 Trellis Diagram Representation

The trellis diagram of a convolutional code is obtained from its state diagram. All state transitions at each time step are explicitly shown in the diagram to retain the time dimension, as is present in the corresponding tree diagram. Usually, supporting descriptions on state transitions, corresponding input and output bits etc.

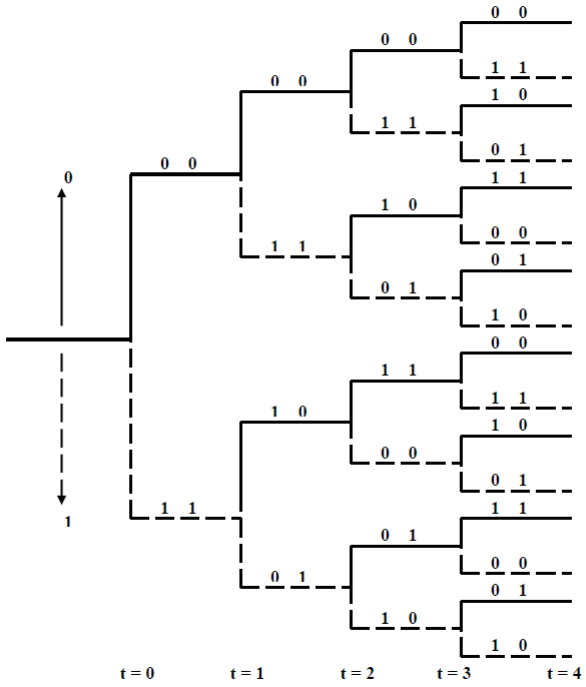

Figure 4.3: Tree Diagram

are labelled in the trellis diagram. It is interesting to note that the trellis diagram, which describes the operation of the encoder, is very convenient for describing the behaviour of the corresponding decoder, especially when the famous Viterbi Algorithm (VA) is followed. Figure **??** shows the trellis diagram for the encoder.

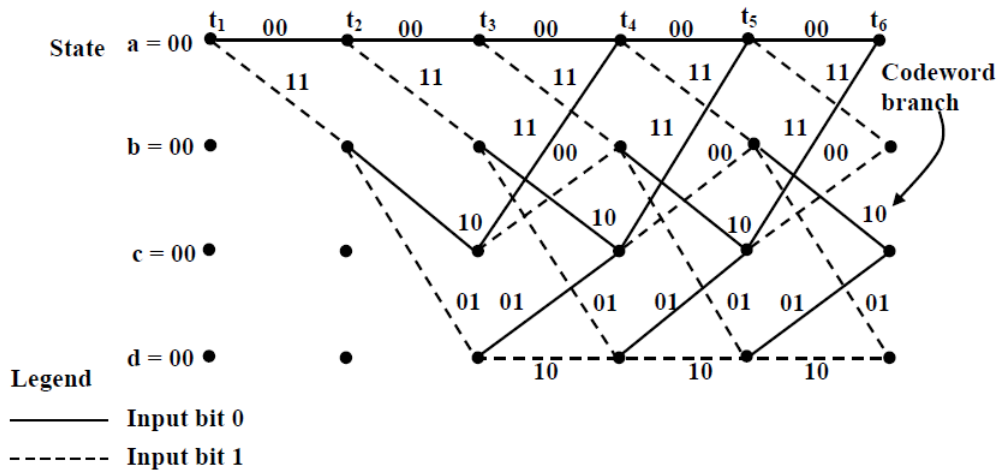

Figure 4.4: Trellis Structure for Encoder

4.2 Viterbi Decoding

4.2.1 Decision and Soft-Decision Decoding

Hard-decision and soft-decision decoding are based on the type of quantization used on the received bits. Hard-decision decoding uses 1-bit quantization on the received samples. Soft-decision decoding uses multi-bit quantization (e.g. 3 bits/sample) on the received sample values.

4.2.2 Hard-Decision Viterbi Algorithm

The Viterbi Algorithm (VA) finds a maximum likelihood (ML) estimate of a transmitted code sequence c from the corresponding received sequence r by maximizing the probability p(rice) that sequence r is received conditioned on the esti-

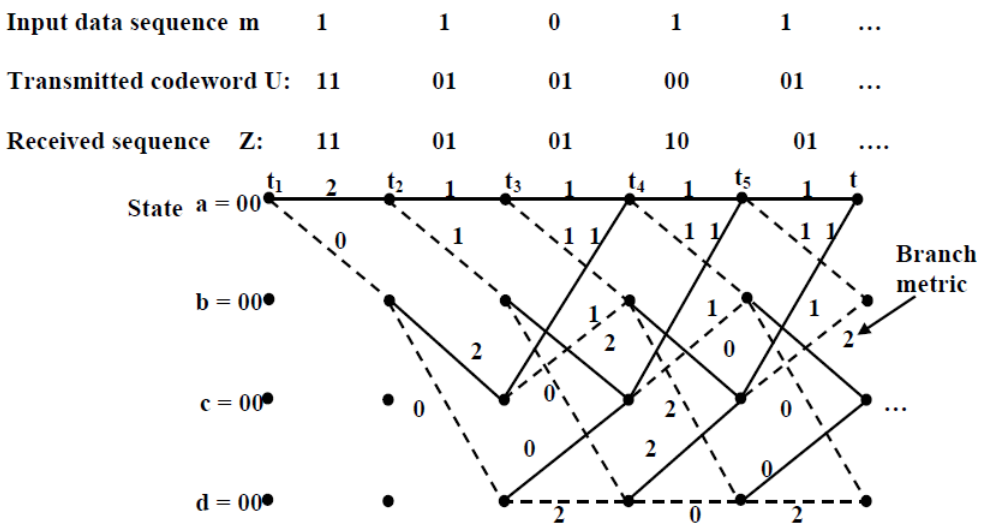

Figure 4.5: Trellis Structure for Decoder

mated code sequence c. Sequence c must be a valid coded sequence.

The Viterbi algorithm utilizes the trellis diagram to compute the path metrics. The channel is assumed to be memory less, i.e. the noise sample affecting a received bit is independent from the noise sample affecting the other bits. The decoding operation starts from state 00, i.e. with the assumption that the initial state of the encoder is 00. With receipt of one noisy code word, the decoding operation progresses by one step deeper into the trellis diagram. The branches, associated with a state of the trellis tell us about the corresponding code words that the encoder may generate starting from this state. Hence, upon receipt of a code word, it is possible to note the branch metric of each branch by determining the Hamming distance of the received code word from the valid code word associated

with that branch. Path metric of all branches, associated with all the states are calculated similarly.

Now, at each depth of the trellis, each state also carries some accumulated path metric, which is the addition of metrics of all branches that construct the most likely path to that state. As an example, the trellis diagram of the code shown has four states and each state has two incoming and two outgoing branches. At any depth of the trellis, each state can be reached through two paths from the previous stage and as per the VA, the path with lower accumulated path metric is chosen. In the process, the accumulated path metric is updated by adding the metric of the incoming branch with the accumulated path metric of the state from where the branch originated. No decision about a received code word is taken from such operations and the decoding decision is deliberately delayed to reduce the possibility of erroneous decision.

The basic operations which are carried out as per the hard-decision Viterbi Algorithm after receiving one code word are summarized below:

1. All the branch metrics of all the states are determined;

2. Accumulated metrics of all the paths (two in our example code) leading to a state are calculated taking into consideration the accumulated path metrics of the states from where the most recent branches emerged;

3. Only one of the paths, entering into a state, which has minimum accumulated path metric is chosen as the survivor path for the state (or, equivalently node);

4. So, at the end of this process, each state has one survivor path. The history of a survivor path is also maintained by the node appropriately (e.g. by storing the code words or the information bits which are associated with the branches making the path);

5. Steps 1) to 4) are repeated and decoding decision is delayed till sufficient number of code words has been received. Typically, the delay in decision making = Lx k code words where L is an integer. For the code the decision delay of 5x3 = 15 code words may be sufficient for most occasions. This means, we decide about the first received code word after receiving the 16th code word. The decision strategy is simple. Upon receiving the 16th code word and carrying out steps 1) to 4), we compare the accumulated path metrics of all the states (four in our example) and chose the state with minimum overall accumulated path metric as the winning node for the first code word. Then we trace back the history of the path associated with this winning node to identify the code word tagged to the first branch of the path and declare this code word as the most likely transmitted first code word.

The above procedure is repeated for each received code word hereafter. Thus, the decision for a code word is delayed but once the decision process starts, we decide once for every received code word. For most practical applications, including delay-sensitive digital speech coding and transmission, a decision delay of Lx k code words is acceptable.

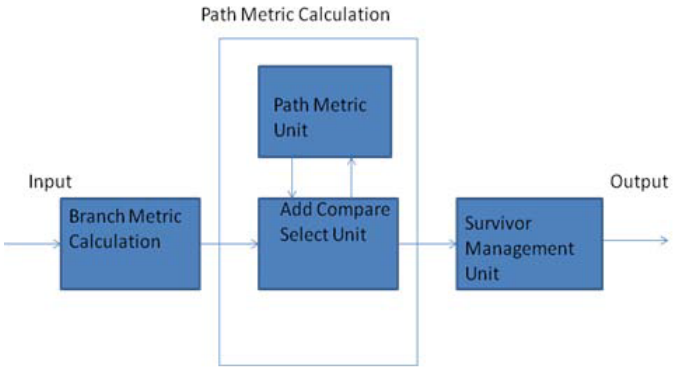

Figure 4.6: Viterbi Decoder Block Diagram

4.2.3 Soft Decision Viterbi Algorithm

In soft-decision decoding, the demodulator does not assign a 0 or a 1 to each received bit but uses multi-bit quantized values. The soft-decision Viterbi algorithm is very similar to its hard-decision algorithm except that squared Euclidean distance is used in the branch metrics instead of simpler Hamming distance. However, the performance of a soft-decision VA is much more impressive compared to its HDD (Hard Decision Decoding) counterpart. The computational requirement of a Viterbi decoder grows exponentially as a function of the constraint length and hence it is usually limited in practice to constraint lengths of K = 9.

Chapter 5

Results and Implementation

5.1 Implementation of Coding Techniques in simulink and on FPGA

5.1.1 Hamming Codes in Simulink

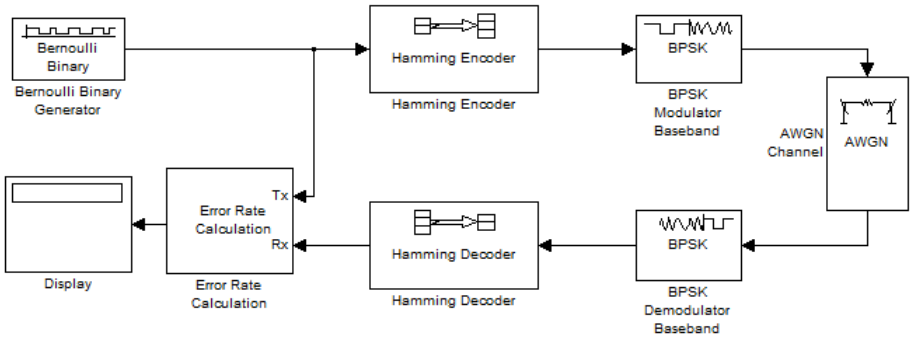

Figure 5.1: Hamming System in Simulink

5.1.2 Hamming Codes on FPGA

Hamming Encoder Module-verilog

```
module hamm_enc(Enc_seq,Org_seq,reset);//encoder module
parameter n=11,k=7;//encoded and input data equence lengths
output [n-1:0] Enc_seq;//encoded sequence
input [k-1:0] Org_seq;//input sequence
input reset;
reg [n-1:0] Enc_seq;
integer i,j;
always @(Org_seq or reset)
begin
if(reset)
Enc_seq = 0;//encoded sequence is 0 when reset is true
else
begin
i=0; j=0;
while((i<n) || (j<k))
begin
while(i==0 || i==1 || i==3 || i==7)
begin
Enc_seq[i] = 0;//placing zeros in parity bit positions in encoded sequence
i=i+1;
end
```

```
Enc_seq[i] = Org_seq[j];//plaing input data bits in data bit positions in encoded
i=i+1;
j=j+1;
end
if(^(Enc_seq & 11'b101_0101_0101))//claculating the values for parity bits of enco
Enc_seq[0] = ~Enc_seq[0];
if(^(Enc_seq & 11'b110_0110_0110))
Enc_seq[1] = ~Enc_seq[1];
if(^(Enc_seq & 11'b000_0111_1000))
Enc_seq[3] = ~Enc_seq[3];
if(^(Enc_seq & 11'b111_1000_0000))
Enc_seq[7] = ~Enc_seq[7];
end
end
endmodule
```

Hamming Decoder Module

```
module hamm_dec(Org_seq,Enc_seq,res);//decoding module for hamming codes
parameter n=11,k=7;//input sequence and encoded sequence length
output [k-1:0] Org_seq;//declare input nad output variables
input [n-1:0] Enc_seq;
input res;
reg [k-1:0] Org_seq;
reg r1,r2,r4,r8;
```

```
reg [3:0] r;
reg [n-1:0] Error;
integer i,j;

always @(Enc_seq or res)//if encoded sequence or reset is prensent on input line t

begin
if(res)//if reset is true
Org_seq=0;//clear the input varible
else
begin
r1 = ^(Enc_seq & 11'b101_0101_0101);//calculating the recieved sequence to find th
r2 = ^(Enc_seq & 11'b110_0110_0110);
r4 = ^(Enc_seq & 11'b000_0111_1000);
r8 = ^(Enc_seq & 11'b111_1000_0000);
r = {r8,r4,r2,r1};//error bit
Error = Enc_seq;
Error[r-1] = ~Error[r-1];//correcting the error
i=0; j=0;
while((i<n) || (j<k))//constructing the  data bits recieved
begin
while(i==0 || i==1 || i==3 || i==7)
i=i+1;
Org_seq[j]=Error[i];
i=i+1;
```

```
j=j+1;
end
end
end
endmodule
```

Main Module

```
module main(i,o,reset,outp);//main module for invoking encoder and decoder
parameter n=11,k=7;//declaring variables for encoded sequence and original data se
input [k-1:0] i;
input reset;
output [k-1:0] o;
output [n-1:0] outp;//variable to display encoded sequence
wire [n-1:0]ed;//for returning encoded sequence from encoder to decoder
hamm_enc h1(ed,i,reset);//invoking encoder module
assign outp=ed;//encoded sequence
hamm_dec h2(o,ed,reset);//invoking decoder module
endmodule
```

Test Bench Module

```
module main_tb;
```

```
// Inputs
reg [6:0] i;
reg reset;

// Outputs
wire [6:0] o;
wire [10:0] outp;



// Instantiate the Unit Under Test (UUT)
main uut (.o(o),.outp(outp),.i(i), .reset(reset));

initial begin
// Initialize Inputs
i= 7'b 1001101;
reset = 0;
#10 reset = 1; // Wait 100 ns for global reset to finish
#100;

// Add stimulus here

end

endmodule
```

5.1.3 Simulation Results of Hamming Codes

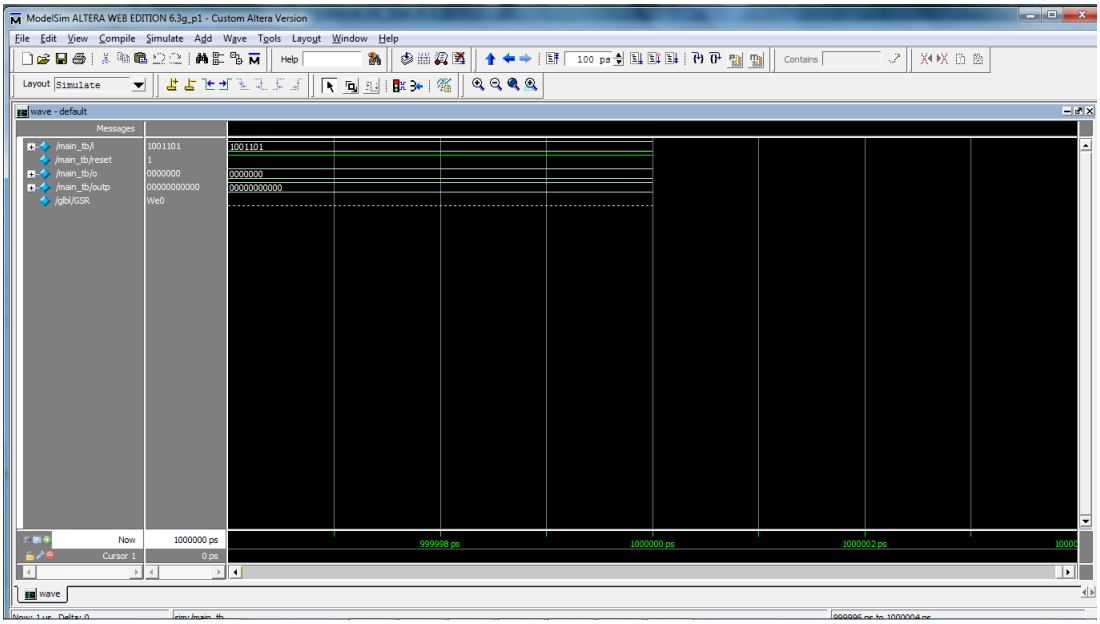

Figure 5.2: Hamming Codes Simulation Results

Output on FPGA

Mapping Report

```
Release 9.1i Map J.30

Xilinx Mapping Report File for Design 'main'


Design Information

------------------
```

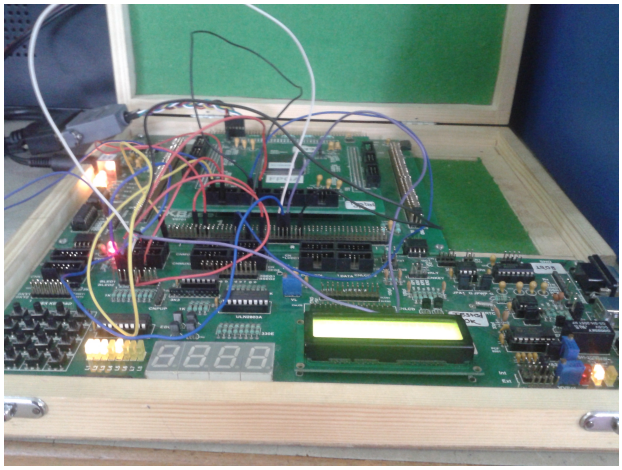

Figure 5.3: Hamming Codes output on FPGA

```
Command Line   : C:\Xilinx91i\bin\nt\map.exe -ise
C:/Xilinx91i/hamming/hamming.ise -intstyle ise -p xc3s400-pq208-5 -cm area -pr b
-k 4 -c 100 -o main_map.ncd main.ngd main.pcf
Target Device  : xc3s400
Target Package : pq208
Target Speed   : -5
Mapper Version : spartan3 -- $Revision: 1.36 $
Mapped Date    : Thu Apr 23 16:39:21 2015

Design Summary
--------------
Number of errors:      0
Number of warnings:    4
Logic Utilization:
  Number of 4 input LUTs:              12 out of   7,168    1%
```

```
Logic Distribution:
  Number of occupied Slices:                              7 out of   3,584    1%
    Number of Slices containing only related logic:       7 out of       7  100%
    Number of Slices containing unrelated logic:          0 out of       7    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:             14 out of   7,168    1%
  Number used as logic:                12
  Number used as a route-thru:          2
  Number of bonded IOBs:               15 out of     141   10%

Total equivalent gate count for design:  78
Additional JTAG gate count for IOBs:  720
Peak Memory Usage:  152 MB
Total REAL time to MAP completion:  2 secs
Total CPU time to MAP completion:   1 secs
```

Static Timing Report

```
--------------------------------------------------------------------------------
Release 9.1i Trace
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.

C:\Xilinx91i\bin\nt\trce.exe -ise C:/Xilinx91i/hamming/hamming.ise -intstyle
```

```
ise -e 3 -s 5 -xml main main.ncd -o main.twr main.pcf -ucf main.ucf


Design file:              main.ncd
Physical constraint file: main.pcf
Device,package,speed:     xc3s400,pq208,-5 (PRODUCTION 1.39 2006-10-19)
Report level:             error report

Environment Variable      Effect
--------------------      ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------


INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
   a 50 Ohm transmission line loading model.  For the details of this model,
   and for more information on accounting for different loading conditions,
   please see the device datasheet.



Data Sheet report:
------------------
```

All values displayed in nanoseconds (ns)

Pad to Pad

Source Pad	Destination Pad	Delay
od<0>	rd<0>	9.630
od<1>	rd<1>	7.079
od<2>	rd<2>	8.017
od<3>	rd<0>	10.041
od<3>	rd<3>	7.368
od<4>	rd<0>	9.228
od<4>	rd<4>	7.764
od<5>	rd<5>	7.462
od<6>	rd<0>	9.140
od<6>	rd<6>	7.648
reset	rd<0>	10.224
reset	rd<1>	7.989
reset	rd<2>	8.623
reset	rd<3>	7.978
reset	rd<4>	8.230
reset	rd<5>	8.230
reset	rd<6>	9.260

```
Analysis completed Thu Apr 23 16:39:29 2015
--------------------------------------------------------------------------------


Trace Settings:
-------------------------
Trace Settings

Peak Memory Usage: 97 MB
```

Synthesis Report

```
Release 9.1i - xst J.30
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to ./xst/projnav.tmp
CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 0.00 s

--> Parameter xsthdpdir set to ./xst
CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 0.00 s

--> Reading design: main.prj


=========================================================================
=========================================================================
HDL Synthesis Report
```

```
Macro Statistics
# Xors                                                  : 11
 1-bit xor2                                            : 2
 1-bit xor3                                            : 7
 1-bit xor4                                            : 1
 1-bit xor6                                            : 1

=========================================================================

Advanced HDL Synthesis Report

Macro Statistics
# Xors                                                  : 11
 1-bit xor2                                            : 2
 1-bit xor3                                            : 7
 1-bit xor4                                            : 1
 1-bit xor6                                            : 1

=========================================================================
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : main.ngr
```

```
Top Level Output File Name         : main
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 15

Cell Usage :
# BELS                             : 14
#      LUT2                        : 7
#      LUT3                        : 1
#      LUT4                        : 4
#      MUXF5                       : 2
# IO Buffers                       : 15
#      IBUF                        : 8
#      OBUF                        : 7
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s400pq208-5

 Number of Slices:                        8  out of   3584     0%
```

```
 Number of 4 input LUTs:                12  out of   7168     0%
 Number of IOs:                         15
 Number of bonded IOBs:                 15  out of    141    10%


---------------------------
Partition Resource Summary:
---------------------------


  No Partitions were found in this design.


-------------------------


Clock Information:
------------------
No clock signals found in this design


Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design


Timing Summary:
---------------
Speed Grade: -5


   Minimum period: No path found
```

```
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 12.431ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 37 / 7
-------------------------------------------------------------------------
Delay:               12.431ns (Levels of Logic = 7)
  Source:            reset (PAD)
  Destination:       rd<0> (PAD)

  Data Path: reset to rd<0>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             11   0.715   1.267  reset_IBUF (reset_IBUF)
     LUT2:I0->O             4   0.479   1.074  e1/out_2_mux00001 (ed<2>)
     LUT4:I0->O             1   0.479   0.000  d2/Mxor__old_r1_4_xo<4>1_SW01 (N180)
     MUXF5:I0->O            1   0.314   0.851  d2/Mxor__old_r1_4_xo<4>1_SW0_f5 (N15
     LUT4:I1->O             1   0.479   0.704  d2/Mxor__old_r1_4_xo<4>1 (d2/_old_r1_
```

```
     LUT4:I3->O            1   0.479   0.681  d2/out_0_mux0001 (rd_0_OBUF)
     OBUF:I->O                 4.909          rd_0_OBUF (rd<0>)
    ----------------------------------------
    Total                     12.431ns (7.854ns logic, 4.577ns route)
                                       (63.2% logic, 36.8% route)


=========================================================================
CPU : 3.52 / 3.69 s | Elapsed : 4.00 / 4.00 s

-->

Total memory usage is 143548 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    8 (   0 filtered)
Number of infos    :    4 (   0 filtered)
```

5.1.4 Convolutional Codes in Simulink

5.1.5 Convolutional Coding on FPGA

D Flip Flop Module

```
module dff(d,q,clock,res);//d flip flop module
```

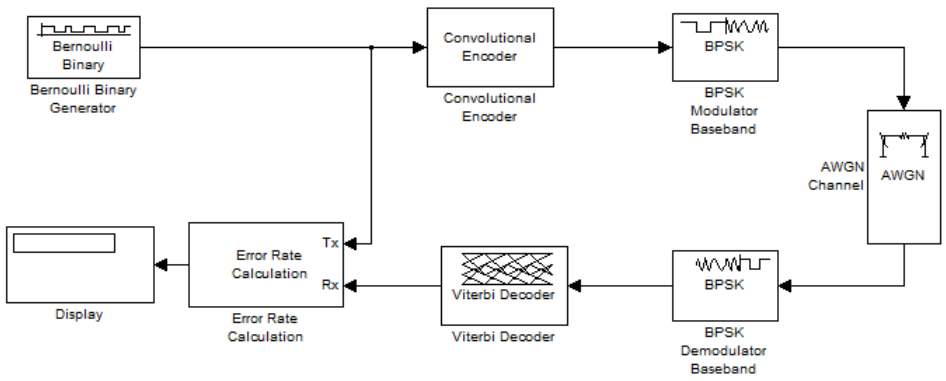

Figure 5.4: Convolutional Coding System in Simulink

```
output q; input d;
input [1:0]clock;input res;

parameter CARDINALITY = 1; reg [CARDINALITY-1:0] q;
wire [CARDINALITY-1:0] d;



always @ (posedge clock[1])

begin
if (!res)
q=d;
else
q=0;
end
```

```
endmodule
```

Main Module

```
module main(X,outp,clk,reset,error);
input clk,reset;
input [1:0] X;
output error;
output [2:0] outp;
wire Y2N,Y1N,Y0N,in0,in1,in2,in3,in4,in5,in6,in7;

viterbi_encode v1(X[1],X[0],Y2N,Y1N,Y0N,clk,reset);//invoking encoder module
viterbi_distances v2(Y2N,Y1N,Y0N,clk,res,in0,in1,in2,in3,in4,in5,in6,in7);//invoki
viterbi v3(in0,in1,in2,in3,in4,in5,in6,in7,outp,clk,reset,error);//invoking decode

endmodule
```

Convolutional Encoder

```
module viterbi_encode(X2N,X1N,Y2N,Y1N,Y0N,clk,res);
input X2N,X1N,clk,res; output Y2N,Y1N,Y0N;
wire X1N_1,X1N_2,Y2N,Y1N,Y0N;
```

```
dff dff_1(X1N,X1N_1,clk,res); dff dff_2(X1N_1,X1N_2,clk,res);
assign Y2N=X1N^X2N; assign Y1N=X1N ^ X1N_2; assign Y0N=X1N_1;


endmodule
```

Compare Select Module

```
module compare_select(p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
    out0,out1,out2,out3,
    ACS0,ACS1,ACS2,ACS3);
 input [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
  output [4:0] out0,out1,out2,out3;
  output ACS0,ACS1,ACS2,ACS3;
  function [4:0] find_min_metric; input [4:0] a,b;
    begin
      if (a <= b) find_min_metric = a; else find_min_metric = b;
    end
  endfunction
  function set_control; input [4:0] a,b;
    begin
      if (a <= b) set_control = 0; else set_control = 1;
    end
  endfunction
assign out0 = find_min_metric(p0_0,p2_0);
```

```
assign out1 = find_min_metric(p0_1,p2_1);
assign out2 = find_min_metric(p1_2,p3_2);
assign out3 = find_min_metric(p1_3,p3_3);
assign ACS0 = set_control (p0_0,p2_0);
assign ACS1 = set_control (p0_1,p2_1);
assign ACS2 = set_control (p1_2,p3_2);
assign ACS3 = set_control (p1_3,p3_3);

endmodule
```

Compute Metric Module

```
module compute_metric(m_out0,m_out1,m_out2,m_out3,
    s0,s1,s2,s3,p0_0,p2_0,
    p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
    error );
 input [4:0] m_out0,m_out1,m_out2,m_out3;
  input [2:0] s0,s1,s2,s3;
  output [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
  output error;
  assign
    p0_0 = m_out0 + s0,
    p2_0 = m_out2 + s2,
    p0_1 = m_out0 + s2,
```

```
    p2_1 = m_out2 + s0,
    p1_2 = m_out1 + s1,
    p3_2 = m_out3 + s3,
    p1_3 = m_out1 + s3,
    p3_3 = m_out3 + s1;
  function is_error; input x1,x2,x3,x4,x5,x6,x7,x8;
  begin
    if (x1||x2||x3||x4||x5||x6||x7||x8) is_error = 1;
    else is_error = 0;
  end
  endfunction
  assign error = is_error(p0_0[4],p2_0[4],p0_1[4],p2_1[4],
    p1_2[4],p3_2[4],p1_3[4],p3_3[4]);

endmodule
```

Metric Module

```
module metric(m_in0,m_in1,m_in2,m_in3,
    m_out0,m_out1,m_out2,m_out3,
    clk,reset );
input [4:0] m_in0,m_in1,m_in2,m_in3;
output [4:0] m_out0,m_out1,m_out2,m_out3;
input clk,reset;
```

```
  dff #(5) metric3(m_in3, m_out3, clk, reset);
  dff #(5) metric2(m_in2, m_out2, clk, reset);
  dff #(5) metric1(m_in1, m_out1, clk, reset);
  dff #(5) metric0(m_in0, m_out0, clk, reset);

endmodule
```

Output Decision Module

```
module output_decision(p0,p1,p2,p3,control,out );
input [2:0] p0,p1,p2,p3; input [1:0] control; output [2:0] out;
  function [2:0] decide;
  input [2:0] p0,p1,p2,p3; input [1:0] control;
  begin
    if(control == 0) decide = p0;
    else if(control == 1) decide = p1;
    else if(control == 2) decide = p2;
    else decide = p3;
    end
  endfunction
assign out = decide(p0,p1,p2,p3,control);

endmodule
```

Path Module

```
module path(in,out,clk,reset,ACS0,ACS1,ACS2,ACS3 );
input [11:0] in; output [11:0] out;
input clk,reset,ACS0,ACS1,ACS2,ACS3; wire [11:0] p_in;
dff #(12) path0(p_in,out,clk,reset);
  function [2:0] shift_path; input [2:0] a,b; input control;
    begin
      if (control == 0) shift_path = a; else shift_path = b;
    end
  endfunction
assign p_in[11:9] = shift_path(in[11:9],in[5:3],ACS0);
assign p_in[ 8:6] = shift_path(in[11:9],in[5:3],ACS1);
assign p_in[ 5:3] = shift_path(in[8: 6],in[2:0],ACS2);
assign p_in[ 2:0] = shift_path(in[8: 6],in[2:0],ACS3);


endmodule
```

Pathin Module

```
module pathin(sout0,sout1,sout2,sout3,
```

```
    ACS0,ACS1,ACS2,ACS3,
    path0,clk,reset );

 input sout0,sout1,sout2,sout3,ACS0,ACS1,ACS2,ACS3;
  input clk,reset; output [11:0] path0;
  wire [2:0] sig0,sig1,sig2,sig3; wire [11:0] path_in;
  dff #(12) firstpath(path_in,path0,clk,reset);
  function [2:0] subset0; input sout0;
    begin
      if(sout0 == 0) subset0 = 0; else subset0 = 4;
    end
  endfunction
  function [2:0] subset1; input sout1;
    begin
      if(sout1 == 0) subset1 = 1; else subset1 = 5;
    end
  endfunction
  function [2:0] subset2; input sout2;
    begin
      if(sout2 == 0) subset2 = 2; else subset2 = 6;
    end
  endfunction
  function [2:0] subset3; input sout3;
    begin
      if(sout3 == 0) subset3 = 3; else subset3 = 7;
```

```
    end
  endfunction
  function [2:0] find_path; input [2:0] a,b; input control;
    begin
      if(control==0) find_path = a; else find_path = b;
    end
  endfunction
assign sig0 = subset0(sout0);
assign sig1 = subset1(sout1);
assign sig2 = subset2(sout2);
assign sig3 = subset3(sout3);
assign path_in[11:9] = find_path(sig0,sig2,ACS0);
assign path_in[ 8:6] = find_path(sig2,sig0,ACS1);
assign path_in[ 5:3] = find_path(sig1,sig3,ACS2);
assign path_in[ 2:0] = find_path(sig3,sig1,ACS3);
endmodule
```

Reduce Module

```
module reduce(in0,in1,in2,in3,
    m_in0,m_in1,m_in2,m_in3,
    control );
input [4:0] in0,in1,in2,in3;
  output [4:0] m_in0,m_in1,m_in2,m_in3;
```

```
  output [1:0] control; wire [4:0] smallest;
  function [4:0] find_smallest;
    input [4:0] in0,in1,in2,in3; reg [4:0] a,b;
      begin
        if(in0 <= in1) a = in0; else a = in1;
        if(in2 <= in3) b = in2; else b = in3;
        if(a <= b) find_smallest = a;
        else find_smallest = b;
      end
  endfunction
  function [1:0] smallest_no;
  input [4:0] in0,in1,in2,in3,smallest;
    begin
      if(smallest == in0) smallest_no = 0;
      else if (smallest == in1) smallest_no = 1;
      else if (smallest == in2) smallest_no = 2;
      else smallest_no = 3;
    end
  endfunction
assign smallest = find_smallest(in0,in1,in2,in3);
assign m_in0 = in0 - smallest;
assign m_in1 = in1 - smallest;
assign m_in2 = in2 - smallest;
assign m_in3 = in3 - smallest;
assign control = smallest_no(in0,in1,in2,in3,smallest);
```

```
endmodule
```

Subset Decode Module

```
module subset_decode(in0,in1,in2,in3,in4,in5,in6,in7,
    s0,s1,s2,s3,
    sout0,sout1,sout2,sout3,
    clk,reset );
input [2:0] in0,in1,in2,in3,in4,in5,in6,in7;
output [2:0] s0,s1,s2,s3;
output sout0,sout1,sout2,sout3;
input clk,reset;
wire [2:0] sub0,sub1,sub2,sub3,sub4,sub5,sub6,sub7;
  dff #(3) subout0(in0, sub0, clk, reset);
  dff #(3) subout1(in1, sub1, clk, reset);
  dff #(3) subout2(in2, sub2, clk, reset);
  dff #(3) subout3(in3, sub3, clk, reset);
  dff #(3) subout4(in4, sub4, clk, reset);
  dff #(3) subout5(in5, sub5, clk, reset);
  dff #(3) subout6(in6, sub6, clk, reset);
  dff #(3) subout7(in7, sub7, clk, reset);
  function [2:0] subset_decode; input [2:0] a,b;
    begin
```

```
      subset_decode = 0;
      if (a<=b) subset_decode = a; else subset_decode = b;
    end
  endfunction
  function set_control; input [2:0] a,b;
    begin
      if (a<=b) set_control = 0; else set_control = 1;
    end
  endfunction
assign s0 = subset_decode (sub0,sub4);
assign s1 = subset_decode (sub1,sub5);
assign s2 = subset_decode (sub2,sub6);
assign s3 = subset_decode (sub3,sub7);
assign sout0 = set_control(sub0,sub4);
assign sout1 = set_control(sub1,sub5);
assign sout2 = set_control(sub2,sub6);
assign sout3 = set_control(sub3,sub7);

endmodule
```

Viterbi Encoder

```
module viterbi(in0,in1,in2,in3,in4,in5,in6,in7,
    out,clk,reset,error);
```

```
input [2:0] in0,in1,in2,in3,in4,in5,in6,in7;
output [2:0] out; input clk,reset; output error;
wire sout0,sout1,sout2,sout3;
wire [2:0] s0,s1,s2,s3;
wire [4:0] m_in0,m_in1,m_in2,m_in3;
wire [4:0] m_out0,m_out1,m_out2,m_out3;
wire [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
wire ACS0,ACS1,ACS2,ACS3;
wire [4:0] out0,out1,out2,out3;
wire [1:0] control;
wire [2:0] p0,p1,p2,p3;
wire [11:0] path0;
  subset_decode u1(in0,in1,in2,in3,in4,in5,in6,in7,
    s0,s1,s2,s3,sout0,sout1,sout2,sout3,clk,reset);
  metric u2(m_in0,m_in1,m_in2,m_in3,m_out0,
    m_out1,m_out2,m_out3,clk,reset);
  compute_metric u3(m_out0,m_out1,m_out2,m_out3,s0,s1,s2,s3,
    p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,error);
  compare_select u4(p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
    out0,out1,out2,out3,ACS0,ACS1,ACS2,ACS3);
  reduce u5(out0,out1,out2,out3,
    m_in0,m_in1,m_in2,m_in3,control);
  pathin u6(sout0,sout1,sout2,sout3,
    ACS0,ACS1,ACS2,ACS3,path0,clk,reset);
  path_memory u7(p0,p1,p2,p3,path0,clk,reset,
```

```
    ACS0,ACS1,ACS2,ACS3);
  output_decision u8(p0,p1,p2,p3,control,out);

endmodule
```

Viterbi Distance Module

```
module viterbi_distances(Y2N,Y1N,Y0N,clk,res,in0,in1,in2,in3,in4,in5,in6,in7);
input clk,res,Y2N,Y1N,Y0N;
output in0,in1,in2,in3,in4,in5,in6,in7;
reg [2:0] J,in0,in1,in2,in3,in4,in5,in6,in7;
reg [2:0] d [7:0];
initial
begin
d[0]=3'b000;d[1]=3'b001;d[2]=3'b100;d[3]=3'b110;
d[4]=3'b111;d[5]=3'b110;d[6]=3'b100;d[7]=3'b001;
end
always @(Y2N or Y1N or Y0N)
begin
J[0]=Y0N;J[1]=Y1N;J[2]=Y2N;
J=8-J;in0=d[J];J=J+1;in1=d[J];J=J+1;in2=d[J];J=J+1;in3=d[J];
J=J+1;in4=d[J];J=J+1;in5=d[J];J=J+1;in6=d[J];J=J+1;in7=d[J];
end
```

```
endmodule
```

Test Bench Module

```
module main_tb;

// Inputs

reg [1:0] X;//inputs to which the signals to be assigned
reg clk;
reg reset;

// Outputs
wire [2:0] outp;
wire error;


// Instantiate the Unit Under Test (UUT)
main uut ( .X(X), .outp(outp), .clk(clk), .reset(reset), .error(error));//instance

        always #50 clk=~clk;
```

```
initial begin
// Initialize Inputs


clk = 0;
X=3;
#05 reset = 1;
#100 reset = 0;  // Hit reset after inputs are stable.




// Add stimulus here


end
always #500 X = X + 1;


endmodule
```

5.1.6 Simulation Results of Convolutional Codes

Output on FPGA

Mapping Report

```
Release 9.1i Map J.30
Xilinx Mapping Report File for Design 'main'
```

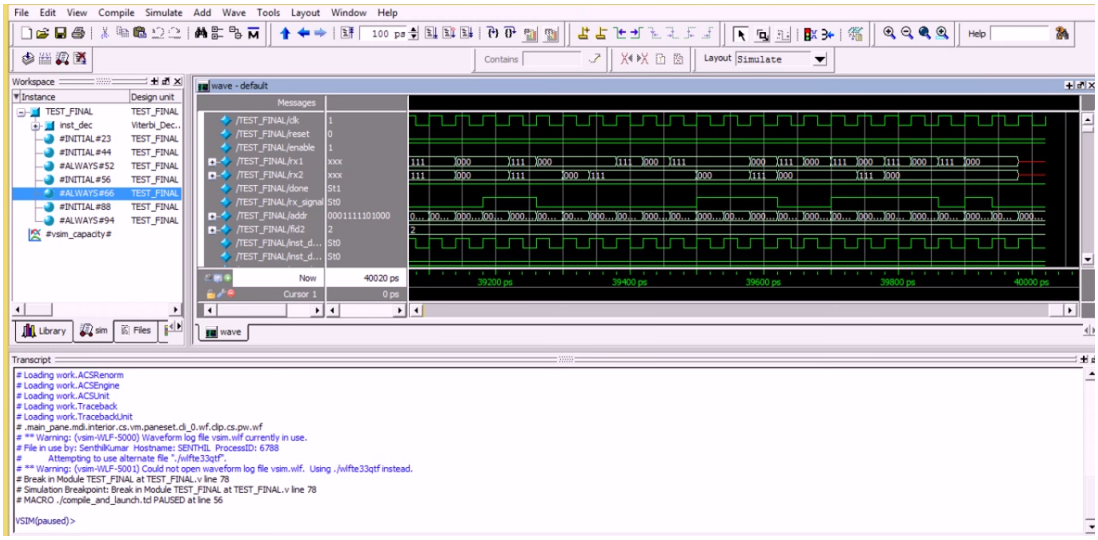

Figure 5.5: Convolutional Coding Simulation Results

```
Design Information
------------------
Command Line   : C:\Xilinx91i\bin\nt\map.exe -ise C:/Xilinx91i/fycmst/fycmst.ise
-intstyle ise -p xc3s400-pq208-5 -cm area -pr b -k 4 -c 100 -o main_map.ncd
main.ngd main.pcf
Target Device  : xc3s400
Target Package : pq208
Target Speed   : -5
Mapper Version : spartan3 -- $Revision: 1.36 $
Mapped Date    : Wed Apr 22 14:19:10 2015

Design Summary
```

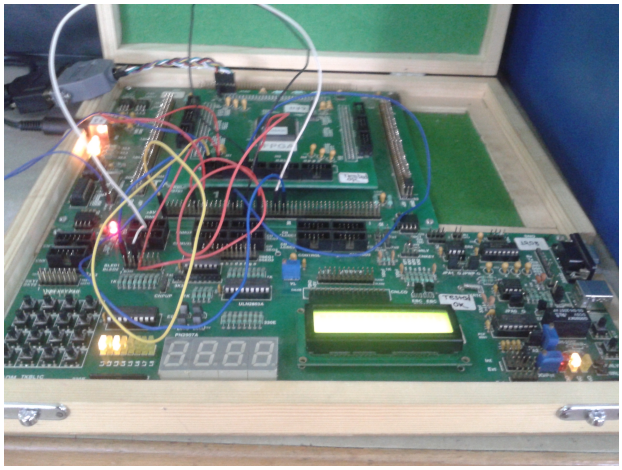

Figure 5.6: Output of Convolutional Codes on FPGA

```
--------------
Number of errors:      0
Number of warnings:    5
Logic Utilization:
  Number of Slice Flip Flops:         186 out of   7,168    2%
  Number of 4 input LUTs:             454 out of   7,168    6%
Logic Distribution:
  Number of occupied Slices:                          237 out of   3,584    6%
    Number of Slices containing only related logic:     237 out of     237  100%
    Number of Slices containing unrelated logic:          0 out of     237    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:           458 out of   7,168    6%
  Number used as logic:                454
  Number used as a route-thru:           4
  Number of bonded IOBs:                 8 out of     141    5%
```

```
    IOB Flip Flops:                    1
  Number of GCLKs:                   1 out of       8   12%

Total equivalent gate count for design:  4,385
Additional JTAG gate count for IOBs:  384
Peak Memory Usage:  156 MB
Total REAL time to MAP completion:  1 secs
Total CPU time to MAP completion:   1 secs
```

Static Timing Report

```
--------------------------------------------------------------------------------
Release 9.1i Trace
Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.

C:\Xilinx91i\bin\nt\trce.exe -ise C:/Xilinx91i/fycmst/fycmst.ise -intstyle ise
-e 3 -s 5 -xml main main.ncd -o main.twr main.pcf -ucf main.ucf

Design file:              main.ncd
Physical constraint file: main.pcf
Device,package,speed:     xc3s400,pq208,-5 (PRODUCTION 1.39 2006-10-19)
Report level:             error report

Environment Variable      Effect
```

```
--------------------      ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
   a 50 Ohm transmission line loading model.  For the details of this model,
   and for more information on accounting for different loading conditions,
   please see the device datasheet.



Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock Clk
------------+------------+------------+-----------------+--------+
            |  Setup to  |  Hold to   |                 | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+-----------------+--------+
Res         |    1.424(R)|    3.065(R)|Clk_BUFGP        |   0.000|
```

```
X<0>        |    1.617(R)|    2.732(R)|Clk_BUFGP         |   0.000|
X<1>        |    2.088(R)|    2.891(R)|Clk_BUFGP         |   0.000|
------------+------------+------------+------------------+--------+


Clock Clk to Pad
------------+------------+------------------+--------+
            | clk (edge) |                  | Clock  |
Destination |   to PAD   |Internal Clock(s) | Phase  |
------------+------------+------------------+--------+
Error       |   21.825(R)|Clk_BUFGP         |   0.000|
Out<0>      |   34.251(R)|Clk_BUFGP         |   0.000|
Out<1>      |   34.030(R)|Clk_BUFGP         |   0.000|
Out<2>      |   34.334(R)|Clk_BUFGP         |   0.000|
------------+------------+------------------+--------+


Clock to Setup on destination clock Clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
Clk            |   17.206|         |         |         |
---------------+---------+---------+---------+---------+



Analysis completed Wed Apr 22 14:19:21 2015
```

```
---------------------------------------------------------------------------


Trace Settings:

-------------------------

Trace Settings


Peak Memory Usage: 101 MB
```

Synthesis Report

```
Release 9.1i - xst J.30

Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.

--> Parameter TMPDIR set to ./xst/projnav.tmp

CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 0.00 s


--> Parameter xsthdpdir set to ./xst

CPU : 0.00 / 0.16 s | Elapsed : 0.00 / 0.00 s


--> Reading design: main.prj

=========================================================================

HDL Synthesis Report


Macro Statistics

# ROMs                                                : 8

 8x3-bit ROM                                         : 8

# Adders/Subtractors                                  : 20
```

```
 3-bit adder                                       : 8
 5-bit adder                                       : 8
 5-bit subtractor                                  : 4
# Registers                                        : 26
 1-bit register                                    : 2
 12-bit register                                   : 12
 3-bit register                                    : 8
 5-bit register                                    : 4
# Comparators                                      : 14
 3-bit comparator lessequal                        : 4
 5-bit comparator equal                            : 3
 5-bit comparator lessequal                        : 7
# Multiplexers                                     : 1
 3-bit 4-to-1 multiplexer                          : 1
# Xors                                             : 1
 1-bit xor2                                        : 1


=========================================================================


=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                             : 8
 8x3-bit ROM                                       : 8
```

```
# Adders/Subtractors                                    : 20
 3-bit adder                                           : 8
 5-bit adder                                           : 8
 5-bit subtractor                                      : 4
# Registers                                             : 190
 Flip-Flops                                            : 190
# Comparators                                           : 14
 3-bit comparator lessequal                            : 4
 5-bit comparator equal                                : 3
 5-bit comparator lessequal                            : 7
# Multiplexers                                          : 1
 3-bit 4-to-1 multiplexer                              : 1
# Xors                                                  : 1
 1-bit xor2                                            : 1

=========================================================================

=========================================================================
Final Register Report

Macro Statistics
# Registers                                             : 187
 Flip-Flops                                            : 187

=========================================================================
```

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : main.ngr
Top Level Output File Name         : main
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 8

Cell Usage :
# BELS                             : 509
#      LUT2                        : 2
#      LUT2_D                      : 2
#      LUT3                        : 184
#      LUT3_D                      : 7
#      LUT3_L                      : 13
#      LUT4                        : 191
#      LUT4_D                      : 43
```

```
#      LUT4_L                          : 12
#      MUXCY                           : 16
#      MUXF5                           : 18
#      VCC                             : 1
#      XORCY                           : 20
# FlipFlops/Latches                    : 187
#      FDR                             : 185
#      FDRS                            : 2
# Clock Buffers                        : 1
#      BUFGP                           : 1
# IO Buffers                           : 7
#      IBUF                            : 3
#      OBUF                            : 4
=========================================================================


Device utilization summary:
---------------------------


Selected Device : 3s400pq208-5


 Number of Slices:                      250  out of   3584     6%
 Number of Slice Flip Flops:            187  out of   7168     2%
 Number of 4 input LUTs:                454  out of   7168     6%
 Number of IOs:                           8
 Number of bonded IOBs:                   8  out of    141     5%
```

```
 Number of GCLKs:                          1  out of      8    12%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------

================================================================================

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
Clk                                | BUFGP                  | 187   |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
-----------------------------------------
No asynchronous control signals found in this design
```

```
Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 18.944ns (Maximum Frequency: 52.789MHz)
   Minimum input arrival time before clock: 3.937ns
   Maximum output required time after clock: 31.104ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'Clk'
  Clock period: 18.944ns (frequency: 52.789MHz)
  Total number of paths / destination ports: 3149289 / 185
-------------------------------------------------------------------------
Delay:               18.944ns (Levels of Logic = 19)
  Source:            v_3/u1/subout3/Q_1 (FF)
  Destination:       v_3/u2/metric0/Q_4 (FF)
  Source Clock:      Clk rising
  Destination Clock: Clk rising

  Data Path: v_3/u1/subout3/Q_1 to v_3/u2/metric0/Q_4
```

```
                                   Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 FDR:C->Q              5   0.626   1.078  v_3/u1/subout3/Q_1 (v_3/u1/subout3/Q
 LUT4_D:I0->LO         1   0.479   0.159  v_3/u1/subset_decode_cmp_le00031_SW0
 LUT3:I2->O           12   0.479   0.973  v_3/u1/subset_decode_cmp_le00031 (v_
 LUT4:I3->O            1   0.479   0.740  v_3/u3/Madd_p1_3_xor<1>11 (v_3/p1_3<
 LUT4:I2->O            3   0.479   0.830  v_3/u4/find_min_metric_cmp_le0003239
 LUT3:I2->O            4   0.479   0.838  v_3/u4/find_min_metric_cmp_le0003256
 LUT3:I2->O            1   0.479   0.681  v_3/u4/find_min_metric_cmp_le0003211
 MUXF5:S->O            4   0.540   0.838  v_3/u4/find_min_metric_4_find_min_me
 LUT4_L:I2->LO         1   0.479   0.123  v_3/u5/b_10_cmp_le0000239 (v_3/u5/b_
 LUT4:I3->O            4   0.479   0.802  v_3/u5/b_10_cmp_le0000267 (v_3/u5/b_
 LUT4:I3->O            6   0.479   1.023  v_3/u5/_old_find_smallest_1_b_10<0>1
 LUT4:I1->O            1   0.479   0.704  v_3/u5/find_smallest_cmp_le00003 (v_
 LUT4_D:I3->LO         1   0.479   0.123  v_3/u5/find_smallest_cmp_le0000152 (
 LUT4:I3->O           16   0.479   1.074  v_3/u5/find_smallest_cmp_le00002_1 (
 LUT4:I3->O            1   0.479   0.000  v_3/u5/Msub_m_in3_lut<0> (v_3/u5/N23
 MUXCY:S->O            1   0.435   0.000  v_3/u5/Msub_m_in3_cy<0> (v_3/u5/Msub
 MUXCY:CI->O           1   0.056   0.000  v_3/u5/Msub_m_in3_cy<1> (v_3/u5/Msub
 MUXCY:CI->O           1   0.056   0.000  v_3/u5/Msub_m_in3_cy<2> (v_3/u5/Msub
 MUXCY:CI->O           0   0.056   0.000  v_3/u5/Msub_m_in3_cy<3> (v_3/u5/Msub
 XORCY:CI->O           1   0.786   0.000  v_3/u5/Msub_m_in3_xor<4> (v_3/m_in3<
 FDR:D                     0.176          v_3/u2/metric3/Q_4
----------------------------------------
```

```
    Total                      18.944ns (8.956ns logic, 9.987ns route)
                                        (47.3% logic, 52.7% route)

=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'Clk'
  Total number of paths / destination ports: 236 / 214
-------------------------------------------------------------------------
Offset:              3.937ns (Levels of Logic = 1)
  Source:            Res (PAD)
  Destination:       v_1/dff_1/Q_0 (FF)
  Destination Clock: Clk rising

  Data Path: Res to v_1/dff_1/Q_0
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O           187   0.715   2.330  Res_IBUF (Res_IBUF)
     FDR:R                     0.892          v_1/dff_2/Q_0
    ----------------------------------------
    Total                      3.937ns (1.607ns logic, 2.330ns route)
                                       (40.8% logic, 59.2% route)

=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'Clk'
  Total number of paths / destination ports: 5313304 / 4
```

```
--------------------------------------------------------------------------------
Offset:              31.104ns (Levels of Logic = 21)
  Source:            v_3/u1/subout3/Q_1 (FF)
  Destination:       Out<2> (PAD)
  Source Clock:      Clk rising

  Data Path: v_3/u1/subout3/Q_1 to Out<2>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q             5   0.626   1.078  v_3/u1/subout3/Q_1 (v_3/u1/subout3/Q
     LUT4_D:I0->LO        1   0.479   0.159  v_3/u1/subset_decode_cmp_le00031_SW0
     LUT3:I2->O          12   0.479   0.973  v_3/u1/subset_decode_cmp_le00031 (v_
     LUT4:I3->O           1   0.479   0.740  v_3/u3/Madd_p1_3_xor<1>11 (v_3/p1_3<
     LUT4:I2->O           3   0.479   0.830  v_3/u4/find_min_metric_cmp_le0003239
     LUT3:I2->O           4   0.479   0.838  v_3/u4/find_min_metric_cmp_le0003256
     LUT3:I2->O           1   0.479   0.681  v_3/u4/find_min_metric_cmp_le0003211
     MUXF5:S->O           4   0.540   0.838  v_3/u4/find_min_metric_4_find_min_me
     LUT4_L:I2->LO        1   0.479   0.123  v_3/u5/b_10_cmp_le0000239 (v_3/u5/b_
     LUT4:I3->O           4   0.479   0.802  v_3/u5/b_10_cmp_le0000267 (v_3/u5/b_
     LUT4:I3->O           6   0.479   1.023  v_3/u5/_old_find_smallest_1_b_10<0>1
     LUT4:I1->O           1   0.479   0.704  v_3/u5/find_smallest_cmp_le00003 (v_
     LUT4_D:I3->O         1   0.479   0.704  v_3/u5/find_smallest_cmp_le0000152 (
     LUT4:I3->O           4   0.479   1.074  v_3/u5/find_smallest_cmp_le00002 (v_
     LUT3:I0->O           3   0.479   1.066  v_3/u5/find_smallest_1_find_smallest
```

```
     LUT4:I0->O            2   0.479   1.040  v_3/u5/smallest_no_cmp_eq0001526 (v_
     LUT4:I0->O            1   0.479   0.740  v_3/u5/smallest_no_cmp_eq0001577 (v_
     LUT4:I2->O            1   0.479   0.851  v_3/u5/smallest_no_1_smallest_no<0>6
     LUT4:I1->O            6   0.479   1.148  v_3/u5/smallest_no_1_smallest_no<0>9
     LUT3:I0->O            1   0.479   0.000  v_3/u8/Mmux_decide_1_decide_3 (N21)
     MUXF5:I1->O           1   0.314   0.681  v_3/u8/Mmux_decide_1_decide_2_f5 (Ou
     OBUF:I->O                 4.909          Out_0_OBUF (Out<0>)
    ----------------------------------------
    Total                     31.104ns (15.011ns logic, 16.093ns route)
                                       (48.3% logic, 51.7% route)

=========================================================================
CPU : 8.28 / 8.45 s | Elapsed : 9.00 / 9.00 s

-->

Total memory usage is 149692 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :   15 (   0 filtered)
Number of infos    :    2 (   0 filtered)
```

where, Green-Hamming Codes, Red-Convolutional Codes

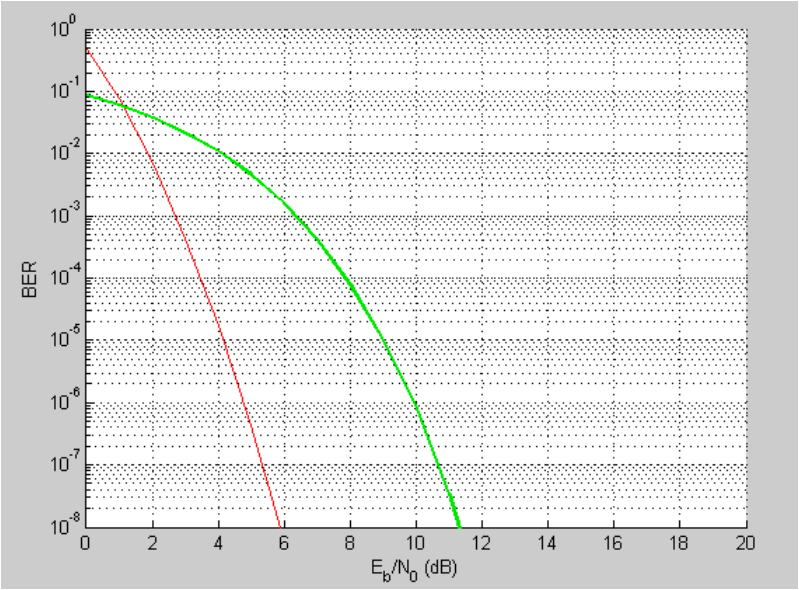

Figure 5.7: BER Comparison on Coding Techniques in BER Tool

5.2 BER Performance Comparison in BER Tool

BER and Modulation Techniques

There are different types of modulation like BPSK, QPSK, 16-PSK, 32-PSK. QAM etc. Each modulation technique has its own error function, so the performance of modulation technique is different at the time when noise is present. But high data rate transmission in limited bandwidth increase the BER.If the communication area is not larger than QAM technique is used, but in case of larger area the QPSK technique is more efficient than the QAM. In QPSK two successive bits are combined reducing the bit rate or signaling rate and also bandwidth of the channel which is a main resource of communication system. Combination of two

bits creates for distinct symbols. QPSK requires less bandwidth than BPSK to be transmitted for same length of data. BER of BPSK and QPSK is almost same.

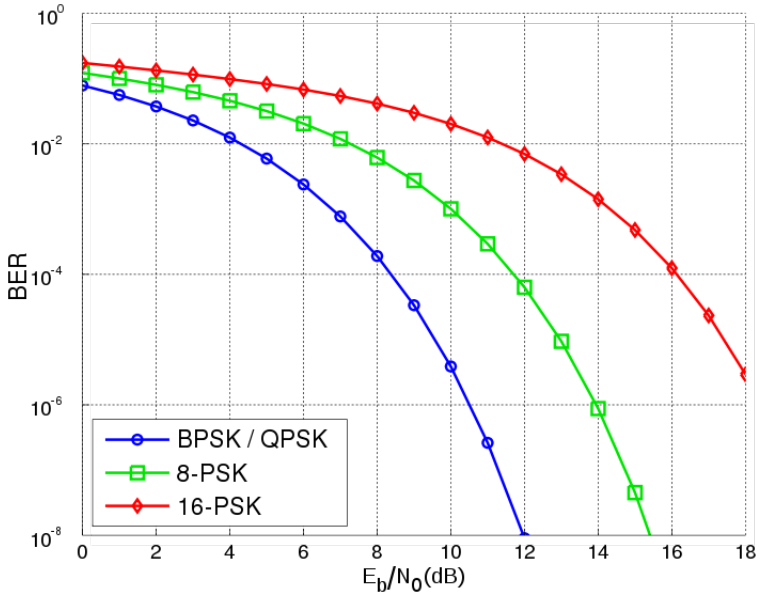

Figure 5.8: BER Comparison for Modulation Techniques

Chapter 6

Tools Used

6.1 Software Tools Used

1. MATLAB
    - Simulink.
    - BERTOOL
2. XILINX 9.1 ISE
3. ModelSim 6.3g_p1

MATLAB

MATLAB is the high-level language and interactive environment used by millions of engineers and scientists worldwide. It lets you explore and visualize ideas and collaborate across disciplines including signal and image processing, communications, control systems, and computational finance.

6.1.1 Simulink

Simulink, developed by MathWorks, is a graphical programming environment for modeling, simulating and analyzing multidomain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries.
Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems.

Key Features:

- Graphical editor for building and managing hierarchical block diagrams
- Libraries of predefined blocks for modelling continuous-time and discrete-time systems
- Simulation engine with fixed-step and variable-step ODE solvers
- Scopes and data displays for viewing simulation results
- Project and data management tools for managing model files and data
- Model analysis tools for refining model architecture and increasing simulation speed
- MATLAB Function block for importing MATLAB algorithms into models
- Legacy Code Tool for importing C and C++ code into models

Building the Model

Simulink provides a set of predefined blocks that you can combine to create a detailed block diagram of your system. Tools for hierarchical modeling, data management, and subsystem customization enable you to represent even the most complex system concisely and accurately.

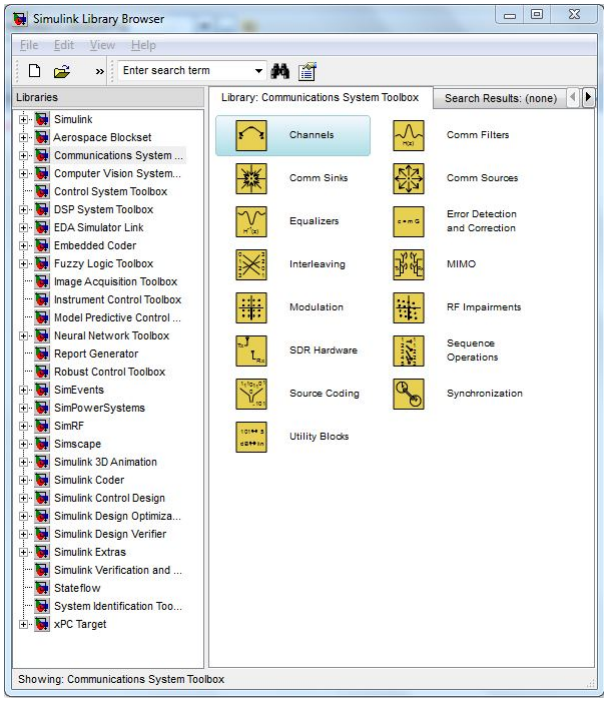

Figure 6.1: Simulink GUI

Selecting Blocks

The Simulink Library Browser includes:

- Continuous and discrete dynamics blocks, such as Integration and Unit Delay

- Algorithmic blocks, such as Sum, Product, and Lookup Table
- Structural blocks, such as Mux, Switch, and Bus Selector

Simulink an build customized functions by using these blocks or by incorporating hand-written MATLAB, C, Fortran, or Ada code into your model.

Custom blocks can be stored in their own libraries within the Simulink Library Browser Simulink add-on products let to incorporate specialized components for aerospace, communications, PID control, control logic, signal processing, video and image processing, and other applications. Add-on products are also available for modeling physical systems with mechanical, electrical, and hydraulic components.

Building and Editing the Model

The Simulink Editor gives you complete control over what you see and use within the model. For example, adding commands and submenus to the editor and context menus. Also adding a custom interface to a subsystem or model by using a mask that hides the subsystem's contents and provides the subsystem with its own icon and parameter dialog box.

Simulating the Model

Can simulate the dynamic behavior of your system and view the results as the simulation runs. To ensure simulation speed and accuracy, Simulink provides

fixed-step and variable-step ODE solvers, a graphical debugger, and a model profiler.

Generating Code

Simulink models can be configured and made ready for code generation. By using Simulink with add-on code generation products, you can generate C and C++, HDL, or PLC code directly from your model.

Communication System Tool Box

Communications System Toolbox provides algorithms and apps for the analysis, design, end-to-end simulation, and verification of communications systems in MATLAB and Simulink. Toolbox algorithms, including channel coding, modulation, MIMO, and OFDM, enable you to compose a physical layer model of your system. You can simulate your models to measure performance.

The system toolbox provides constellation and eye diagrams, bit-error-rate, and other analysis tools and scopes for validating your designs. These tools enable you to analyze signals, visualize channel characteristics, and obtain performance metrics such as error vector magnitude (EVM). Channel and RF impairment models and compensation algorithms, including carrier and symbol timing synchronizers, enable you to realistically model your link-level specifications and compensate for the effects of channel degradations.

Using Communications System Toolbox hardware support packages, you can connect your transmitter and receiver models to radio devices and verify your designs with over-the-air testing. The system toolbox supports fixed-point arithmetic and C or HDL code generation. Algorithms are available as MATLAB functions, System objects, and Simulink blocks.

6.1.2 BER Tool

Bertool launches the Bit Error Rate Analysis Tool (BERTool). The BERTool application enables you to analyze the bit error rate (BER) performance of communications systems. BERTool computes the BER as a function of signal-to-noise ratio. It analyzes performance either with Monte-Carlo simulations of MATLAB functions and Simulink models or with theoretical closed-form expressions for selected types of communication systems.

The tool will plot and analyze BER performance over a range of user-defined SNR values. Features include curve fitting, confidence intervals, and plotting of both simulated results and theoretical bounds.

6.1.3 XILINX ISE

Xilinx ISE (Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize (”compile”) their designs, perform timing analysis, examine RTL diagrams, simulate a design’s reaction to different stimuli, and configure the target device with the programmer.

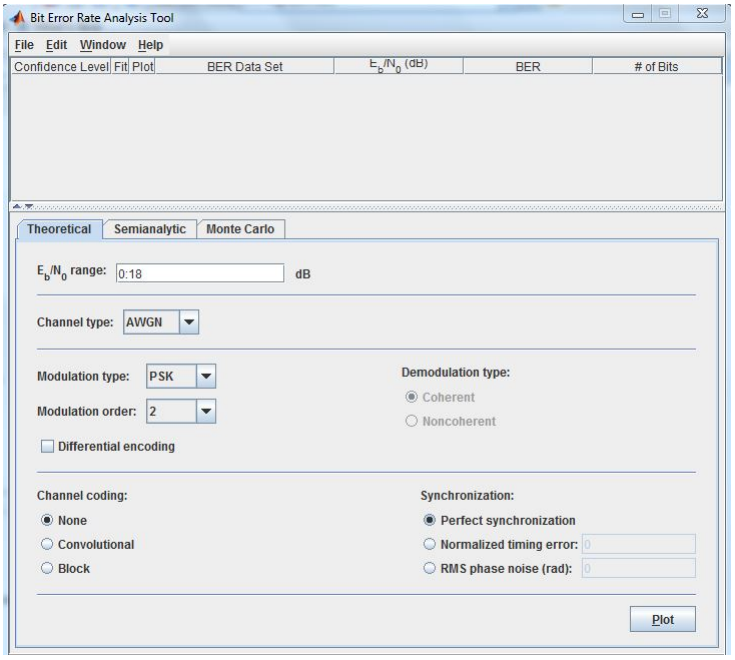

Figure 6.2: BERTOOL GUI

The Xilinx ISE is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors. The Xilinx ISE is primarily used for circuit synthesis and design, while the ModelSim logic simulator is used for system-level testing. Other components shipped with the Xilinx ISE include the Embedded Development Kit (EDK), a Software Development Kit (SDK) and ChipScope Pro.

User Interface

The primary user interface of the ISE is the Project Navigator, which includes the design hierarchy (Sources), a source code editor (Workplace), an output console (Transcript), and a processes tree (Processes).

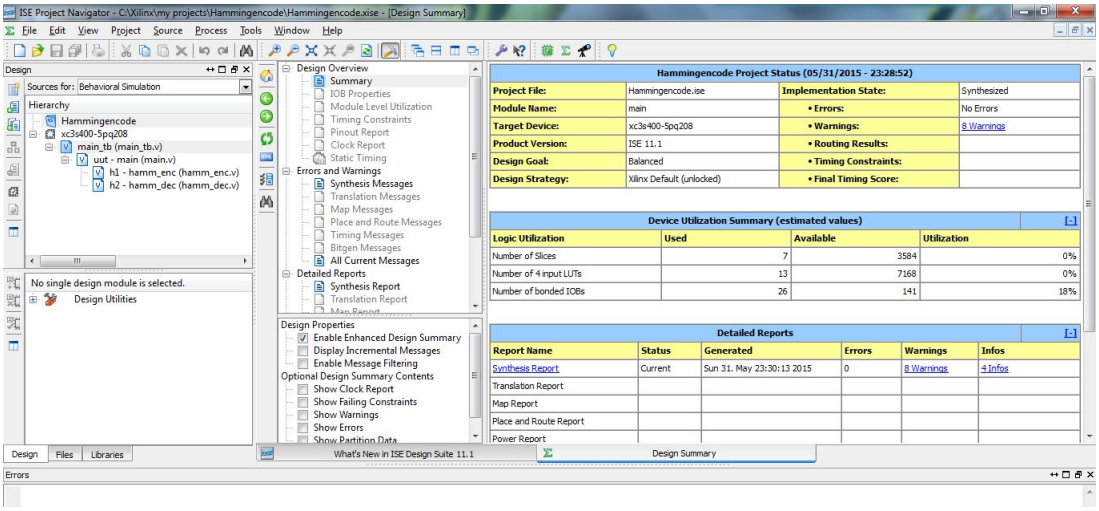

Figure 6.3: XILINX 9.1 ISE GUI

The Design hierarchy consists of design files (modules), whose dependencies are interpreted by the ISE and displayed as a tree structure. For single-chip designs there may be one main module, with other modules included by the main module, similar to the main() subroutine in C++ programs. Design constraints are specified in modules, which include pin configuration and mapping. The Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with ecah function.

The Transcript window provides status of currently running operations, and informs engineers on design issues.Such issues may be filtered to show Warnings, Errors, or both.

Simulation

System-level testing may be performed with the ModelSim logic simulator, and such test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms, or monitors which observe and verify the outputs of the device under test. ModelSim may be used to perform the following types of simulations:

- Logical verification, to ensure the module produces expected results
- Behavioural verification, to verify logical and timing issues
- Post-place and route simulation, to verify behaviour after placement of the module within the reconfigurable logic of the FPGA Synthesis

Xilinx's patented algorithms for synthesis allow designs to run up to 30percent faster than competing programs, and allows greater logic density which reduces project costs. Also, due to the increasing complexity of FPGA fabric, including memory blocks and I/O blocks, more complex synthesis algorithms were developed that separate unrelated modules into slices, reducing post-placement errors. IP Cores are offered by Xilinx and other third-party vendors, to implement system-level functions such as digital signal processing (DSP), bus interfaces, networking protocols, image processing, embedded processors, and peripherals. Xilinx has been instrumental in shifting designs from ASIC-based implementation to FPGA-based implementation.

6.1.4 ModelSim

ModelSim implements the Verilog and SystemVerilog languages as defined by the following standards:

- IEEE 1364-2005 and 1364-1995 (Verilog)
- IEEE 1800-2009 and 1800-2005 (SystemVerilog)

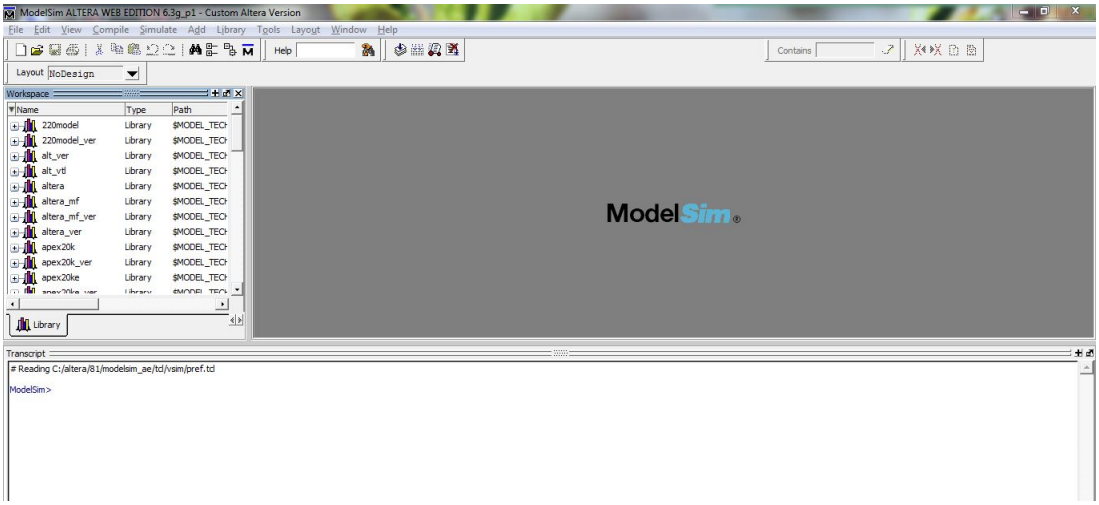

Figure 6.4: ModelSim GUI

SystemVerilog is built on top of IEEE Std 1364 for the Verilog HDL and improves the productivity, readability, and reusability of Verilog-based code. The language enhancements in SystemVerilog provide more concise hardware descriptions, while still providing an easy route with existing design and verification products into current hardware implementation flows. The enhancements also

provide extensive support for directed and constrained random testbench development, coverage-driven verification, and assertion-based verification.

The standard for SystemVerilog specifies extensions for a higher level of abstraction for modeling and verification with the Verilog hardware description language (HDL). This standard includes design specification methods, embedded assertions language, testbench language including coverage and assertions application programming interface (API), and a direct programming interface (DPI).

Basic Verilog Usage

Simulating Verilog designs with ModelSim consists of the following general steps:

- Compile your Verilog code into one or more libraries using the vlog command. See Verilog Compilation for details.
- Load your design with the vsim command. Refer to Verilog Simulation.
- Simulate the loaded design and debug as needed.

Verilog Compilation

The first time you compile a design there is a two-step process:

- Create a working library with vlib or select File ¿ New ¿ Library.
- Compile the design using vlog or select Compile ¿ Compile.

Verilog Case Sensitivity

Note that Verilog and SystemVerilog are case-sensitive languages. For example, clk and CLK are regarded as different names that you can apply to different signals or variables. This differs from VHDL, which is case-insensitive.

6.2 Hardware Used

FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing hence ”field-programmable”. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). (Circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare.)

A Spartan FPGA from Xilinx

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be ”wired together”, like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work

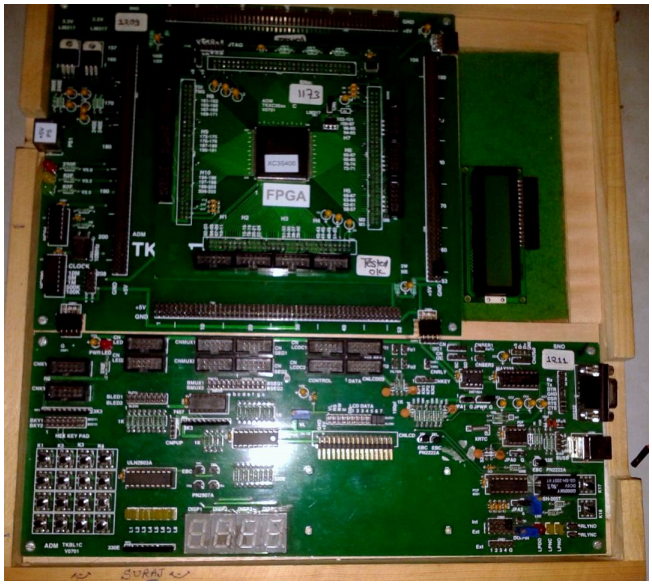

Figure 6.5: FPGA Board

with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualisation of a design. Then, using an electronic design automation tool, a technology-mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA company's proprietary place-and-route software. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA. This file is transferred to the FPGA/CPLD via a serial interface (JTAG) or to an external memory device like an EEPROM. The most common HDLs are VHDL and Verilog, although in an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent

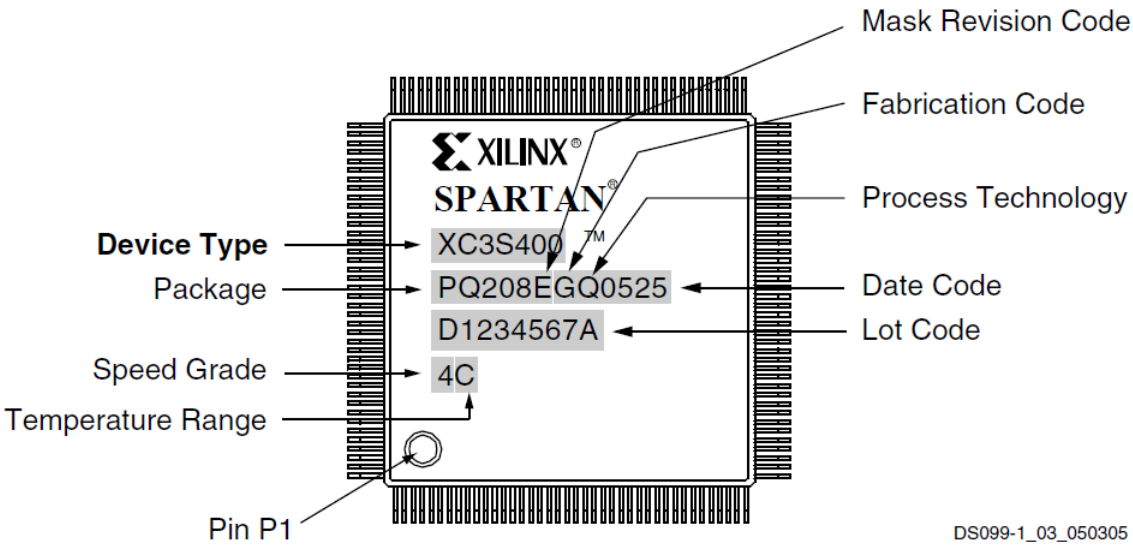

Figure 6.6: FPGA Specifications

of assembly languages, there are moves to raise the abstraction level through the introduction of alternative languages. National Instruments' LabVIEW graphical programming language (sometimes referred to as "G") has an FPGA add-in module available to target and program FPGA hardware.

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called IP cores, and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as OpenCores (typically released under free and open source licenses such as the GPL, BSD or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back-annotated onto the netlist.

Xilinx co-founders Ross Freeman and Bernard Vonderschmitt invented the first commercially viable field-programmable gate array in 1985 the XC2064.[9][not in citation given] The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 had 64 configurable logic blocks (CLBs), with two three-input lookup tables (LUTs). More than 20 years later, Freeman was entered into the National Inventors Hall of Fame for his invention.

Xilinx continued unchallenged and quickly grew from 1985 to the mid-1990s, when competitors sprouted up, eroding significant market share. By 1993, Actel (now Microsemi) was serving about 18 percent of the market. The 1990s were an explosive period of time for FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications.

Applications

Technically speaking, an FPGA can be used to solve any problem which is computable. This is trivially proven by the fact FPGA can be used to implement a soft microprocessor.[citation needed] Their advantage lies in that they are sometimes significantly faster for some applications due to their parallel nature and optimality in terms of the number of gates used for a certain process.
Specific applications of FPGAs include digital signal processing, software-defined radio, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas. Xilinx and Altera are the current FPGA market leaders and long-time industry rivals.Together, they control over 80 percent of the market.

Both Xilinx and Altera provide freeware, proprietary Windows and Linux design software (ISE and Quartus) which provides limited sets of devices.

Spartan3

The Spartan-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to 5,000,000 system gates.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Features:

- Low-cost, high-performance logic solution for high-volume, consumer-oriented applications
- Densities up to 74,880 logic cells
- SelectIO interface signaling
- Up to 633 I/O pins
- 622+ Mb/s data transfer rate per I/O
- 18 single-ended signal standards
- 8 differential I/O standards including LVDS, RSDS
- Termination by Digitally Controlled Impedance
- Signal swing ranging from 1.14V to 3.465V
- Double Data Rate (DDR) support
- DDR, DDR2 SDRAM support up to 333 Mb/s
- Logic resources
- Abundant logic cells with shift register capability
- Wide, fast multiplexers
- Fast look-ahead carry logic
- Dedicated 18 x 18 multipliers
- JTAG logic compatible with IEEE 1149.1/1532

- SelectRAM hierarchical memory
- Up to 1,872 Kbits of total block RAM
- Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
- Clock skew elimination
- Frequency synthesis
- High resolution phase shifting
- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE and WebPACK software development systems
- MicroBlaze and PicoBlaze processor, PCI, PCI Express PIPE Endpoint, and other IP cores
- Pb-free packaging options
- Automotive Spartan-3 XA Family variant

Chapter 7

Conclusion

7.1 Conclusion

The results carry out a comparison of error rate performance of the coding schemes. BER is inversely proportional to SNR and hence to Eb/N0. Eb/No ratio is taken in X-axis and BER is taken in Y-axis.BER is measured for 1 to 10 units of Eb/n0.

The simulation results shows that performance of (poly 2 trellis (7, [171,133])) convolution coding technique is better than that of (11,7,1) hamming codes.

The graph infers that convolutional codes have better BER performance than Hamming Codes. This result is obtained by using BPSK modulation technique. It is observed that the BER value is affected by the modulation technique used.

7.2 Future Scope

- This work can be extended for estimating the performance of error correcting codes in Rayleigh and Ricean fading channels.
- This work can be used to make the coding techniques software dependent. This in turn gives scope to make the other blocks of the communication system software dependent.
- Reconfigurable, adaptive channel encoders and decoders can be designed for different channel abnormalities.
- Cognitive radio can be designed taking throughput, effective bandwidth utilisation into consideration.
- SDR-Software Defined Radio which do not require much hardware changes for handling different types signals,different communications can be designed, and this architecture have only two hardware blocks that too RF end blocks.

Bibliography

[1] Prakash C.Gupta. *"Data Communications"*, Prentice-Hall of India pvt. ltd.,New Delhi,2002.

[2] Shu Lin / Daniel J. Costello,Jr. *"Error Control Coding:Fundamentals and Applications"*,Prentice-Hall, Inc. Englewood Cliffs,New Jersey 07632,1983.

[3] Simon Haykin. *"Communication Systems"*,John Wiley and Sons,Inc. New York,2000.

[4] Jayshree S. Nandaniya,Nilesh B. Kalani,Dr.G.R.Kulkarni. *"Comparative Analysis of Different Channel Coding Techniques"*,IRACST International Journal of Computer Networks and Wireless Communications (IJCNWC), vol.4,No.2,April 2014.

[5] Himanshu Saraswat, Govind Sharma Sudhir Kumar Mishra and Vishwajeet. *"Performance Evaluation and Comparative Analysis of Various Concatenated Error Correcting Codes Using BPSK Modulation for AWGN Channel"*,International Journal of Electronics and Communication Engineering. ISSN 0974-2166 Volume 5, Number 3 (2012), pp. 235-244,International Research Publication House.

[6] Jagpreet Singh , Dr. Shalini Bahel. *"Comparative Study of Data Transmission Techniques of Different Block Codes over AWGN Channel using Simulink"*,International Journal of Engineering Trends and Technology (IJETT) Volume 9 Number 12 - Mar 2014.

[7] Nagaraj H Nemagoudar, Aruna Rashmi V. *"HDL Implementation of Hamming Code Encoder and Decoder"*, March 2014

[8] Proakis J, Salehi.*"Fundamental of Communication Systems"*,International Edition,2004.

[9] Mahe Jabeen, Salma Khan. *"Design of Convolution Encoder and Reconfigurable Viterbi Decoder"*,International Journal of Engineering and Science, Vol. 1, Issue 3 (Sept 2012), PP 15-21.

[10] Andrew.W.Moore. *"Digital Communications 1"*,andrew.moore@cl.cam.ac.uk, 2009.

[11] Dr.Wa'il.AH.Hadi,*"Communication Systems 2"*,2008.

[12] George Kennedy, Bernard Davis, S R M Prasanna. *"Electronic Commiunication Systems"*,5e, TMH Plublications,2011.

[13] John G Proakis Masoud Salehi,*"Digital Communications"*,TMH Publications,5e,2008.

[14] Theodore S Rappaport, *"Wireless Communications Principles and Practice"*,Pearson Publications, 2e 2002.

[15] Simon Haykin., *"Digital communications"*,Wiley Publications, 4e 2010.

[16] [5] John G Proakis, Masoud Salehi.*"Contemporary Communication Systems-using Matlab"*,2e 2003.

[17] P.Sunil Kumar, Dr.M.G.Sumithra and Ms.M.Sarumathi .*"Performance Comparison of Rayleigh and Rician Fading Channels In QAM Modulation Scheme Using Simulink Environment"*,International Journal of Computational Engineering Research, Volume-03, Issue-5.

[18] Vikas Gupta, Dr. Chanderkant Verma. *"Error Detection and Correction: An Introduction"*,International Journal of Advanced Research in Computer Science and Software Engineering,Volume 2, Issue 11, November 2012.

[19] Priti Shankar.*"Error Correcting Codes"*,Series,Article-Resonance 1997.

[20] Yunghsiang S. Han. *"Introduction to Binary Convolutional Codes"*.

[21] IIT,Kharagpur. *"Channel Coding"*,Module 6, version 2.