DATA ENCODING AND DECODING USING MANCHESTER CODING TECHNIQUES

BY

OLORUNTOMI FAHAD OKIKIOLA
ND/23/COM/PT/0156

A PROJECT REPORT SUBMITTED TO THE

DEPARTMENT OF COMPUTER SCIENCE, INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY, KWARA STATE POLYTECHNIC ILORIN

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF NATIONAL DIPLOMA (ND) IN COMPUTER SCIENCE

JULY, 2025

CERTIFICATION

This is to certify that this project is written by **OLORUNTOMI, Fahad Okikiola** with matriculation number **ND/23/COM/PT/0156** in Computer Science Department, Institute of Information and Communication Technology, Kwara State Polytechnic, Ilorin.

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

MR. ISIAKA, O.S. Date

(Project Supervisor)

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

MR. OYEDEPO, F.S. Date

(*Head of Department*)

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

External Supervisor Date

DEDICATION

I dedicated this project to almighty God, whose support and encouragement were invaluable throughout this endeavor. May this work contribute to a more secure future.

ACKNOWLEDG

MENT

I would like to express my sincere gratitude to my project supervisor, Mr. Isiaka O.S., for his invaluable guidance, support, and mentorship throughout the course of this project.

His insightful feedback, expert advice, and encouragement have been instrumental in shaping the development of the packet filtering technique for network security. I deeply appreciate his dedication and patience, which motivated me to overcome challenges and strive for excellence in my work.

This project would not have been possible without his unwavering support. Thank you, Mr. Isiaka O.S., for your mentorship and encouragement.

Also to my Parents and my siblings Sfor their support and word of encouragement

TABLE OF CONTENTS

ABSTRACT

The introduction of the internet brought with it lots of new problems in data communication. In response to these challenges, network analysts have discovered various coding techniques such as hamming coding techniques and Manchester coding techniques. The purpose of the project is to write code that detects and corrects errors in data transmission using the Manchester coding technique. The application will be written in the PYTHON programming language. The adopted methodology is prototype methodology. Finally, a system with automatic error detection and correction will be designed.

CHAPTER ONE

GENERAL INTRODUCTION

1.1. INTRODUCTION

Communication system is generally subject to data corruption as data travels from source to destination. There is a possibility of the occurrence of error that may occur and thereby reducing the quality of the message content, and for this reason, encoding and decoding of data in data communication is very paramount. Error needs to be detected and corrected. Error correcting codes (ECCs) are commonly used to protect against soft errors and thereby enhance system reliability and data integrity, (Daramola, 2014). In computers, encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage. Decoding is the opposite process -- the conversion of an encoded format back into the original sequence of characters. Encoding and decoding are used in data communications, networking, and storage to ensure integrity and reliability transmission of data, (Jameel, 2018).

Coding theory is concerned with the transmission of data across noisy channels and the recovery of corrupted messages (Altaian, 2006). It has found widespread applications in electrical engineering, digital communication, mathematics, and computer science. While the problems in coding theory often arise from engineering applications, it is fascinating to note the crucial role played by mathematics in the development of the field. The importance of algebra in coding theory is a commonly acknowledged fact, with many deep mathematical results being used in elegant ways in the advancement of coding theory; therefore coding theory appeals not just to engineers and computer scientists, but also to mathematicians and hence, coding theory is sometimes called algebraic coding theory, (Doran, 2002). An algebraic techniques involving finite fields, group theory, polynomial algebra as well as linear algebra deal with the design of error correcting codes for the reliable transmission of information across noisy channels.

Channel encoding deals with the source encoded message u , by introducing some extra data bits that will be used in detecting and/or even correcting the transmitted message, Hall, (2003). Source encoding involves changing the message source to a suitable code say u to be transmitted through the channel. Thus the result of the source encoding is a code word, say v . Likewise, channel decoding and source decoding are applied on the destination side to decode the received code word r as correctly as possible. For example, let us consider a message source of four fruit words to be transmitted: apple, banana, cherry and grape. The source encoder encodes these words into the following binary data $(u_1 u_2 u_3 u_4)$:

Apple → u_1 = (0, 0), banana → u_2 = (0, 1),

Cherry → u_3 = (1, 0), Grape → u_4 = (1, 1).

Suppose the message 'apple' is to be transmitted over a noisy channel. The bits $u_1 = (0,0)$ will be transmitted instead. Suppose an error of one bit occurred during the transmission and the code (0, 1) is received instead. The receiver may not realize that the message was corrupted and the received message will be decoded into 'banana'. Encoding should not be confused with encryption, a process in which data is deliberately altered so as to conceal its content. Encryption can be done without changing the particular code that the content is in, and encoding can be done without deliberately concealing the content.

The main advantage of using digital signal is that errors introduced by noise during the transmission can be detected and possibly corrected.For communication using cables, the random motion of charges in conducting (e.g. resistors), known as thermal noise, is the major source of noise. For wireless communication channel, noise can be introduced in various ways. In the case of mobile phones, noise includes the signals sent by other mobile phone users in the system, (Daramola, 2014). Manchester Coding is a common technique which comes under the Digital Data Coding. The term coding includes both Encoder and Decoding. Purpose of an encoder in Manchester coding is to convert the bits or data into a coded data and that of the Decoder is to recover the original data from the coded form. The Encoder is used along with the transmitter and the Decoder along with the receiver. Here each interval is divided into two halves. Therefore there will be a change of signal level at the middle of each bit, (Anjali and Satishkumar, 2015).

1.2. STATEMENT OF THE PROBLEM

In any environment, noise, electromagnetic radiations and any other forms of disturbances affect communication leading to corrupted messages, errors in the received messages or even to an extent of the message not being received at all. Despite the overwhelming advantages of standard digital communication compared to analog signaling, there is a general problem, which is the issue of synchronization: the receiver must know when exactly to sample the incoming data. And if data is to be transmitted over a long channel, we could run into trouble.

1.3. AIM AND OBJECTIVES

The aim of this project work is to develop an encoding and decoding of data using Manchester coding techniques that guarantees confidentiality, data integrity and authentication. The objectives of this project work are listed below;

i. Adaptation of Manchester coding techniques in data transmission.

ii. Implementation of the adapted algorithm using PYTHON.

1.4. SIGNIFICANCE OF THE STUDY

Transmission of data across noisy channel and other forms of interference affect communication resulting to misdirected messages. Hence, we need the recovery of these corrupted messages. Considering the present concern with privacy and secrecy, and the prospect that such problems will increase significantly as communication services and data repositories grow, importance is thus attached to finding means of detecting and correcting any error that occur.

1.5. SCOPE AND LIMITATION OF THE PROJECT

While the areas of application of encoding and decoding of data are in Deep space communication, Satellite communication, Data transmission, Data storage, Mobile communication, File transfer, and Digital audio/video transmission. The scope of this research is limited to data transmission.

1.6. ORGANIZATION OF THE PROJECT

This is the overall summary of the work presented in this project. Chapter one emphasizes on general introduction, statement of the problem, aims and objectives, significance of the study, scope and limitation of the proposed study. Chapter two is centered on the literature review. Chapter three is project methodology this deals with the method of data collection, analysis of data, problems of the existing system, and description of the proposed system and benefit of the proposed system**.** Chapter four deals with the design, implementation and documentation of the system, the design involves the system designs, output design form and the input design form, and the procedure of the design the implementation involves programming language used and the hardware and software support, the documentation of the system involves operations of the system and the maintenance of the system. Chapter five is summary of the work done, the conclusion, and recommendation.

CHAPTER TWO

LITERATURE REVIEW

2.1 REVIEW OF RELATED PAST WORKS

Various researchers have done related works on ECC (Error correcting codes) and even channel coding also known as FORWARD ERROR CORRECTION; some of their works are listed below:

Adham (2015) described how we can generate 7 redundancy bit for 64 bit information data. These redundancy bits are to be interspersed at the bit positions (n = 1, 2, 4, 8, 16, 32 and 64) of the original data bits, so to transmit 64 bit information data we need 7 redundancy bit generated by even parity check method to make 71 bit data string. He reported that at the destination receiver point, we receive 71 bit data, this receives data may be corrupted due to noise. In Hamming technique the receiver will decide if data have an error or not, so if it detected the error it will find the position of the error bit and corrects it. His paper presents the design of the transmitter and the receiver with Hamming code redundancy technique using VHDL. The Xilinx ISE 10.1 Simulator was used for simulating VHDL code for both the transmitter and receiver sides.

Shaik, Sreenivasulu and Syed (2015) referenced that several EDAC techniques have been proposed and employed to effectively detect and correct errors introduced during data transmission over a communication channel or at the destination domain during storage. Some of these techniques can detect: only single error, all unidirectional errors, only burst errors, any bit in a data packet is change from one to zero or zero to one it means error is occur in same, errors with known locations assume a code is correct if the error location are known or cannot detect errors which appear in the same location in a pair of message codes. Coding techniques that detects and correct errors are more precise at detecting error locations and correcting them, however if more than one error occur, it becomes a challenge to detect all errors in a data frames and converted back its original form. In this paper, an advanced error detection and correction method to protect against errors is proposed. This method is based on 4D parities checking.

Debalina & Krishanu (2015) implemented the architecture that was simulated with different technologies (16nm, 22nm, 32nm, and 45nm) with the help of TANNER EDA Tool for the study of total power dissipation of the circuit. The analysis shows that with the decrease of channel length, there was a decrease of 12.65 % and 2.37 % power dissipation in 16nm compared to 22nm in encoder and decoder circuit. The values of model parameters are used from Predictive Technology Model (PTM). For circuit verification, the hamming code algorithm is implemented in Spartan 3A family XC3S700A FPGA device using VHDL coding technique.

Vaidehi & Justus (2016), asserted that the high level of security and the fast hardware and software implementation of the Advanced Standard Encryption (AES) have made it in the first choice for many critical applications. Advanced Encryption Standard (AES) is the self – determining hardware architecture for producing crypto mechanism. In a secure transmission system, Content Addressable Memory (CAM) based transmissions always provide Silent Data Corruption (SDC). In order to overcome this problem, Extended hamming code has been executed in their paper for reducing the SDC effect in AES Encryption and Decryption. In the Fault detection and Detection (DAED) are done by using hamming codes.

Babitha & Divya (2015) insinuated that error correction codes are used in semiconductor memories to protect information against soft errors. Soft error is major concern for memory reliability especially for memories that are used in space applications. In such cases simple error correction codes are preferred due to their simple encoding/decoding logic, low redundancy and low encoding/decoding latency. Hamming codes are attractive as they are simple to construct for any word length and the encoding/decoding can be done with low delay. But they only allow single error correction or double error detection, so a multiple error can lead to a wrong decoding. Nowadays multiple errors are becoming more frequent as integration scale increases. Multiple errors occurs mainly to adjacent bits.

2.2 REVIEW OF GENERAL STUDY

Himmat & Sandhya (2010) show the working of a forward error correction coding techniques using convolutional encoding with Viterbi decoding. This paper initially explains the working of a convolutional encoder. The encoded bit stream is then passed through an additive white Gaussian noise (AWGN) channel, quantized and received at the decoder. The original data stream is recovered by either a hard decision Viterbi decoder or a soft decision Viterbi decoder. This entire FEC technique is demonstrated, both practically, using Matlab and theoretically. They show the simulation plots, characterizing the performance and factors affecting the FEC coding technique. These factors include primarily the noise level as well as the encoder memory size. The decoding of convolution encoders are investigated in respect of the parameter bit error rate.

Garima, Prerna & Zahid (2007) demonstrated many data coding techniques. The coding technique chosen for a particular application is determined by the performance requirements of the system. Short links with high signal to noise ratio can use simple linear line codes such as RZ, NRZ, AMI or Manchester. When the link is longer and affected by noise, more advanced coding techniques are employed. Redundancy can be added to the line code to provide coding gain. The coding gain improves system performance. Line codes which add redundancy include multilevel line codes, bipolar violation insertion codes and codes which maintain bounded RDS. For more advanced systems, error checking codes may be used when the system also uses one of the ARQ protocols.

Joanne & Mishra (2011) introduce a novel binary, long double error correcting, systematic code (8 2 5) that can detect and correct errors up to two bits in the received vector using simple concept of syndrome decoding. The motivation behind the construction of this code is the idea to achieve 100% error correction on the receiver side and to use the encoder/decoder that is simpler than the existing double error correcting codes.

2.3 HISTORY OF DATA TRANSMISSION CODES

The history of data-transmission codes began in 1948 with the publication of a famous paper by Claude Shannon. Shannon showed that associated with any communication channel or storage channel is a number C (measured in bits per second), called the capacity of the channel, which has the following significance: Whenever the information transmission rate R (in bits per second) required of a communication or storage system is less than C then, by using a data-transmission code, it is possible to design a communication system for the channel whose probability of output error is as small as desired. Shannon, however, did not tell us how to find suitable codes; his contribution was to prove that they exist and to define their role. Throughout the 1950s, much effort was devoted to finding explicit constructions for classes of codes. The first block codes were introduced in 1950 when Hamming described a class of single-error-correcting block codes and he published what is now known as Hamming code, which remains in use in many applications today.

In 1957, among the first codes used practically were the cyclic codes which were generated using shift registers. It was quickly noticed by Prange that the cyclic codes have a rich algebraic structure, the first indication that algebra would be a valuable tool in code design. In the 1960s, the major advances came in 1960 when Hocquenghem and Bose and Ray-Chaudhuri found a large class of multiple-error-correcting codes (the BCH codes). The discovery of BCH codes led to a search for practical methods of designing the hardware or software to implement the encoder and decoder. In the same year independently, Reed, Solomon and Arimoto found a related class of codes for non-binary channels. Concatenated codes were introduced by Forney (1966).

During the 1970s, these two avenues of research began to draw together in some ways and to diverge further in others. Meanwhile, Goppa (1970) defined a class of codes that is sure to contain good codes, though without saying how to identify the good ones.

The 1980s saw encoders and decoders appear frequently in newly designed digital communication systems and digital storage systems, Hamming.

The 1990s witnesses an evaluation of all groups in informatics at the universities in Norway. The evaluation was performed by a group of internationally recognized experts. The committee observed that the period 1988-92, had the largest number of papers (27) published in internationally refereed journals among all the informatics groups in Norway. In the period 1995-1997 the goal of finding explicit codes which reach the limits predicted by Shannon's original work has been achieved. The constructions require techniques from a surprisingly wide range of pure mathematics: linear algebra, the theory of fields and algebraic geometry all play a vital role, Han. Not only has coding theory helped to solve problems of vital importance in the world outside mathematics, it also has enriched other branches of mathematics, with new problems as well as new solutions, Kolman. In 1998 Alamouti described a space-time code.

In 2000 Aji, McEliece and others synthesize several decoding algorithms using message passing ideas. In the period 2002-2006 many books and papers are introduce such as Algebraic soft-Decision Decoding of Reed- Solomon Codes by Koetter R., and Error Control Coding: Fundamentals and Applications by Lin and Costello and Error Correction Coding by Moon. During this decade, development of algorithms for hard-decision decoding of large nonbinary block codes defined on algebraic curves, Kabatiansky. Decoders for the codes known as hermitian codes are now available and these codes may soon appear in commercial products. At the same time, the roots of the subject are growing even deeper into the rich soil of mathematics.

Doumen (2003) researched on the aims of cryptography in providing secure transmission of messages in the sense that two or more persons can communicate in a way that guarantees confidentiality, data integrity and authentication. Sebastia (2003) studied on the Block error correcting codes. He found that the minimum distance decoder maximizes the likelihood of correcting errors if all the transmission symbols have the same probability of being altered by the channel noise. He also noted that if a code has a minimum distance d, then d(C) - 1 is the highest integer with the property that the code detects d(C) - 1 errors.

Todd, studied on the Error control coding. He showed that if a communication channel introduces fewer error than the minimum distance errors, d(C), then these can be detected and if d(C)- 1 errors are introduced, then error detection is guaranteed. He also noted that the probability of error detection depends only on the error introduced by the communication channel and that the decoder will make an error if more than half of the received bit strings are in error. In 2009, Nyaga studied the Cyclic ISBN-10 to improve the conventional ISBN-10. They designed a code that would detect and correct multiple errors without many conditions attached for error correction and found out that the code could correct as many errors as the code could detect. The method involves trial and error calculation and thus it needs to be improved on and simplified to speed up the process.

2.4 ERROR DETECTION

In information theory and coding theory with applications in computer science and telecommunication, error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data in many cases.

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data that has been determined to be corrupted. Error-detection and correction schemes can be either systematic or non-systematic: In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of *check bits* (or *parity data*), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, an error has occurred at some point during the transmission. In a system that uses a non-systematic code, the original message is transformed into an encoded message that has at least as many bits as the original message, (Hossen M.D., Karmakar S., and Mehedi al aowab M.D., 2017).

If the channel characteristics cannot be determined, or are highly variable, an error-detection scheme may be combined with a system for retransmissions of erroneous data. This is known as automatic repeat request (ARQ), and is most notably used in the Internet. An alternate approach for error control is hybrid automatic repeat request (HARQ), which is a combination of ARQ and error-correction coding.

Error correction may generally be realized in two different ways:

i. *Automatic repeat request (ARQ)* (sometimes also referred to as *i*): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data. Every block of data received is checked using the error detection code used, and if the check fails, retransmission of the data is requested – this may be done repeatedly, until the data can be verified.

ii. *Forward error correction (FEC)*: The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data.

2.5 MANCHESTER CODING TECHNIQUES

In data transmission, Manchester encoding is a form of digital encoding in which data bits are represented by transitions from one logical state to the other. This is different from the more common method of encoding, in which a bit is represented by either a high state such as +5 volts or a low state such as 0 volts. When the Manchester code is used, the length of each data bit is set by default. This makes the signal self-clocking. The state of a bit is determined according to the direction of the transition. In some systems, the transition from low to high represents logic 1, and the transition from high to low represents logic 0. In other systems, the transition from low to high represents logic 0, and the transition from high to low represents logic 1. The chief advantage of Manchester encoding is the fact that the signal synchronizes itself. This minimizes the error rate and optimizes reliability, (Rouse, 2005).

Manchester coding states that there will always be a transition of the message signal at the mid-point of the data bit frame. What occurs at the bit edges depends on the state of the previous bit frame and does not always produce a transition. A logical “1” is defined as a mid-point transition from low to high and a “0” is a mid-point transition from high to low. Manchester encoding (first published in 1949) is a synchronous clock encoding technique used by the physical layer to encode the clock and data of a synchronous bit stream. In this technique, the actual binary data to be transmitted over the cable are not sent as a sequence of logic 1's and 0's (known technically as Non Return to Zero (NRZ)). Instead, the bits are translated into a slightly different format that has a number of advantages over using straight binary encoding (i.e. NRZ), (Wesołowski, 2009).

In the Manchester encoding, logic 0 is indicated by a 0 to 1 transition at the centre of the bit and logic 1 is indicated by a 1 to 0 transition at the centre of the bit. Note that signal transitions do not always occur at the 'bit boundaries' (the division between one bit and another), but that there is always a transition at the centre of each bit. The Manchester encoding rules are summarized below:

The original data	e sent
Logic 0	0 to 1 (upward transition at bit centre)
Logic 1	1 to 0 (downward transition at bit centre)

Figure2.1: Manchester coding rules

Source: https://erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html

The following diagram shows a typical Manchester encoded signal with the corresponding binary representation of the data (1,1,0,1,0,0) being sent.

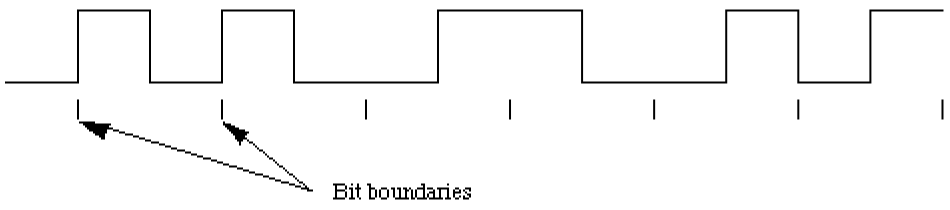


Figure 2.2: Manchester encoded signal

Source: https://erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html

Note that signal transitions do not always occur at the 'bit boundaries' (the division between one bit and another), but that there is always a transition at the centre of each bit.The encoding may be alternatively viewed as a phase encoding where each bit is encoded by a postive 90 degree phase transition, or a negative 90 degree phase transition. The Manchester code is therefore sometimes known as a Biphase Code.For example, the pattern of bits " 0 1 1 1 1 0 0 1 " encodes to " 01 10 10 10 10 01 01 10". Another more curious example is the pattern " 1 0 1 0 1 etc" which encodes to "10 01 10 01 10 " which could also be viewed as "1 00 11 00 11 0 ". Thus for a 10 Mbps Ethernet LAN, the preamble sequence encodes to a 5 MHz square wave! (i.e., One half cycle in each 0.1 microsecond bit period.) Hence, a transmission rate of 10 Mbps implies that each bit is sent in 0.1 microseconds.

For a coaxial cable, the speed at which the signal travels along the cable is approximately 0.77 times the speed of light (i.e. 0.77x3x10E8). A bit therefore occupies 23 metres of cable. Under the same conditions the smallest frame would be 13.3 km! If you wish to do the same calculation for a twisted pair cable, you would have to take into consideration that the propagation speed is slower at 1.77x10E8 (0.59c). Increasing the bit rate, for example using 100BTx, decreases the time available to send each bit into the wire, but does not change the speed at which the edge of the bits travel through the cable!, (Fairhurst, 2007).

CHAPTER THREE

ANALYSIS OF THE CURRENT SYSTEM

3.1 RESEACH METHODOLOGY

Manchester code follows an algorithm to encode data like all other coding methods. The algorithm goes like: the Encoding, the Receiver and the Decoding.

The Encoding: The encoding involves a transmitter that encodes clock and signals in a synchronous bit stream. It represents data in bit stream, but with line transitions. A transition from HIGH to LOW is represented by logic 0, and a transition from LOW to HIGH is represented by logic 1. The idea is the modality of bits " 0 1 1 1 1 0 0 1 " encodes to " 01 10 10 10 10 01 01 10". Another example is the modality " 1 0 1 0 1 etc" more curious which encodes to "10 01 10 01 10”.

Transitions in the Manchester Encoding occur at the center and beginning of each bit. The transition at the center is defined by the bit value, while the transition at the beginning is dependent on the value of the previous bit. Let’s consider the following diagram:

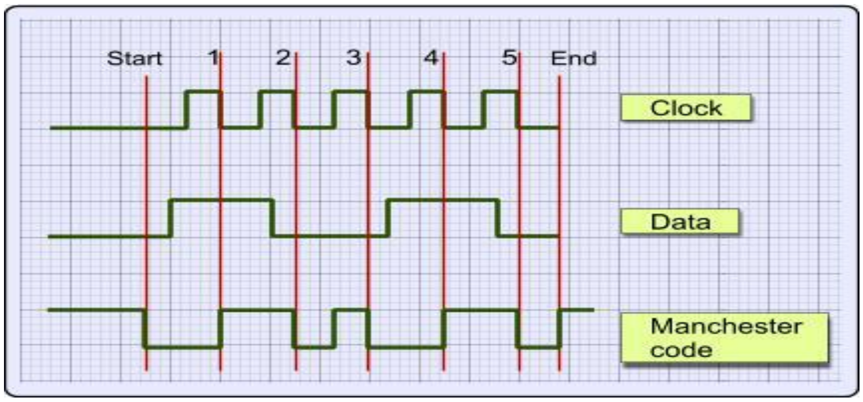


Figure 3.1: Diagrammaticalrepresentation of encoding processes

Source: (Jameel, 2018)

The data that was used in encoding is the binary number 10010, from left to right on every falling edge of the clock the coding happen. It's has a LOW to HIGH transition, on the first falling edge of the clock, the data is HIGH The code has a HIGH to LOW transition because the data is LOW, on the second falling edge of the clock the same algorithm is utilized for the rest of the signal.

As the preceding Manchester encoded signals shows:

i. The value of 1 for the first bit forces a high-to-low transition at the center of that bit.

ii. The value of 0 for the second bit forces a low-to-high transition at the center of that bit and, because the first bit transitioned from high-to-low, no transition occurs at the start of that bit.

iii. The value of 0 for the third bit forces a low-to-high transition at the center of that bit and because the second bit transitioned from low-to-high, a high-to-low transition occurs at the start of that bit.

iv. The value of 1 for the fourth bit forces a high-to-low transition at the center of that bit and, because the third bit transitioned from low-to-high, no transition occurs at the start of that bit.

v. The value of 0 for the fifth bit forces a low-to-high transition at the center of that.

The Receiver: A device that receives the encoded bit stream is responsible for decoding the bit stream by separating the clock and data information. In most cases, the receiver must retrieve the original data stream by using only the encoded signal. This simplifies the communications channel, but means the receiver must overcome the following:

i. Differences between the clock used to encode the signal and the clock in the receiver (see the figure below).

ii. The two clocks can be close in frequency, but small frequency errors occur.

iii. The phase between the clocks will be arbitrary.

The Manchester Receiver validates the computations performed by a Manchester Receiver device that is modeled. Numerous approaches are available for implementing a Manchester receiver. This example uses a Delay Lock Loop (DLL) that requires the receiver to use a clock that is very close in frequency to the transmit clock. This results in a simple clock recovery circuit that has a limited frequency lock range. The receiver over-samples the received data stream at 16 times the data rate. Thus, the receive clock must have a nominal period of 1/16th the data period. To compensate for minor differences between the transmitting and receiving clocks or drifts in the channel delay, the receiver adjusts its data period by up to one clock cycle (+/-) per data period. Thus, the receiver can use 15, 16 or 17 clock cycles to recover the data encoded from the incoming sampled signal. For example, if the receiver clock is slightly faster than the transmitter clock (frequency error), the receive cycle occasionally needs to add an extra clock cycle to compensate. Large sudden phase errors, such as those that occur at startup time, require multiple data periods to acquire a good lock on the signal. By limiting the maximum phase correction to 1/16th of the total data period, the receiver can be slow to correct large phase errors.

Decoding with Inphase and Quadrature Convolution: Decoding a received Manchester signal can occur in several ways, but the approach taken in this example is to consider Manchester Encoding as a digital phase modulation with two symbols: +180 and -180 degrees. By convolving the incoming signal with a reference in phase (I) and quadrature (Q) waveform at the modulation frequency, it is possible to extract the data and retrieve information about any phase errors in the received waveform.

3.2 DESCRIPTION OF THE EXISTING SYSTEM

This system is use to encode or decode data in communication processes. The stream of bits to be sent over a channel is passed to the encoder. The encoder encodes with hamming 7, 4 encoder, then transmits to the receiving end. This end decides it and immediately figure out if there is error in the bits received and fashion out on what bit does the error is. And finally correct the error to get actual streams of bits sent. Hamming codes require *O(lg(n))* parity bits for *n* data bits. Each parity bit checks some (but not all) of the data bits. If an error occurs in a data bit, all the parity bits that checked that data bit will show the error, allowing us to uniquely determine where the error occurred.

3.3 PROBLEM OF THE EXISTING SYSTEM

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

To guarantee correction of up to *t* errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

If code scheme has a Hamming distance $d_{min} = 4$ this code guarantees the detection of up to three errors (s = 3), but it can correct only up to one error. In other words, if this code is used for error correction, part of its capability is wasted. Error correction codes need to have an odd minimum distance (3, 5, 7 . . .). The existing system errors:

i. Its highly dependent on distance

ii. It is very demanding and problematic

iii. It can detect only an odd number of bits in error

iv. It detects single and burst errors

3.4 ANALYSIS OF THE PROPOSED SYSTEM

The proposed system is coding of data using Manchester coding technique. In this technique, the actual binary data to be transmitted over the cable are not sent as a sequence of logic 1's and 0's (known technically as Non Return to Zero (NRZ)). Instead, the bits are translated into a slightly different format that has a number of advantages over using straight binary encoding (i.e. NRZ). In the Manchester encoding, logic 0 is indicated by a 0 to 1 transition at the centre of the bit and logic 1 is indicated by a 1 to 0 transition at the centre of the bit. As there is always a transition at the centre of each bit, the design start with a data source that is used to transmit data in the form of (0s and 1s) connected to a Manchester encoder.

3.5 ADVANTAGES OF THE PROPOSED SYSTEM

The advantages are:

i. It enhances integrity of data sent to receiver.

ii. Encoding and decoding are easy to implement.

iii. They would be effective as a simple and efficient code over a channel where it is known that errors are burst-free and tend to be widely dispersed.

iv. Executes faster

CHAPTER FOUR

DESIGN AND IMPLEMENTATION OF THE SYSTEM

4.1 DESIGN OF THE SYSTEM

System design involves the logical description of the entire components that makes up the system. Description of the system design involves the output design, input design, and procedure design of the new system.

4.1.1 OUTPUT DESIGN

The output is referred to as report, processing result, message etc. that related directly to input files which are stored and being processed which later turn to generate output. Below is sample of the output;

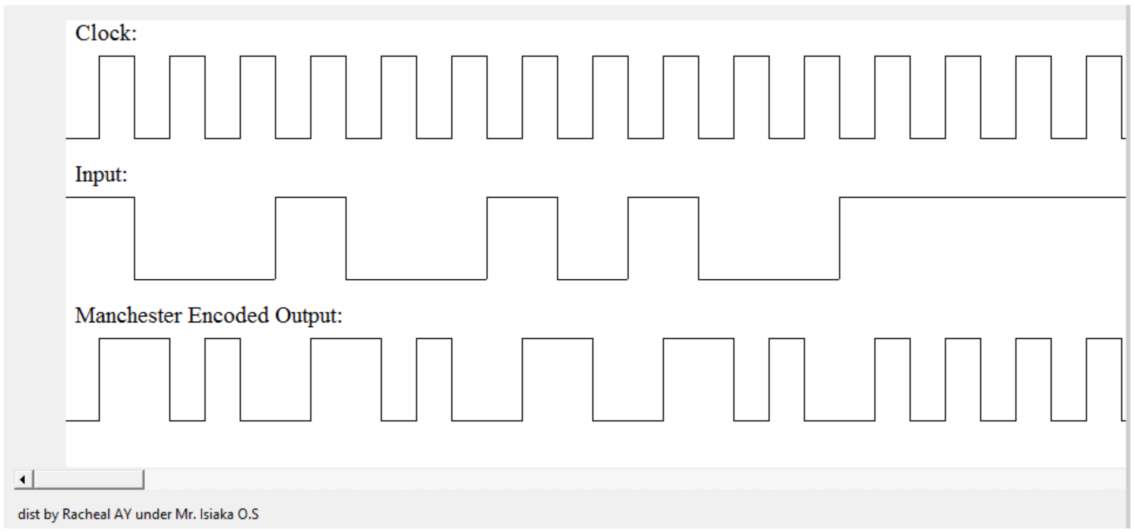

Clock:
Input:
Manchester Encoded Output:
dist by Racheal AY under Mr. Isiaka O.S

Figure 4.1 Manchester encoded output for ‘I’m kwarapoly students’

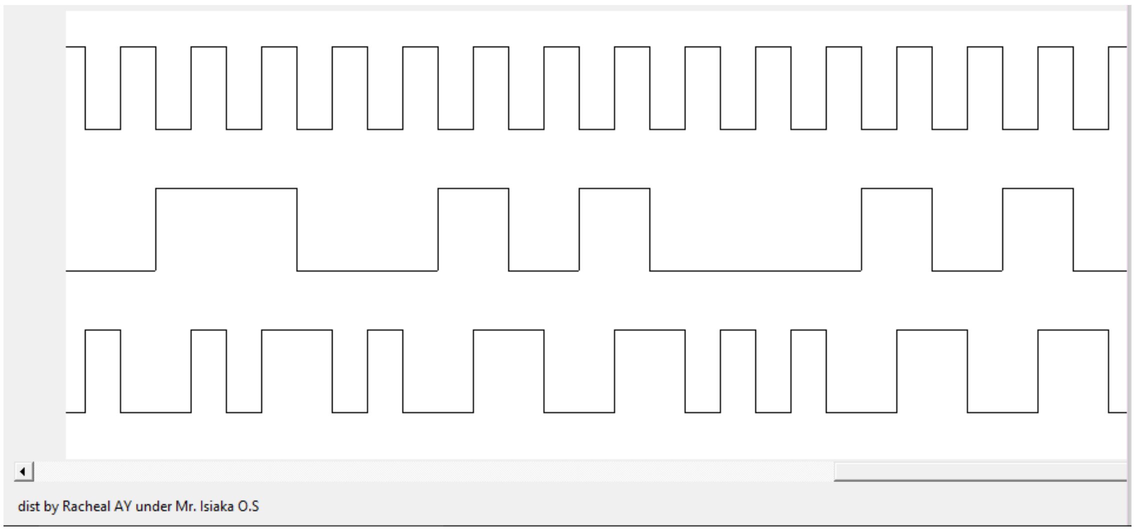


Figure 4.2 Manchester encoded output for 'Encode'

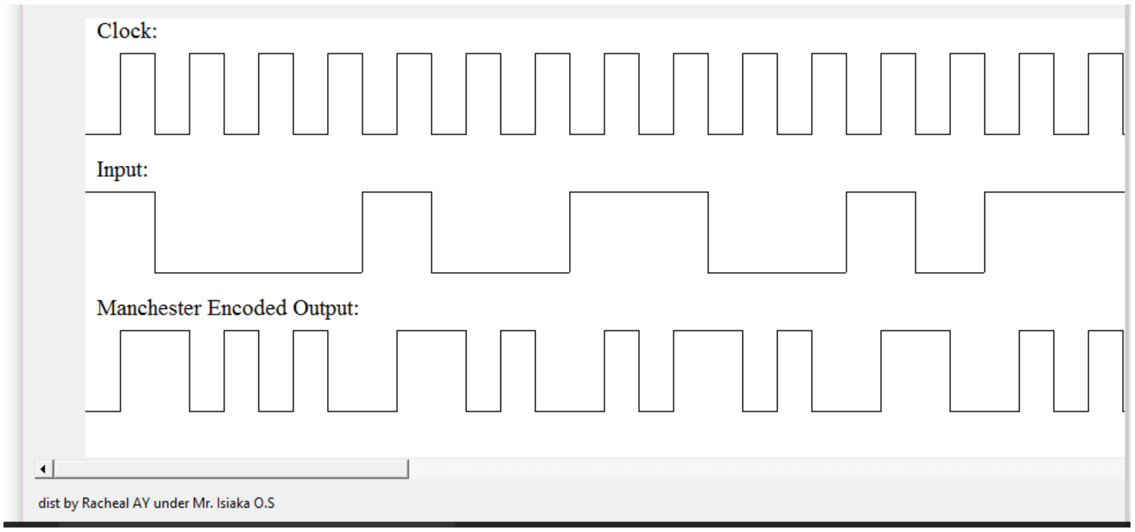
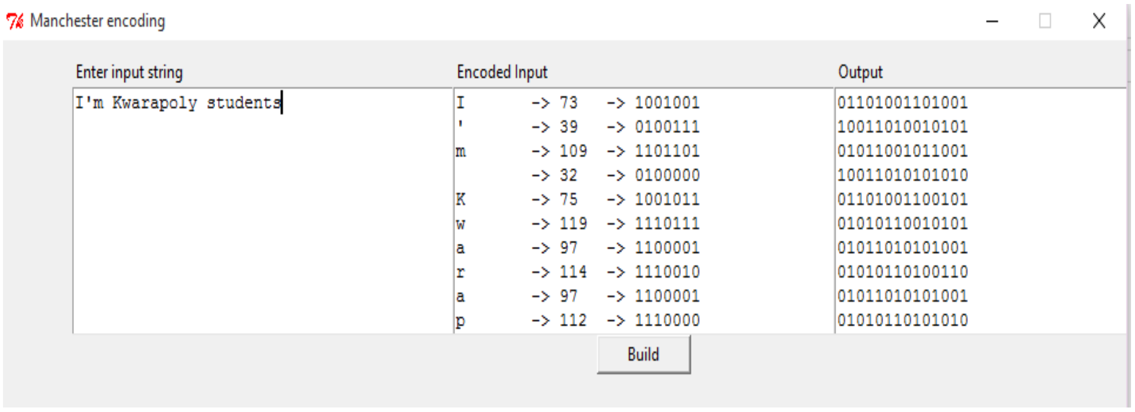

Clock:
Input:
Manchester Encoded Output:
dist by Racheal AY under Mr. Isiaka O.S

Figure 4.3: Manchester encoded output for 'Decode'

4.1.2 INPUT DESIGN

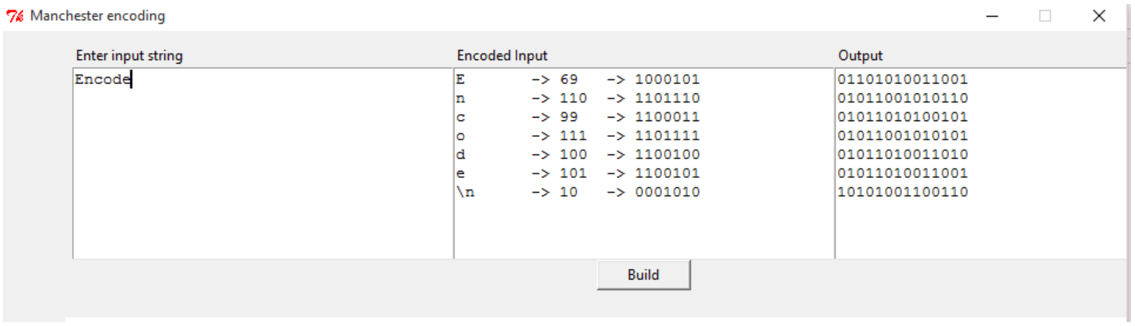
The input to the new system is use in encoding the data. The inputs are processed to obtain the desired outputs. The design is as follow:



```
Manchester encoding

Enter input string          Encoded Input                         Output
I'm Kwarapoly students      I     -> 73    -> 1001001             01101001101001
                            '     -> 39    -> 0100111             10011010010101
                            m     -> 109   -> 1101101             01011001011001
                                  -> 32    -> 0100000             10011010101010
                            K     -> 75    -> 1001011             01101001100101
                            w     -> 119   -> 1110111             01010110010101
                            a     -> 97    -> 1100001             01011010101001
                            r     -> 114   -> 1110010             01010110100110
                            a     -> 97    -> 1100001             01011010101001
                            p     -> 112   -> 1110000             01010110101010
```

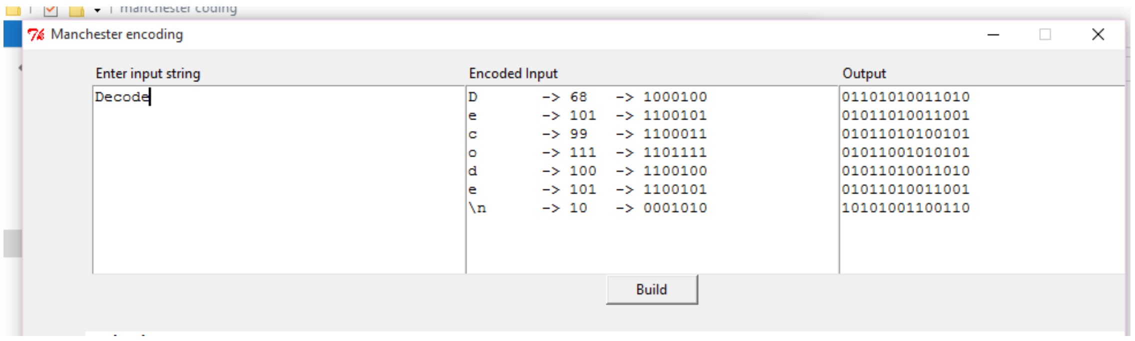
Figure 4.4: ‘I’m kwarapoly students’ sample data to be encoded with Manchester



```
Manchester encoding

Enter input string          Encoded Input                         Output
Encode                      E     -> 69    -> 1000101             01101010011001
                            n     -> 110   -> 1101110             01011001010110
                            c     -> 99    -> 1100011             01011010100101
                            o     -> 111   -> 1101111             01011001010101
                            d     -> 100   -> 1100100             01011010011010
                            e     -> 101   -> 1100101             01011010011001
                            \n    -> 10    -> 0001010             10101001100110
```

Figure 4.5: ‘Encode’ sample data to be encoded with Manchester



```
Manchester encoding

Enter input string          Encoded Input                         Output
Decode                      D     -> 68    -> 1000100             01101010011010
                            e     -> 101   -> 1100101             01011010011001
                            c     -> 99    -> 1100011             01011010100101
                            o     -> 111   -> 1101111             01011001010101
                            d     -> 100   -> 1100100             01011010011010
                            e     -> 101   -> 1100101             01011010011001
                            \n    -> 10    -> 0001010             10101001100110
```

Figure 4.6: ‘Decode’ sample data to be encoded with Manchester

4.2 SYSTEM IMPLEMENTATION

Implementation is the process of putting the theory into practical i.e. all system designed are being written down and put into test before it is presented.

4.2.1 CHOICE OF PROGRAMMING LANGUAGE

For this project PYTHON was chosen over other programming languages, for the following strong points.

Multi-Threading – The ability to run several threads simultaneously is a feature exhibited by modern CPUs. The programming language takes a huge advantage of this feature, and allows us to write the program in such a way that when training a set of inputs, verification for another account can be done. Also several training operations can take place at once, while several verification operations can also take place at the same time.

Looping – The functionality of the Manchester coding simulation depends on the array. Hence the more optimized way of working with loops provided by PYTHON comes in handy. With the index for each loop, we can easily iterate over a large array without having to separately define indexes.

4.2.2 SOFTWARE REQUIREMENTS

A software requirement is the specification of the minimum software needed to run the new system of maternal and mortality rate analysis. The software required to run this program is as follows:

i. Operating System: Windows, Unix, Linux etc.

ii. Programming Language: PYTHON (hypertext preprocessor)

4.2.3 HARDWARE REQUIREMENTS

In order for the system to function efficiently the following minimum hardware configuration is required:

i. 2.0 GHz Intel Pentium IV Processor and above

ii. 1GB of available RAM (4GB recommended)

iii. Minimum of 30GB of available disk space

iv.

4.3 SYSTEM DOCUMENTATION

Documentation of the system covers the program documentation (packaging) and system operation (program usage.)

The program contains a box to type any words (samples data to encode or decode with Manchester coding techniques). After entering the data, user should click build to see the representation of the supplied data in bits and Manchester coding diagrammatic equivalent.

4.3.1 PROGRAM DOCUMENTATION

After the program has been well tested well with input that the output has already been known, the next step is to install or deploy the software for use. The processes of installing are been stated below:

i. Get piece of the application

ii. Install the application

iii. Launch it

iv. Start using it

4.3.2 OPERATING THE SYSTEM

This basically discusses the environment in which the result processing application will run. Since, the computerized students registration and result processing runs as a single user, then the preferred choice of operating system is window operating system. It is a graphic user interface (GUI) which presents commands available in form of icon for performing certain tasks. It further supports the execution of the compiled application without any restriction.

To operate this result processing program

i. Click on start menu

ii. Click on all program

iii. Then select the result processing program

iv. Start operating the application

4.3.3 MAINTENANCE OF THE SYSTEM

It is very essential to maintain the system in order to keep the system file up to date. Computer hardware should be prevented against system failure and the software should be updated frequently.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 SUMMARY

Since designing and simulation of encoding and decoding of data in data communication is very essential, more so there is need to provide a very easy way of checking redundancy and likewise to know if data transferred has been manipulated or not and to reduce the stress of error checking over a network. This project detects and corrects errors using Manchester Coding techniques to detect if there is error and also correct the error. It uses 0s and 1s to send data over a channel and by so doing encoding the data that is to be sent from the sender to the receiver.

5.2 CONCLUSION

If data is to be transmitted over a channel, it is possible to absolutely get an error free data at the receiving end and in achieving this with Manchester coding techniques there would be error free communication of data over a channel from the sender to the receiver in data communication.

5.3 RECOMMENDATION

There are a number of solutions to handle the increasing number of errors due to technology scaling in transmission. Information redundancy or coding is one of the most feasible solutions. Nowadays the research has focused on handling limited number of error, single or double. When considering future technology scaling down, it is expected that the error rates increase drastically, these errors can be categorized into two types: transient errors which causes the component to malfunction for some time, other type is permanent errors which cause the component to malfunction forever. In the long run, an efficient fault tolerance approach that considers occurrence of multiple simultaneous permanent and transient errors would be needed in encoding of data. Therefore, Manchester coding techniques is recommended to be used at various data transmission channel.

REFERENCES

Brewster R.L. (ed.) (1994). *Data Communications and Networks*, 3rd edn.

Clark G.C. and Cain J.B. (1981).*Error-Correction Coding for Digital Communications*. Plenum Press Mathematik, **1**, 269–271

Dunlop J. and Smith D.G. (1994).*Telecommunications Engineering*, 3rd edn. Chapman & Hall

Ferouzan B.A. (2000). *Data Communications and Networking*, 2nd edn. McGraw-Hill

Flood J.E. (1997). *Telecommunication Networks*, 2ndedn. IEEE

Ford L.R. Jr and Fulkerson D.R. (1962). *Flows in Networks*. Princeton University Press

Halsall F. (1996). *Data Communications, Computer Networks and Open Systems*, 4th edn. Addison-Wesley

Halsall F. (2001). *Multimedia Communications*. Addison-Wesley

Hamming R.W. (1950).*Error detecting and error correcting codes*.Bell Systems Technical Journal, **29**, 147–160

Handel R., Huber M.N. and Schroeder S. (1998).*ATM Networks*: Concepts, Protocols, Applications, 3rdedn. Addison-Wesley

Held G. (1998).*Data Communications Networking Devices*: Operation, Utilization and Lan and Wan Internetworking, 4th edn. John Wiley

Meggitt J.E. (1961). *Error correcting codes and their implementation for data transmission systems*. IRE Transactions on Information Theory, IT-**7**, 234–244

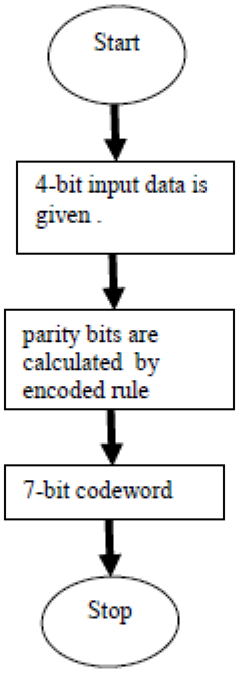
Peterson W.W. (1961). *Error Correcting Codes*. Cambridge: MIT Press

Stallings W. (1991). *Data and Computer Communications*, 3rd edition. Prentice Hall (Upper SaddleRiver, NJ)

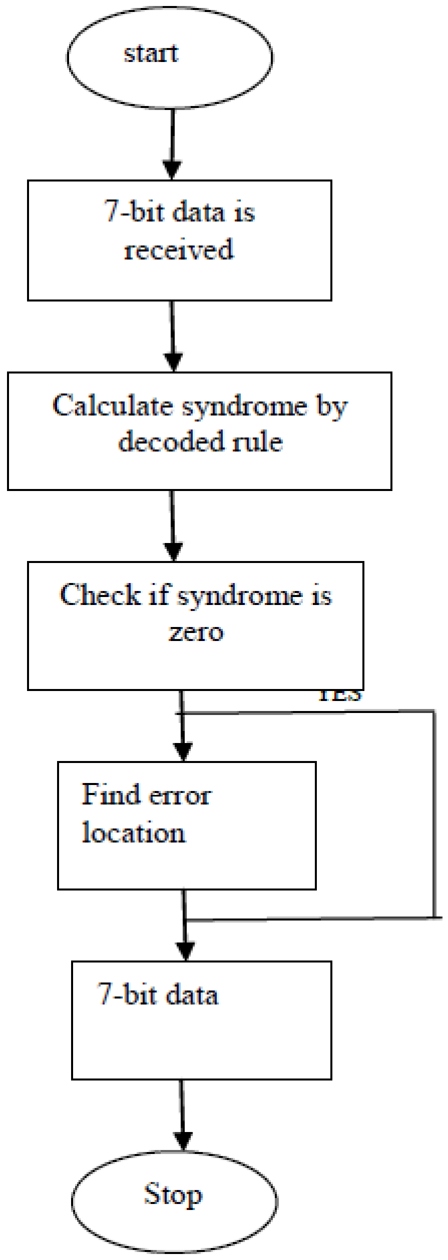
Stallings W. (2003).*Cryptography & Network Security*, 3rd edn. Pearson Education Ghanbari M. (1999). Data Coding: An introduction to standard codecs. IEEE

FLOWCHART

MANCHESTER ENCODER


Start
4-bit input data is given .
parity bits are calculated by encoded rule
7-bit codeword
Stop

MANCHESTER DECODER



SOURCE CODE

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<PropertyGroup>
<Configuration Condition=" '$(Configuration)' == " ">Debug</Configuration>
<Platform Condition=" '$(Platform)' == " ">x86</Platform>
<ProductVersion>
</ProductVersion>
<SchemaVersion>2.0</SchemaVersion>
<ProjectGuid>{A4E231D2-6244-403A-A0CE-FBAB3B245A4D}</ProjectGuid>
<OutputType>WinExe</OutputType>
<StartupObject>Hamming_Algorithm.My.MyApplication</StartupObject>
<RootNamespace>Hamming_Algorithm</RootNamespace>
<AssemblyName>Hamming Algorithm</AssemblyName>
<FileAlignment>512</FileAlignment>
<MyType>WindowsForms</MyType>
<TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
<TargetFrameworkProfile>Client</TargetFrameworkProfile>
<IsWebBootstrapper>false</IsWebBootstrapper>
<PublishUrl>C:\Users\Yusuph\pix\</PublishUrl>
<Install>true</Install>
<InstallFrom>Disk</InstallFrom>
<UpdateEnabled>false</UpdateEnabled>
<UpdateMode>Foreground</UpdateMode>
<UpdateInterval>7</UpdateInterval>
<UpdateIntervalUnits>Days</UpdateIntervalUnits>
<UpdatePeriodically>false</UpdatePeriodically>
<UpdateRequired>false</UpdateRequired>
<MapFileExtensions>true</MapFileExtensions>
<ApplicationRevision>1</ApplicationRevision>
<ApplicationVersion>1.0.0.%2a</ApplicationVersion>
<UseApplicationTrust>false</UseApplicationTrust>
<PublishWizardCompleted>true</PublishWizardCompleted>
<BootstrapperEnabled>true</BootstrapperEnabled>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|x86' ">
<PlatformTarget>x86</PlatformTarget>
<DebugSymbols>true</DebugSymbols>
<DebugType>full</DebugType>
<DefineDebug>true</DefineDebug>
<DefineTrace>true</DefineTrace>
```

```
<OutputPath>bin\Debug\</OutputPath>
<DocumentationFile>Hamming Algorithm.xml</DocumentationFile>
<NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
<WarningLevel>1</WarningLevel>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|x86' ">
<PlatformTarget>x86</PlatformTarget>
<DebugType>pdbonly</DebugType>
<DefineDebug>false</DefineDebug>
<DefineTrace>true</DefineTrace>
<Optimize>true</Optimize>
<OutSputPath>bin\Release\</OutputPath>
<DocumentationFile>Hamming Algorithm.xml</DocumentationFile>
<NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
<WarningLevel>1</WarningLevel>
</PropertyGroup>
<PropertyGroup>
<OptionExplicit>On</OptionExplicit>
</PropertyGroup>
<PropertyGroup>
<OptionCompare>Binary</OptionCompare>
</PropertyGroup>
<PropertyGroup>
<OptionStrict>Off</OptionStrict>
</PropertyGroup>
<PropertyGroup>
<OptionInfer>On</OptionInfer>
</PropertyGroup>
<PropertyGroup>
<ApplicationManifest>My Project\app.manifest</ApplicationManifest>
</PropertyGroup>
<PropertyGroup>
<SignManifests>true</SignManifests>
</PropertyGroup>
<PropertyGroup>
<ManifestCertificateThumbprint>5BA067D3005571737F46708F14C854DA530CF657</Manife
stCertificateThumbprint>
</PropertyGroup>
<PropertyGroup>
<ManifestKeyFile>Hamming Algorithm_TemporaryKey.pfx</ManifestKeyFile>
</PropertyGroup>
<PropertyGroup>
<GenerateManifests>true</GenerateManifests>
```

```
</PropertyGroup>
<PropertyGroup>
<TargetZone>LocalIntranet</TargetZone>
</PropertyGroup>
<ItemGroup>
<Reference Include="System" />
<Reference Include="System.Data" />
<Reference Include="System.Deployment" />
<Reference Include="System.Drawing" />
<Reference Include="System.Windows.Forms" />
<Reference Include="System.Xml" />
<Reference Include="System.Core" />
<Reference Include="System.Xml.Linq" />
<Reference Include="System.Data.DataSetExtensions" />
</ItemGroup>
<ItemGroup>
<Import Include="Microsoft.VisualBasic" />
<Import Include="System" />
<Import Include="System.Collections" />
<Import Include="System.Collections.Generic" />
<Import Include="System.Data" />
<Import Include="System.Drawing" />
<Import Include="System.Diagnostics" />
<Import Include="System.Windows.Forms" />
<Import Include="System.Linq" />
<Import Include="System.Xml.Linq" />
</ItemGroup>
<ItemGroup>
<Compile Include="ApplicationEvents.vb" />
<Compile Include="CodeFile1.vb" />
<Compile Include="Form1.vb">
<SubType>Form</SubType>
</Compile>
<Compile Include="Form1.Designer.vb">
<DependentUpon>Form1.vb</DependentUpon>
<SubType>Form</SubType>
</Compile>
<Compile Include="Form2.Designer.vb">
<DependentUpon>Form2.vb</DependentUpon>
</Compile>
<Compile Include="Form2.vb">
<SubType>Form</SubType>
</Compile>
```

```
<Compile Include="Module1.vb" />
<Compile Include="Module2.vb" />
<Compile Include="My Project\AssemblyInfo.vb" />
<Compile Include="My Project\Application.Designer.vb">
<AutoGen>True</AutoGen>
<DependentUpon>Application.myapp</DependentUpon>
</Compile>
<Compile Include="My Project\Resources.Designer.vb">
<AutoGen>True</AutoGen>
<DesignTime>True</DesignTime>
<DependentUpon>Resources.resx</DependentUpon>
</Compile>
<Compile Include="My Project\Settings.Designer.vb">
<AutoGen>True</AutoGen>
<DependentUpon>Settings.settings</DependentUpon>
<DesignTimeSharedInput>True</DesignTimeSharedInput>
</Compile>
<Compile Include="SplashScreen1.Designer.vb">
<DependentUpon>SplashScreen1.vb</DependentUpon>
</Compile>
<Compile Include="SplashScreen1.vb">
<SubType>Form</SubType>
</Compile>
</ItemGroup>
<ItemGroup>
<EmbeddedResource Include="Form1.resx">
<DependentUpon>Form1.vb</DependentUpon>
</EmbeddedResource>
<EmbeddedResource Include="Form2.resx">
<DependentUpon>Form2.vb</DependentUpon>
</EmbeddedResource>
<EmbeddedResource Include="My Project\Resources.resx">
<Generator>VbMyResourcesResXFileCodeGenerator</Generator>
<LastGenOutput>Resources.Designer.vb</LastGenOutput>
<CustomToolNamespace>My.Resources</CustomToolNamespace>
<SubType>Designer</SubType>
</EmbeddedResource>
<EmbeddedResource Include="SplashScreen1.resx">
<DependentUpon>SplashScreen1.vb</DependentUpon>
</EmbeddedResource>
</ItemGroup>
<ItemGroup>
<None Include="Hamming Algorithm_TemporaryKey.pfx" />
```

```
<None Include="My Project\app.manifest" />
<None Include="My Project\Application.myapp">
<Generator>MyApplicationCodeGenerator</Generator>
<LastGenOutput>Application.Designer.vb</LastGenOutput>
</None>
<None Include="My Project\Settings.settings">
<Generator>SettingsSingleFileGenerator</Generator>
<CustomToolNamespace>My</CustomToolNamespace>
<LastGenOutput>Settings.Designer.vb</LastGenOutput>
</None>
</ItemGroup>
<ItemGroup>
<BootstrapperPackage Include=".NETFramework,Version=v4.0,Profile=Client">
<Visible>False</Visible>
<ProductName>Microsoft .NET Framework 4 Client Profile %28x86 and
x64%29</ProductName>
<Install>true</Install>
</BootstrapperPackage>
<BootstrapperPackage Include="Microsoft.Net.Client.3.5">
<Visible>False</Visible>
<ProductName>.NET Framework 3.5 SP1 Client Profile</ProductName>
<Install>false</Install>
</BootstrapperPackage>
<BootstrapperPackage Include="Microsoft.Net.Framework.3.5.SP1">
<Visible>False</Visible>
<ProductName>.NET Framework 3.5 SP1</ProductName>
<Install>false</Install>
</BootstrapperPackage>
<BootstrapperPackage Include="Microsoft.Windows.Installer.3.1">
<Visible>False</Visible>
<ProductName>Windows Installer 3.1</ProductName>
<Install>true</Install>
</BootstrapperPackage>
</ItemGroup>
<Import Project="$(MSBuildToolsPath)\Microsoft.VisualBasic.targets" />
<!-- To modify your build process, add your task inside one of the targets below and uncomment
it.
     Other similar extension points exist, see Microsoft.Common.targets.
<Target Name="BeforeBuild">
</Target>
<Target Name="AfterBuild">
</Target>
  --></Project>
```