DESIGN AND IMPLEMENTATION OF ENCODER-DECODER FOR LDPC (Low Density Parity Check-Codes)

THESIS SUBMITED
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

MASTER OF ENGINEERING
IN
ELECTRONICS & TELECOMMUNICATION ENGINEERING

By
PUJA SHAW
Examination Roll no.:M4ETC1617
Regd.No.:128932 of 2014-2015
Class Roll no.:001410702022

Under the guidance of
Prof. MRINAL KANTI NASKAR

Department of Electronics & Telecommunication Engineering
Jadavpur University, Kolkata-700032
West Bengal, India

May-2016

FACULTY OF ENGINEERING & TECHNOLOGY
JADAVPUR UNIVERSITY

This is to certify that the thesis entitled "**DESIGN AND IMPLEMENTATION OF ENCODER-DECODER FOR LDPC (Low Density Parity Check-Codes)**" has been carried out by **PUJA SHAW** (Class Roll No.: **001410702022** ,Examination Roll No.:**M4ETC1617** And Registration No.: **128932 of 2014-2015**) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the degree of Master of Electronics & Telecommunication Engineering.

Prof. Mrinal Kanti Naskar
Supervisor
Department of Electronics and
Telecommunication Engineering
Jadavpur University
Kolkata-700032

Prof. Palaniandavar Venkateswaran
Head of the department
Electronics and Telecommunication
Engineering
Jadavpur University
Kolkata-700032

Prof. Sivaji Bandyopadhyay
Dean
Faculty Council of
Engineering and Technology
Jadavpur University
Kolkata-700032

FACULTY OF ENGINEERING & TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL#

The forgoing thesis, entitled "**DESIGN AND IMPLEMENTATION OF ENCODER-DECODER FOR LDPC (Low Density Parity Check-Codes)**" is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or accept every statement made, opinion expressed or conclusion drawn there in but approve the thesis only for the purpose for which it has been submitted.

Committee on final Examination for
Evaluation of the thesis:

_____
Additional Examiner

_____
Supervisor

#*only in case the thesis is approved*

DECLARATION OF ORGINALITY AND COMPILANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original work by the undersigned candidate, as part of his Master of Electronics and Telecommunication studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and references all material and results that are not original to this work.

Name : PUJA SHAW

Examination Roll No. :

Thesis Title : DESIGN AND IMPLEMENTATION OF ENCODER-DECODER FOR LDPC (Low Density Parity Check-Codes)

_____

Signature of the candidate

ACKNOWLEDGEMENT

This thesis is the result of the work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude to all of them.

With immense pleasure , I express my sincere gratitude, regards and thanks to my project guide Prof. Mrinal Kanti Naskar for his excellent guidance, invaluable suggestions and continuous encouragement throughout the project. I have been fortunate to have him as my guardian than guide as he has been a great influence on me, both as a person and as a professional.

I wish to express my thanks to all my teachers who taught me during the first year of my course work and also to all my lab mates for their constant support.

Above all , I am blessed with such a caring family. I extend my deepest gratitude to my parents, for their invaluable love, affection, encouragement and support and for their struggle to make me a good person inspite of many obstacles.

(PUJA SHAW)

Table of Contents

List of figures

Pages

List of Tables

Chapter 6. LDPC Decoder

ABSTRACT

LDPC codes have very good error correcting performance approaching Shannon's limit. Good error correcting performance enables efficient and reliable communication system. But LDPC decoding algorithm need to be executed efficiently to meet the cost, bandwidth, power requirements. Since its discovery there has been many researches still on construction and implementations of LDPC codes. LDPC codes can be designed over a wide space with parameters such as girth, rate, length. LDPC hardware design and implementation depends on the parity matrix construction. There is a need to develop methods for constructing LDPC codes over a wide range of rate and length with good performance and ease of hardware implementation.

This thesis is about construction and hardware implementation of LDPC codes. But all the issues regarding complexities of constructing/ designing and implementation of LDPC codes discussed above are too many to cover in this thesis. The main contribution of this thesis is on development of design of LDPC construction and implementation of belief propagation algorithm for LDPC decoding.

Chapter 1

Introduction

1. Introduction

Communication systems transmit data from source to destination through a channel or medium like air, wire lines, optical fiber etc. The reliability of the received data depends on the channel and external noise that can interface or distort the signal representing the data. The noise introduces error in the transmitted data. Shannon showed that the reliable transmission could be achieved if the data rate is less than channel capacity. The theorem shows that a sequence of codes with rate less than the channel capacity have the capability of correcting all the errors as the code length goes to infinity.

In digital communication system there are mainly two methods of error correction. The first one is ARQ (Automatic Repeat Request) is a technique in which the receiver detects the occurrence of an error and ask the sender to resend the message. Resending is repeated until a message arrives that the receiver believes is error free. This method of error correction requires a two way channel and is not a feasible method for one way systems.

Another way of error correction is FEC (Forward Error Correction) . In this method, the receiver tries to guess the message by using redundant bits. This scheme adds redundant bits to a message in the form of extra bits called parity bits. Using this redundant information, the receiver is able to detect and correct a message without requesting for a retransmission.

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual message bits. The receiver checks this relationship between the two sets of bits to detect errors and correction too.

LDPC (Low Density Parity Check) codes are a powerful FEC coding scheme. LDPC codes were developed in 1960 by Robert Gallager at MIT in his PhD thesis. LDPC codes were forgotten until his work was rediscovered in 1996 and gradually it became the coding scheme of choice in the late 1990s, used for applications such as the Deep Space Network and satellite communications.

1.1 Error Detection and Correction

Networks must be able to transfer data from one device to another with acceptable accuracy. For most applications , a system must guarantee that the data received are identical to the data transmitted. Any time data transmitted from one node to the other, they can become corrupted in passage. Many factors can alter one or more bits of a message. If due to presence of noise the message gets altered, the receiver should be able to detect and correct if there is any error due to noise in thethe message received at the other end. To be able to detect or correct error, redundancy is used. Redundancy is extra bits which is appended to the message to be sent by the sender and these extra bits i.e. the redundant bits are used by the receiver for error detection and correction. On obtaining the correct or intended message the redundant bits are removed and message is extracted by the receiver.

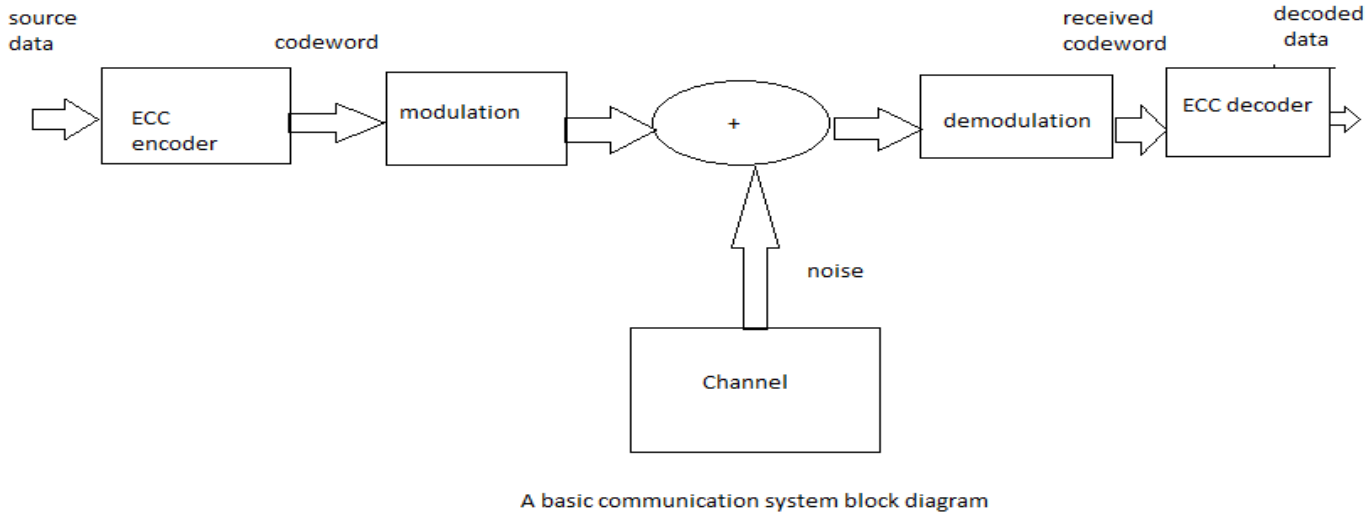


Fig.1. A basic communication system block diagram

1.2 Types of error

Whenever bits flow from one point to another, they are subjected to unpredictable changes because of interference. This interference can change the shape of the signal. In a single bit error, a 0 is changed to 1 or a 1 is changed to 0. In burst error, multiple bits are changed.

1.2.1 Single Bit Error

The term single bit error means that only 1 bit of a given data unit is changed from 1 to 0 or from 0 to 1.

To understand the impact let us assume that a data unit of 8 bits (0000**0**010) was sent, but (0000**1**010) was received. Clearly, the fourth bit from the left of

the data unit got corrupted during transmission. Fig.2 shows the effect of a single bit error on a data unit.

sent **00000010**

received **00001010**

Fig.2 Single bit error

1.2.2 Burst Error

The term Burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Fig.3 shows the effect of a burst error on a data unit. In this case, (010**00**10**0**01**0**00011) was sent, but (010**11**10**1**01**1**00011) was received. A burst error does not necessarily mean that the errors occur in consecutive bits. The length of the burst is measured from the first corrupted bit ti the last corrupted bit.

Sent **0100010001000011**

Received **0101110101100011**

Fig.3 Burst error of length 8

A burst error is more likely to occur than a single-bit error. The duration of noise is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits.

The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at 1 kbps, a noise of 1/100 s can affect 10 bits ; if we are sending data at 1 Mbps, the same noise can affect 10.000 bits.

1.3 Redundancy

The central concept in detecting or correcting errors is redundancy. To be able todetect or correct errors, we need to send some extra bits with our data. These extra bits are called redundant bits and these are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

1.4 Detection Versus Correction

The correction of errors is more difficult than the detection. In error detection, We only look to see if there is any error or not. The answer is simple yes or no. We are not even interested in the number of errors. A single bit error in that case is no different from a burst error because in error detection we are just bothered about the presence of error, the position or length of error cannot be detected in that case.

In error correction, we need to know exact number of bits that are corrupted and more importantly, their position/location in the message. The number of errors and the size of message are important factors. If we need to correct a single error in an 8 bit data unit, we need to consider eight possible error locations.

1.5 Forward error correction Versus Retransmission

There are two main methods of error correction, Forward error correction is the process in which the receiver tries to guess the message by using redundant bits. Whereas correction by transmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message until a message arrives that the receiver believes is error free.

1.6 Coding

Redundancy is achieved through various coding schemes. The sender adds the redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationship between the two sets of bits to detect and correct the errors. The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme. We can broadly classify the coding schemes in two groups (1) block coding (2) convolutional coding. In this thesis we have discussed about LDPC codes which belongs to a forward error correction block coding scheme.

Chapter 2

Background of LDPC codes

2. Background of LDPC codes

An (r,s) LDPC code is a Linear Block code with a check matrix H, satisfying the condition that every column of the matrix has r ones and every row has s ones.. An LDPC code with small values of r and s , has a sparse parity check matrix with very few ones in each row and column. Typically, r $\leq \log_2 n$, where n is the blocklength.

2.1 Block Coding

In block coding , messages are divided into blocks of $\boldsymbol{k}$ bits , called datawords. Then redundant bits are added to each block to make the length $\boldsymbol{n} = \boldsymbol{k}+\boldsymbol{r}$, where $\boldsymbol{r}$ is the redundant bits. The resulting $\boldsymbol{n}$ bit blocks are called Codewords . With k bits, a combination of $\mathbf{2}^{\boldsymbol{k}}$ datawords can be created ; similarly with n bits, a combination of $\mathbf{2}^{\boldsymbol{n}}$ Codewords can be generated.

2.2Linear Block Coding

A block code is linear if and only if exclusive-or (modulo-2 addition) of two valid codes creates another code. This property allows to create a generator matrix G that defines the code. All linear block codes including LDPC uses an encoder matrix called generator matrix G to add redundant information to the dataword to generate the codeword. These redundant bits are called parity bits and are appended to the dataword to get the codeword.

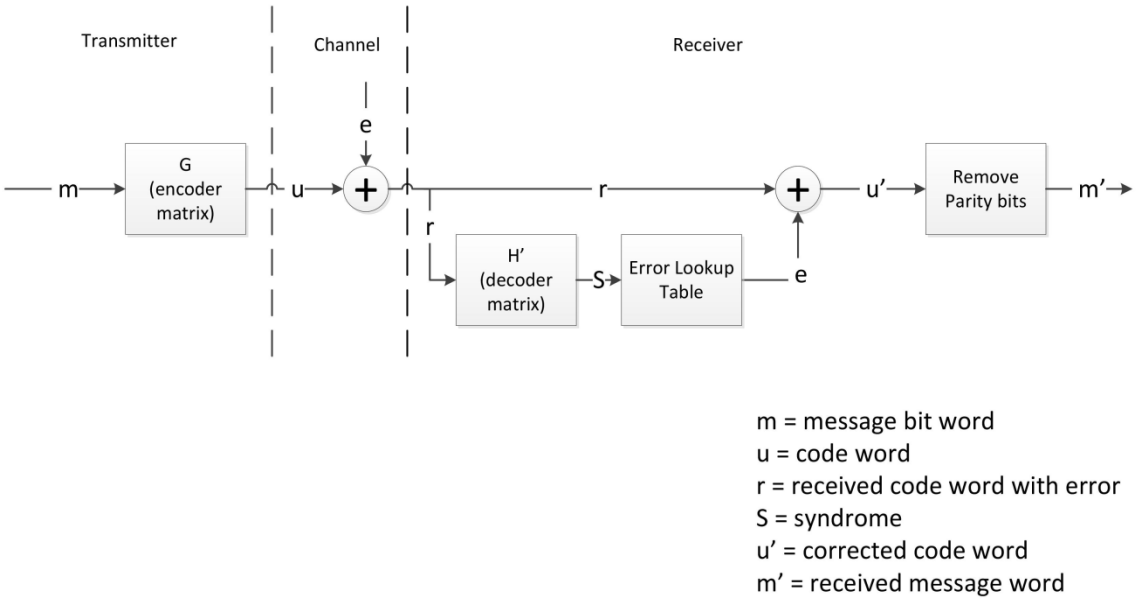


Fig.4 Block diagram of linear code encoder and decoder.

A block code is linear if and only if exclusive-or (modulo-2 addition) of two valid codes creates another code. This property allows to create a generator matrix G that defines the code.

A linear block code is generally denoted by ***(n,k)*** notation, where k denotes the number of bits in the message and n denotes the number of bits transmitted in the codeword. All linear block codes including LDPC uses an encoder matrix called generator matrix G to add redundant information to the dataword to generate the codeword. These redundant bits are called parity bits and are appended to the dataword to get the codeword. LDPC codes are class of linear block codes. The

name comes from the characteristic of their parity check matrix contains very few non zero elements here 1 than zeros.

2.3 Generator Matrix *'G'*

The generator matrix ***G*** is made up of ***k*** linearly independent row vectors of size ***n***. At the encoder end, the codeword that is to be transmitted is obtained by multiplying the message(dataword) with the generator matrix G. If ***u*** be the dataword and ***c*** be the codeword then,

$$\boldsymbol{c = u\,G}.$$

Since the encoder need to store only ***G*** matrix , it reduces the space complexity from $\boldsymbol{2^k \times n}$ (arbitrary encoding) to $\boldsymbol{k \times n}$ (size of G matrix). The generator matrix can be put in a systematic form as

$$\boldsymbol{G = [\, I_k l\, P\,]}$$

where P is an (n-k)×k sub matrix and I is identity matrix. From generator matrix, a parity check matrix ***H*** is generated which is udes at the decoder end.

2.4 Parity check matrix '*H*'

From generator matrix G , parity check matrix H can be obtained. The two matrices are related as

$$\boldsymbol{GH^T = 0}$$

If G= [I_k l P], then ***H*** obtained can be given as

$$H = [- P^T \; I \; I_{n-k} \;]$$

At the decoder end if y be the received message, it is checked if it is error free or not by using this parity check matrix H.

The received message y will be the codeword if

$$yH^T = S = 0$$

This operation results in a value called the syndrome (***S***). If the syndrome is equal to zero, there are no errors detected.

If the syndrome is not zero, the value is used to refer to a look-up table listing error patterns that result in the given syndrome. When the bit error pattern is identified, it is used to correct the corrupted codeword by flipping the incorrect bits. Finally, the parity bits generated by the encoder matrix G are removed and the result is the corrected message.

2.5 Low Density Parity Check(LDPC) codes

LDPC codes are class of linear block codes. The name comes from the characteristic of their parity check matrix contains very few non zero elements here 1 than zeros.

The BER performance of LDPC codes are heavily dependent on the parity check 'H' matrix. Sparser the H matrix ,the lower will be the complexity of the decoder. For this reason ,many algorithms has been developed H matrix such that maximum BER performance is obtained and also complexity of decoder is reduced.

2.5.1 Representation of LDPC codes

Although LDPC codes are defined by a sparse matrix, a bipartite graph also called Tanner graph is also used to represent LDPC codes.

A Tanner graph is a graphical representation of the linear code based on the set of check equations. It is a bipartite graph because it has two sets of nodes : symbol nodes/variable nodes(V-nodes) and check nodes(C-nodes). Each symbol node is connected only to the check nodes and similarly, each check node is connected to only symbol nodes.

Each component of the codeword is represented in the Tanner graph by a symbol node(V-nodes) . Thus, there are ***n*** V-nodes. Each check equation of the code is represented in the Tanner graph by check nodes(C-nodes). Thus, there are ***n-k*** C-nodes in the Tanner graph. Each check node is connected by an edge to those symbols that it checks.

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \quad (1)$$

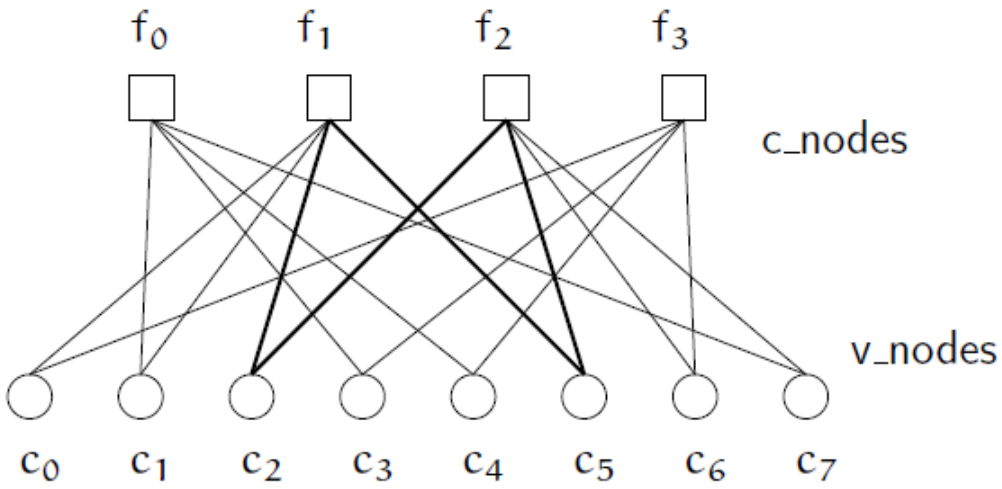


Fig.5 Tanner graph corresponding to the parity check matrix in equation (1)

Tanner graph not only helps in representation of the LDPC codes completely, but also helps to describe the decoding algorithms discussed later in this thesis.

2.5.2 Regular and Irregular LDPC codes

If ‘r’ be the number of 1’s in every column(column weight) and ‘s’ be the number of 1’s in every row(row weight) , An LDPC code with fixed ‘r’ and ‘s’ is called a regular LDPC code. If the number of ones columns and that in the rows are approximately r and s respectively, it is then called an Irregular LDPC code.

The H matrix in equation (1) has r=2 and s=4.

The regularity of LDPC codes can also be observed from its Tanner graph representation. If in Tanner graph representation of an LDPC code, the V-nodes are having equal number of incoming edges and the C-nodes also having equal number of incoming edges , then the LDPC code is regular.

2.6 Project Approach

After gaining a fundamental understanding of LDPC codes, the step taken was to perform and analyze the LDPC system. It included the encoder design in circuit maker, MATLAB implementation of LDPC decoder. This project was initially broken into three phases. In the first phase, the goal was to obtain a good understanding of LDPC codes and encoding-decoding techniques using circuit maker. In the second phase, the decoding algorithm was implemented using MATLAB. In the final phase, it was suppose to be ported to FPGA.

Chapter 3

LDPC encoder

3.LDPC Encoder

3.1 Constructing LDPC Codes

LDPC code construction requires a knowledge of the pattern in which the rows and columns of the parity check or the check nodes and variable nodes of corresponding tanner graph are connected. The main objective in code construction are good decoding performance and easier hardware implementation. This is mainly achieved by having row-column connections that have a regular pattern. There are various methods for constructing LDPC code of a given length and rate. LDPC codes can be constructed by using random constructions (unstructured row-column connection) or structured connection (row-column connection predefined) Or a combination of both.

3.2 Random constructions

In random construction method, the rows and columns (check nodes and variable nodes) are connected without any structure or predefined connection pattern. Constructions could be done by randomly adding edges to a tanner graph or by '1' entries in the corresponding parity check matrix. But this will not produce desired rate and probably lead to formation of cycle in the graph. But the resulting codes can be optimized by putting some constraints on making random choices when the code is built. Random construction with constraints add a connection if and only if it does not violate the desired row column weights.

Random constructions have flexibility in the design and construction but lack row column connections regularity, which increases decoder interconnection complexity.

3.2.1(a) Algorithm 1

Step1: Set i=1.

Step2: Generate a random binary vector of length $\frac{nr}{s}$ and a Hamming weight r. This will be the i^{th} column of H.

Step3: If the weight of each row of H at this point is $\leq$ s , and the scalar product of each pair of columns is ≤ 1 , set i =i+1.

Else , go to Step2.

Step4: If i=n , stop.

Else, go to Step2.

3.2.1(b) Other random construction algorithms

1. Bit flipping algorithm

H matrix is generated starting with all zero elements in the matrix then randomly flipping bits in the matrix.

This algorithm constructs an LDPC code connecting rows and columns of a code, one at a time making sure that the desired girth or row column weights are not violated.

2.Progressive edge growth algorithm

It is another non algebraic random construction algorithm . It is similar to bit flipping algorithm.

This algorithm builds a tanner graph by connecting nodes edge by edge provided the edge added has minimal impact on the girth og the graph.

With this algorithm, codes can be obtained with optimized performance.

Like bit flipping algorithm, the major disadvantage with this algorithm is their hardware complexity making them impractical at very high lengths.

The random and unstructured constructions results in routing complexity and congestion in decoder implementations.

3.3 Structured constructions

The parity check matrix H could be constrained algebraically to obtain codes with a particular structure.

Step1: p> $\frac{r-1}{s-1}$

Step2: Construct a $p \times p$ matrix 'j' from the identity matrix I_p by cyclically shifting its rows by one position to the right.

Step3: The parity check matrix obtained is

$$\begin{bmatrix} j^0 & \cdots & j^0 \\ \vdots & \ddots & \vdots \\ j^0 & \cdots & j^{((r-1)(s-1))} \end{bmatrix}$$

Where $j^0 = I_p$ and the l^{th} power of j is obtained from I_p by cyclically shifting its rows by l mod p positions to the right. This matrix has exactly r ones in each column and s ones in each row. Thus it is a regular or structured LDPC code.

Chapter 4

LDPC encoder implementation

4.a Encoder design

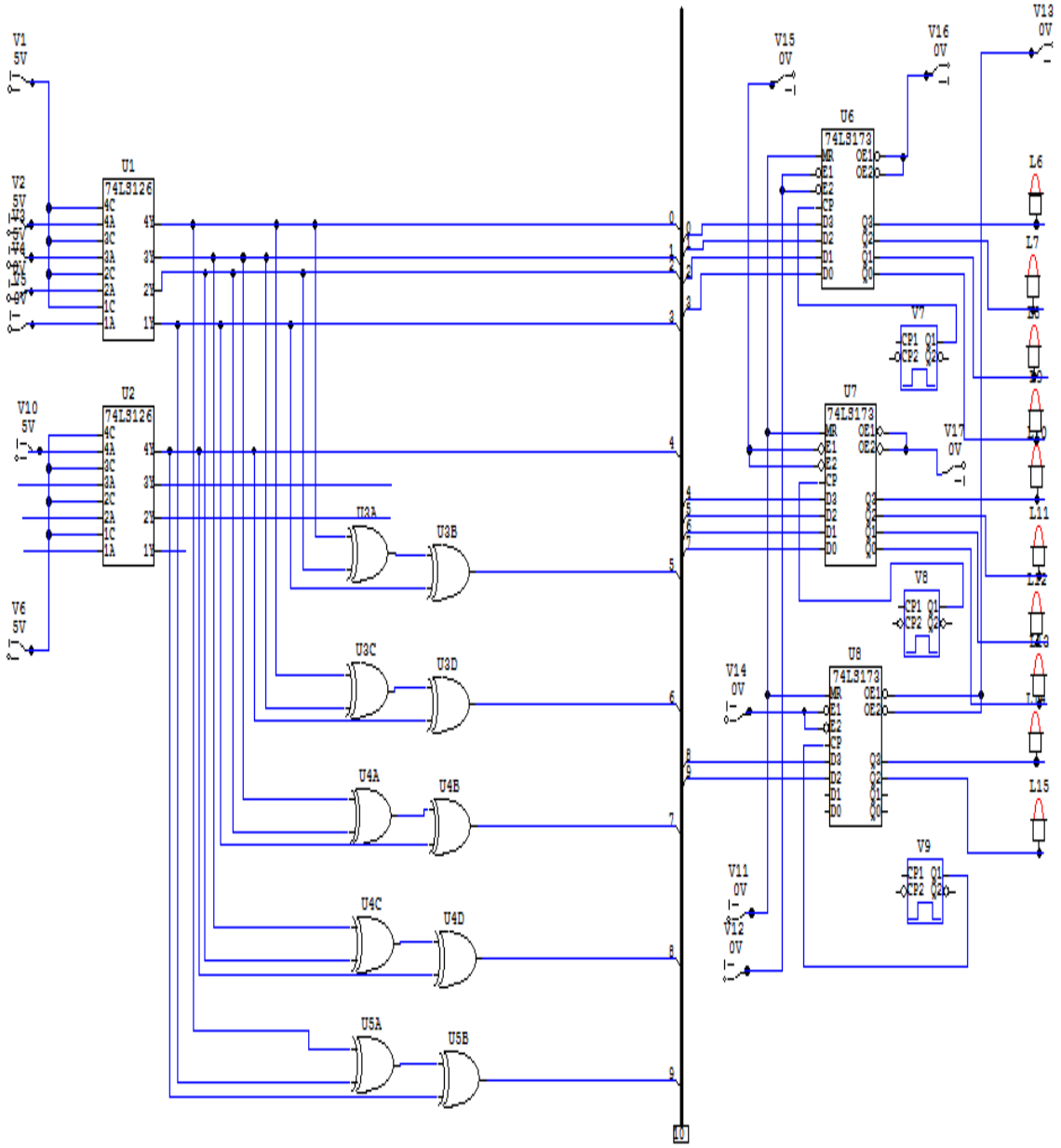


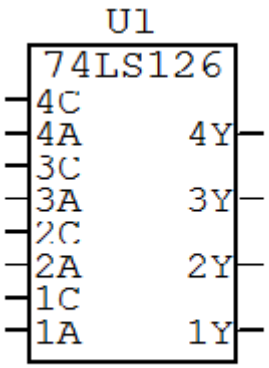
Fig.6 Low level Encoder implementation

4.a.1 Encoder design implementation

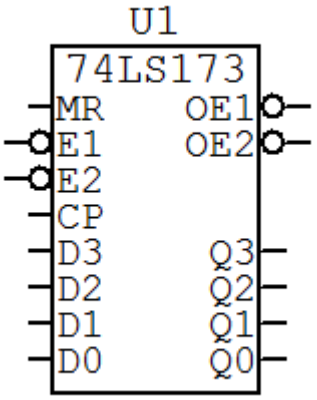
After gaining fundamental understanding about LDPC constructions, a LDPC encoder design is developed using circuit maker. This section deals with some basic ICs and combinational GATEs. The ICs/GATEs used to develop the design has been listed below.

Blocks used:

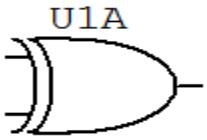
1. 74LS126- LS quad tri state buffer.



2. 74LS173-LS qyad D type tri state flipflop.



3. 2 input XOR gates.



4. LEDs.

As can be seen in Fig.6, Each bits of the dataword is fed to a temporary register(here 74LS126 IC buffer is used to store the input bits).It is basically desired to multiply the message bits i.e. the dataword with the generator matrix G to obtain the codeword that would transmitted to the receiver end. This is followed by modulo-2 operation (XORing) of message bits to obtain the redundant bits that would be appended to the dataword to get the desired codeword.

For the dataword [11001] the corresponding codeword obtained is [1100111100].The corresponding operation is shown in Fig.7 in the next page.

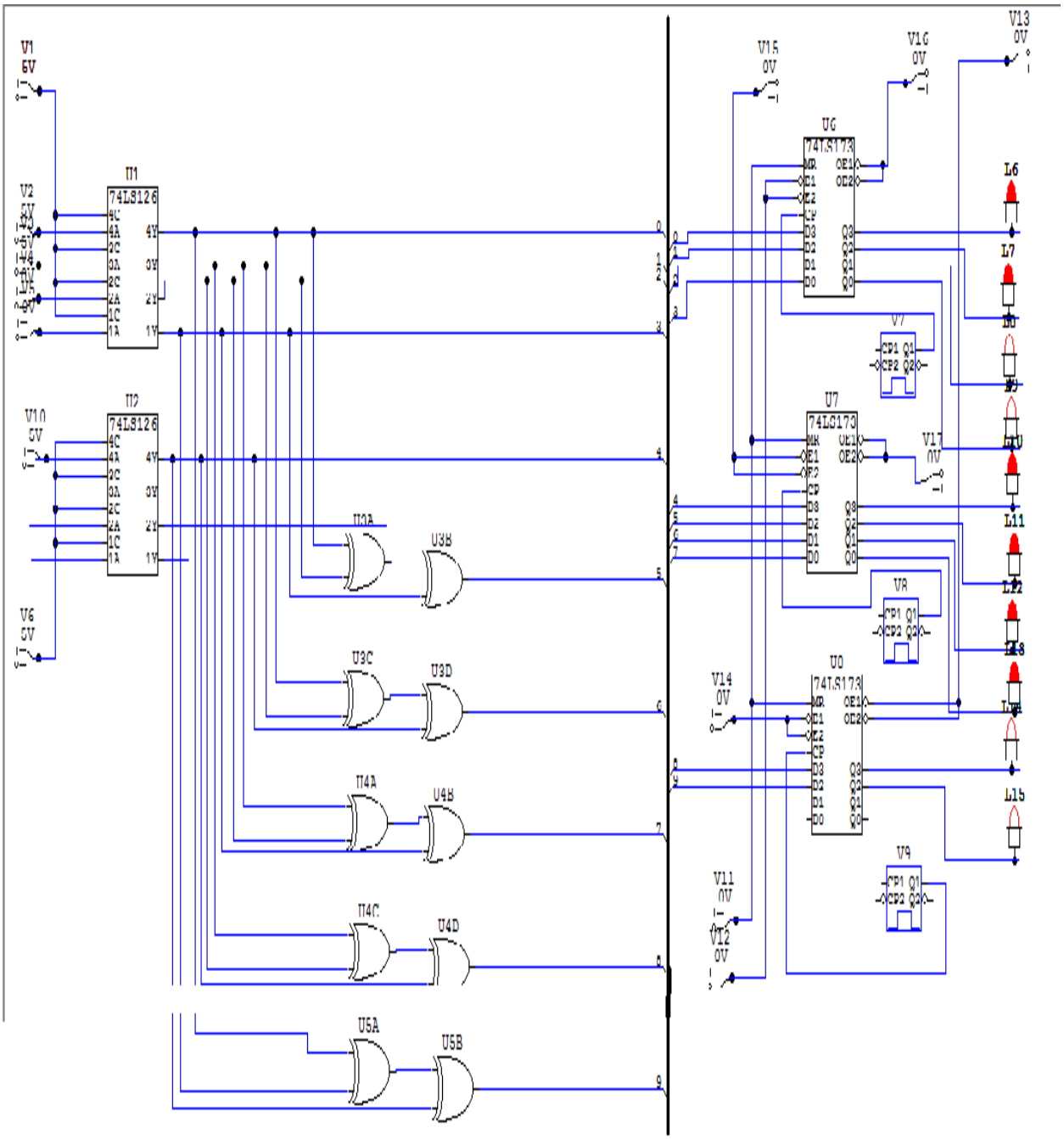
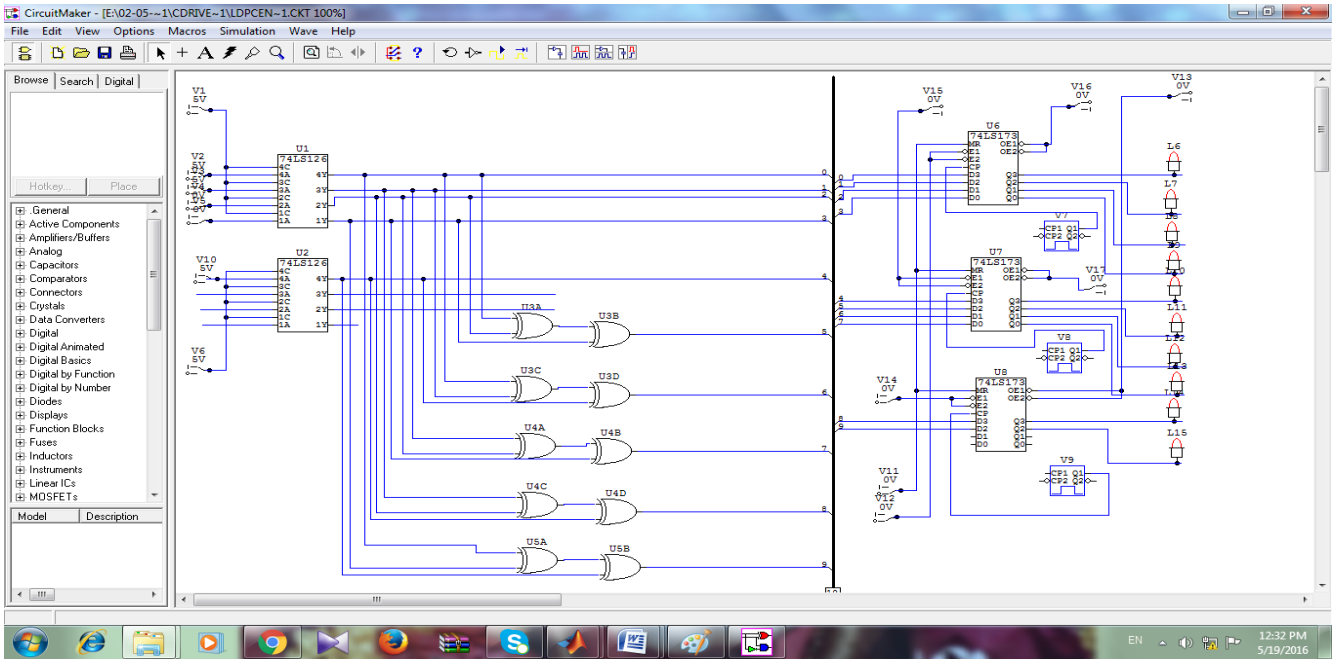


Fig.7 For message bits [11001], the corresponding codeword



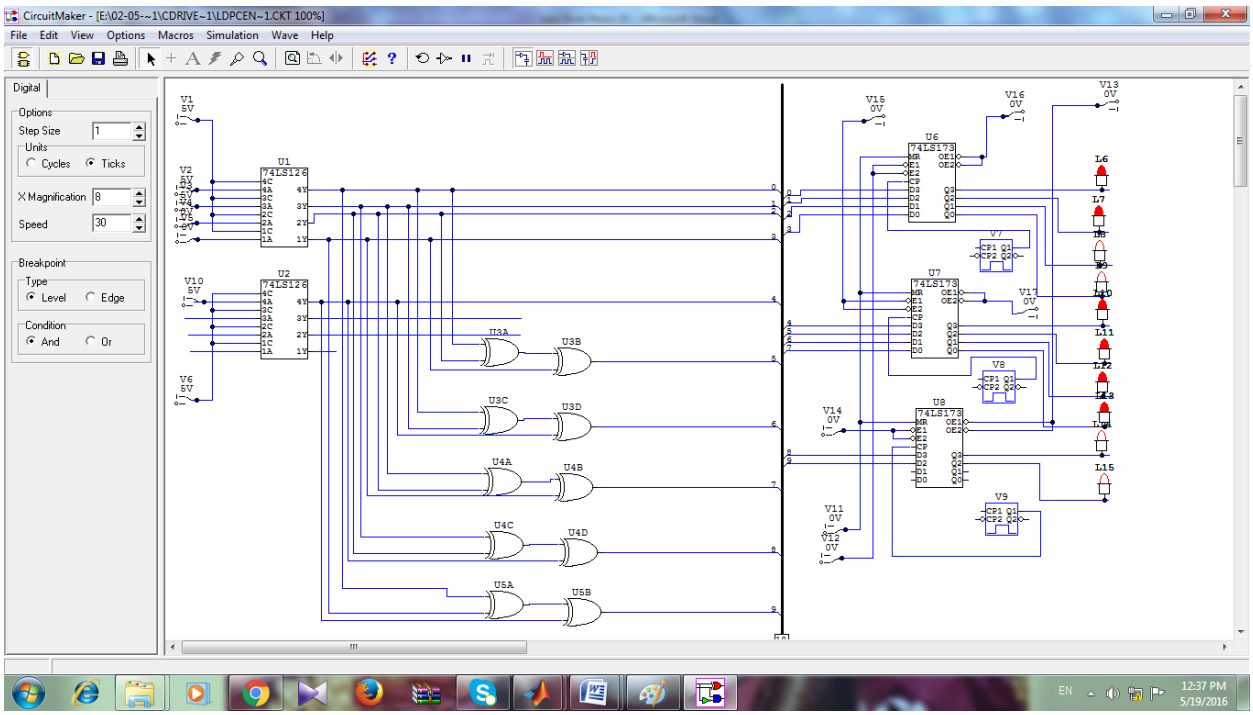


Fig. 8 LDPC encoder using circuit maker

4.b FPGA implementation of LDPC codes

4.b.1 Introduction of Verilog

Now let's look to Verilog, using which I have tried to implement the hardware. **Verilog**, standardized as **IEEE 1364**, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits.

Hardware description languages such as Verilog differ from software programming languages because they include ways of describing the propagation time and signal strengths (sensitivity). There are two types of assignment operators; a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables.

Like C, Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible with C. Syntactic differences include: required bit-widths for variable declarations, demarcation of procedural blocks (Verilog uses begin/end instead of curly braces {}), and many other minor differences. Verilog requires that variables be given a definite size. In **C** these sizes are assumed from the 'type' of the variable (for instance an integer type may be 8 bits).

A Verilog design consists of a hierarchy of modules. Modules encapsulate *design hierarchy*, and communicate with other modules through a set of declared input,

output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent

and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and signal strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

A subset of statements in the Verilog language are synthesizable. Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a netlist, a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific FPGA or VLSI technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blueprint (such as a photo mask set for an ASIC or a bitstream file for an FPGA).

4.b.2 Reasons of using Verilog

a) Verilog is an international IEEE standard specification language for describing digital hardware used by industries Worldwide.

b) Verilog enables hardware modelingfrom the gate to system level.

c) Verilog provides a mechanism for digital design and reusable design documentation.

4.b.3 Capability

a) The language can be used as a medium between the chip vendor and the CAD tool users.

b) The language supports hierarchy; that is a digital system is modeled as a set of interconnected components; each component in turn can be modeled as a set of interconnected subcomponent.

c) The language supports flexible design methodology.

d) It supports synchronous and asynchronous timing models.

e) It supports 3 different description styles: structural, dataflow and behavioral.

f) Different kind of delays, timing constraints and spike can be described very naturally using this language.

4.b.4 Various stages of FPGA

The various stages of FPGA has been discussed in the next page. 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.

The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit.

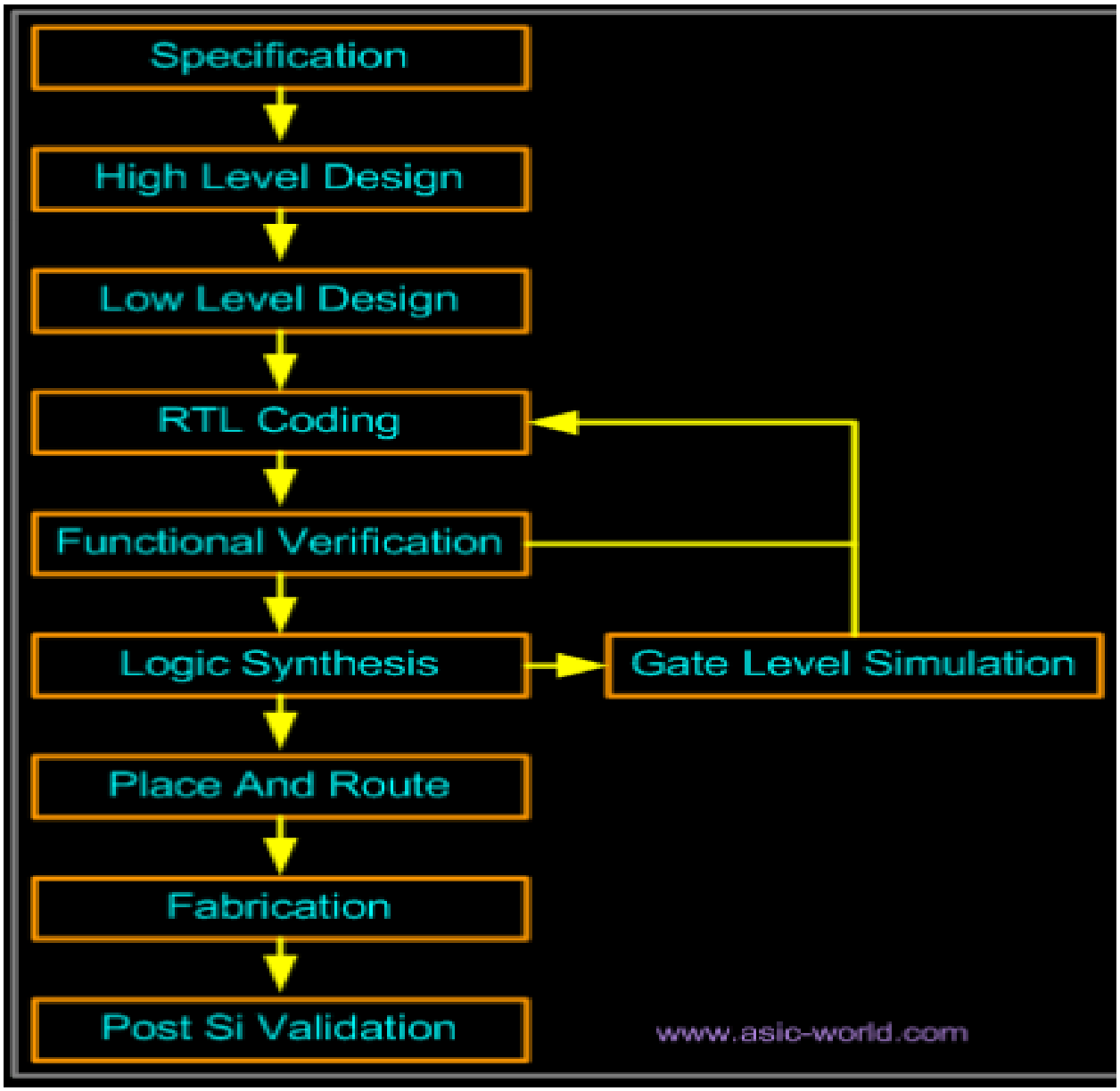


Fig.9 Stages of FPG

4.b.5 Test Bench

The correctness of a program can be checked by writing test bench. **Test benches** are used to simulate your design without the need of any physical hardware. The biggest benefit of this is that you can actually inspect every signal that is in your design.

4.b.6 Verilog implementation of LDPC encoder and result

The Verilog encoder implementation involves multiplication of the message matrix with the generator matrix. In this project the size of the generator matrix used is 5×10 , so the message dimension is 5 bits.

Step1: Taking the required number of registers for storing the two matrices.

Step2: The matrix multiplication is written between the message matrix (1× 5) and the generator matrix (5×10). This will be combination of ‘AND’ and ‘OR’ gates.

Step3: The message bits are transmitted through test bench.

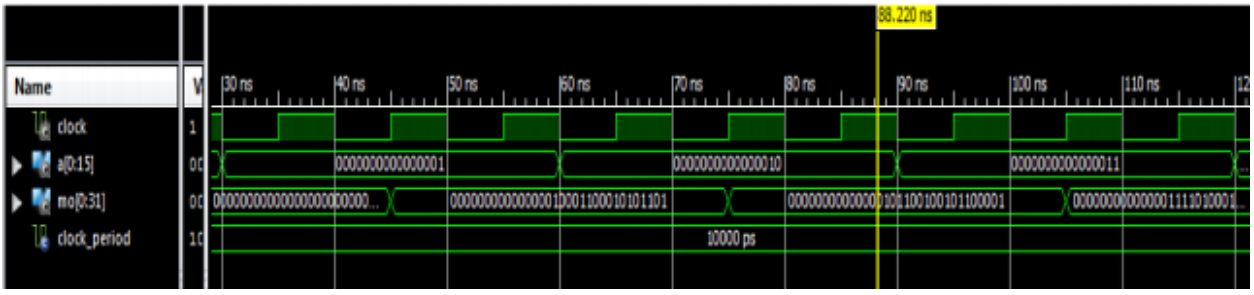


Fig.10 Encoder test bench wave-form

Chapter 5

LDPC decoder

5. LDPC Decoder

LDPC code decoding tries to reconstruct the transmitted, c, from the possibly corrupted received word ,y. It is achieved by using parity check matrix, H. The condition that $yH^T=0$ defines the set of parity check constraints that must be satisfied for the received codeword (y) to be same as the transmitted codeword (c).

5.1 LDPC Hard-decision Decoding Algorithms

Most commonly used decoding algorithms are belief propagation algorithm, message passing algorithm, and the sum product algorithm.

Before going to these algorithm a simple hard decision algorithm is discussed to have a better knowledge of decoding theory n technique.

5.1.1 Bit flipping algorithm

An LDPC code can be decoded using simple iterative bit flipping algorithm. This algorithm can be summarized in the following steps:

- Perform $S= yH^T$. Where y is the received codeword and S is the syndrome.
- Compute all check sums and the number of unsatisfied parity checks involving each bits of y.
- Flip those bits of y which are involved in largest number of unsatisfied parity checks.
- Go back to the third step. Keep iterating until either-(a) All checks are satisfied, or (b) A predetermined number of iterations are reached.

The flipping algorithm does not guarantee that errors up to half the minimum distance are corrected. However, for large block lengths , this sub optimum algorithm works remarkably well.

The decision of which variable bit is to be flipped using a simple algorithm explained below in the next part.

The algorithm is explained on the basis of the example code shown in equation (1) and the corresponding tanner graph in Fig.1.

An error free received codeword would be e.g. c = [1 0 0 1 0 1 0 1]. Let's suppose that we have a BHC channel and the received the codeword with one error bit c1 flipped to 1.

1. In the first step all v-nodes ci send a "message" to their c-nodes fj containing the bit they believe to be the correct one for them. At this stage the only information a v-node ci has, is the corresponding received i-th bit of c, yi.

That means for example, that c0 sends a message containing 1 to f1 and f3, node c1 sends messages containing y1 (1) to f0 and f1, and so on.

Here f represents all check nodes and c represents the message bits received. All received message bits are fed to the variable nodes. All variable nodes pass this bit to the connect check nodes. The next step is discussed in the next page.

The table below shows how every variable node sends a message that contains the bit which it believes to be correct to all the check nodes it is connected to. In return it receives a response message from the connected check nodes which contains the bit that check node believes to be correct.

C-node	Received
f 0	received: c1 =1 , c3 =1, c4 =0, c7 =1 sent: 0=c1, 0=c3, 1=c4, 0= c7
f 1	received: c0 =1, c1 =1, c2 =0, c5 =1 sent: 0=c0, 0=c1, 1=c2, 0=c5
f 2	received: c2=0, c5=1, c6=0, c7=1 sent: 0=c2, 1=c5, 0=c6, 1=c7
f 3	received: c0=1, c3=1, c4=0, c6=0 sent: 1=c0, 1=c3, 0=c4, 0=c6

Table 1 : overview over messages received and sent by the c-nodes in step 2 of the message passing algorithm.

After receiving the response message the variable nodes compares the two messages and decides whether to retain the bit or flip it. This mainly done by voting/maximum possibility. This method is discussed in details in the next step.

2. In the second step every check nodes fj calculate a response to every connected variable node. The response message contains the bit that fj believes to be the correct one for this v-node ci assuming that the other v-nodes connected to fj are correct.In other words: In the above discussed example, every c-node fj is connected to 4 v nodes. So a c-node fj looks at the message received from three v-nodes and calculates the bit that the fourth v-node should have in order to fulfill the parity check equation.Table 2 gives an overview about this step.

3. Next phase: the v-nodes receive the messages from the check nodes and use this additional information to decide if their originally received bit is OK. A simple way to do this is a majority vote. When coming back to our example that means, that each v-node has three sources of information concerning its bit. The original bit received and two suggestions from the check nodes. Table 2 illustrates this step.

4. Go to step 2.

In the above example, the second execution of step 2 would terminate the decoding process since c1 has voted for 0 in the last step. This corrects the transmission errors and all check equations are satisfied.

v-node	Message(y) received	Message from the check nodes	Decision (c)
c 0	1	f 1=0 f 3=1	1
c 1	1	f 0=0 f 1=0	0
c 2	0	f 1=1 f 2=0	0
c 3	1	f 0=0 f 3=1	1
c 4	0	f 0=1 f 3=0	0
c 5	1	f 1=0 f 2=1	1
c 6	0	f 2=0 f 3=0	0
c 7	1	f 0=1 f 2=1	1

Table 2: Step 3 of the described decoding algorithm. The v-nodes use the answer messages from the c-nodes to perform a majority vote on the bit value.

5.2 LDPC soft-decision decoding algorithm(Belief Propagation Algorithm)

The above description of hard-decision decoding was mainly for the purpose to get an overview about the idea. Soft-decision decoding of LDPC codes, which is based on the concept of belief
propagation, yields a better decoding performance. The underlying idea is exactly the same as in propagation hard decision decoding. Before presenting the algorithm lets introduce some notations:

• $P_i = Pr(c_i = 1|y_i)$

• q_{ij} is a message sent by the variable node c_i to the check node f_j. Every message contains always the pair $q_{ij}(0)$ and $q_{ij}(1)$ which stands for the amount of belief that y_i is a ”0” or a ”1”.

• r_{ji} is a message sent by the check node f_j to the variable node c_i. Again there is a $r_{ji}(0)$ and $r_{ji}(1)$ that indicates the (current) amount of believe in that y_i is a ”0” or a ”1”.

The step numbers in the following description correspond to the
hard decision case.

1. All variable nodes send their q_{ij} messages. Since no other information is avaiable at this step, $q_{ij}(1) = P_i$ and $q_{ij}(0) = 1 - P_i$.

2. The check nodes calculate their response messages rji.

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in v_{j/i}}(1 - 2q_{i'j}(1)) \quad (2)$$

$$r_{ji}(1) = 1 - r_{ji}(0) \quad (3)$$

So they calculate the probability that there is an even number of 1's among the variable nodes except ci (this is what Vj/i means). This propability is equal to the probability rji(0) that ci is a 0.

3. The variable nodes update their response messages to the check nodes. This is done according to the following equations,

$$q_{ij}(0) = k_{ij}\ (1 - p_i)\prod_{j' \in c_{i/j}} r_{j'i}(0) \quad (4)$$

$$q_{ij}(1) = k_{ij}\ p_i\ \prod_{j' \in c_{i/j}} r_{j'i}(1) \quad (5)$$

here the constant Kij is chosen in a way to ensure that qij(0)+qij(1) = 1. Ci/j now means all check nodes except fj.

At this point the v-nodes also update their current estimation $\widehat{c_i}$ of their variable ci. This is done by calculating the probabilities for 0 and 1 and voting for the bigger one. The used equations

$$Q_i(0) = K_i\ (1\text{-}\ p_i) \prod_{j \in c_i} r_{ji}(0)$$

(6)

and

$$Q_i(1) = K_i p_i \prod_{j \in c_i} r_{ji}(1)$$

(7)

are quite similar to the ones to compute qij(b) but now the information from every c-node is used.

$$\widehat{c_i} = 1 \text{ ; if } Q_i(1) > Q_i(0)$$

Else

(8)

$$\widehat{c_i} = 0.$$

If the current estimated codeword fulfils now the parity check equations the algorithm terminates. Otherwise termination is ensured through a maximum number of iterations.

Chapter 6

LDPC decoder implementation

6.1 Introduction

For most applications LDPC encoding and decoding is done using hardware to speed up processing. Successful application of LDPC codes to various systems depends mainly on encoders and decoders meeting cost, power, complexity and speed requirements of those systems. Applications differ on these requirements and LDPC performance expectations. The large size of LDPC codes, wide range of rates and unstructured interconnection patterns are some of the characteristics that make hardware implementation a challenge. Although decoder implementations are often targeted for a particular application, their architectures are often required to be scalable, programmable and have low chip area. Some applications require a wide range on some parameters such as rate and length. Also, application requirements may change from time to time. In such cases, it is desirable to have a flexible decoder that could easily be adapted to new requirements. The adaptability of hardware to new requirements may depend on its architecture.

LDPC decoder architectures differ mainly in the arrangement of check and variable processing nodes and their interconnections, message passing or scheduling, node implementations and number of nodes. The interconnection between nodes can be done using a variety of components such as memory blocks, buses and crossbar switches. Message scheduling can also take various forms. The most common or natural scheduling criteria is called flooding. In flooding all check node messages are sent to all variable nodes after computation and vice versa. Other scheduling methods include staggering in which only a fraction of nodes send on demand messages across. Staggering tries to reduce memory conflicts and improve computation units utilization. Depending on the interconnection network and storage of messages there might be memory access conflicts.

Because the network is very likely not to accommodate all messages at the same time, scheduling of messages is needed. During message transmission no computations are done which reduces computation nodes utilization. Decoding equations can be implemented in different forms, including approximations and look up tables. LDPC decoders are often classified according to the number of processing nodes in comparison to the size of the code. Below are the three classifications based on this criteria→

- **Number of Processing Nodes:**

1. Fully Parallel architectures: A fully parallel implementation of the decoder consists of mapping directly the symbol and check nodes in the Tanner graph to the respective symbol and check modules. The edges of the graph become physical buses of width equal to chosen precision. A fully parallel implementation, while efficient in speed point of view is demanding in terms of area, due to interconnect between the processing elements. Although the computations for calculating the check to symbol and symbol to check messages are not particularly complex and require a small area to be implemented, the massive number of interconnections in the graph lead to complex wiring. In fully parallel architecture the number and complexity of interconnects results in the implementation where almost 60% of the area is being dominated by wires. Moreover, the number of computational blocks required is in one to one relationship with the number of nodes in the Tanner graph. For medium or long code the resource demands and complexity of hardware implementation will become infeasible.

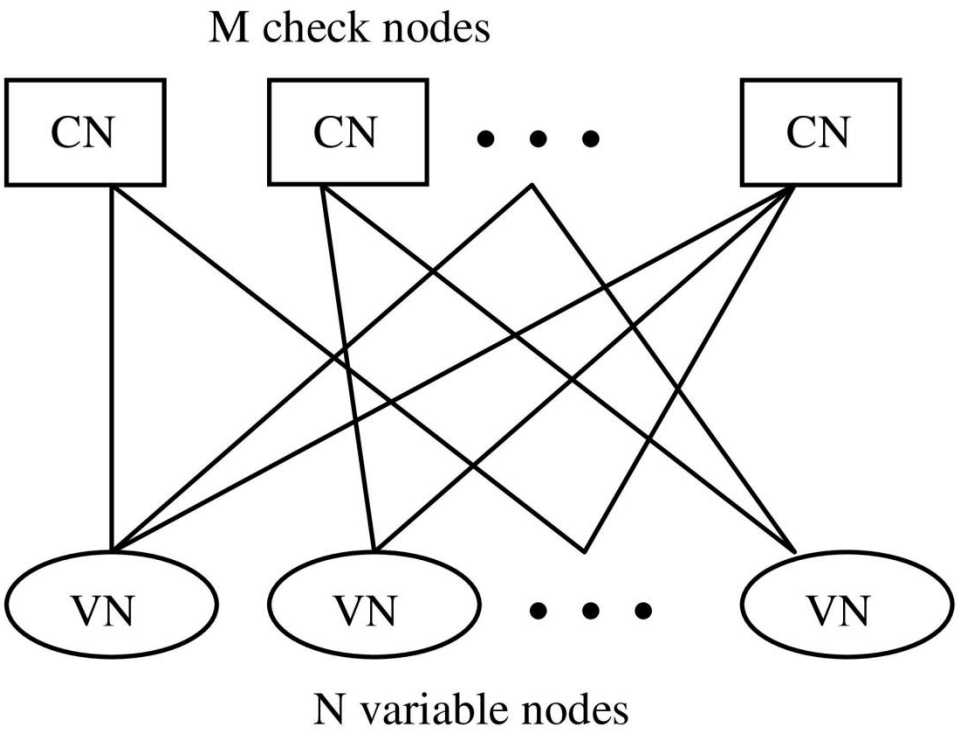


Fig.11 Fully parallel LDPC decoder architecture

2. Serial architectures: Fully parallel architectures instantiate each node and edge of the Tanner graph. The architecture requires maximum hardware with respect to the size of the code. The other extreme case would be to provide minimum hardware. Serial architectures have one check and one variable node computation units. The check node unit processes one row at a time and the variable node does the same with columns. As in fully parallel architectures the interconnection network could be bidirectional or unidirectional. c Any code structure could be executed on this architecture.

However, serial implementations would not meet time constraints of most applications which require high throughputs.

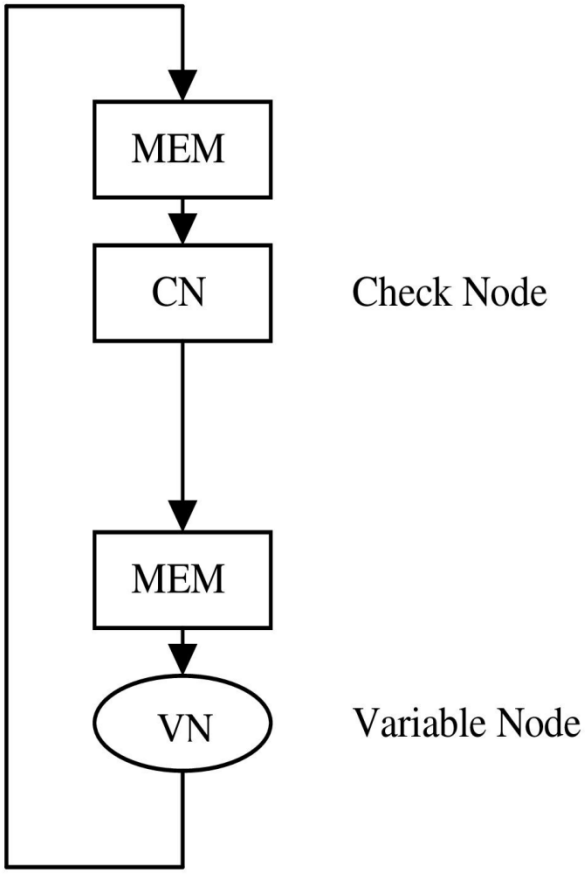


Fig.12 Serial LDPC decoder architecture with unidirectional connections.

3. Semi-parallel architectures: A trade-off between the routing problem typical of the parallel architecture and the reduction of the throughput characteristic of the serial architecture is to use a semi-parallel architecture. This type of architecture distributes the process across a number of computational blocks. The check and symbol node messages passed along the edges of the Tanner

graph are stored temporarily in a memory. This solution reduces both the complexity of the routing compared with the parallel architecture and the complexity in the memory control compared with the serial one. If codes with more regularity are chosen, this architecture can be designed to take advantage of the

structure of the parity check matrix, in order to simplify both memory control and interconnection network. These architectures generally achieve a good trade-off between hardware complexity and throughput but suffer from memory access conflicts. These conflicts occur when multiple data accesses per cycle are required for the same memory bank. These conflicts are difficult to avoid if the code implemented is of random nature but can be avoided for structured codes.

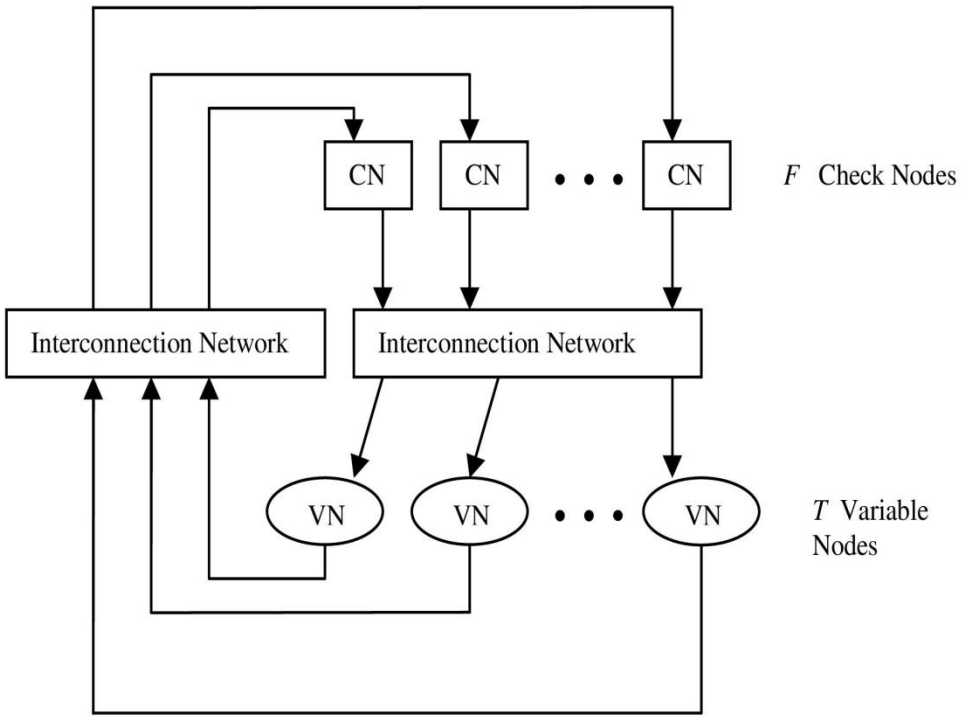


Fig.13 Semi-parallel LDPC decoder architecture with unidirectional connections.

6.2 Decoding

The decoding algorithms are generally message passing algorithms as a set of message is being exchanged between the variable nodes and the check nodes.

A node on the tanner graph uses only the informations given by the nodes connected to it. A basic decoding is done as follows

1. Every variable node sends the message to connected check nodes, this message contains the bit that they believe to be correct.

2. The check node calculate a response for every variable node connected to it. The check node while responding to a variable node assumes that the message received from other connected variable nodes as correct and sends a message in response which holds the bit that it believes to be correct.

3. The variable nodes receive the message from check node and uses this additional information to decide the correct bit.

4. Then each check equation is checked if they all satisfy then the message is correct, else the process is repeated.

5. The v-nodes receive the messages from the check nodes and use this additional information to decide if their originally received bit is OK. A simple way to do this is a majority vote. When coming back to our example that means, that each v-node has three sources of information concerning its bit. The original bit received and two suggestions from the check nodes.

6. If the current estimated codeword fulfils now the parity check equations the algorithm terminates. Otherwise termination is ensured through a maximum number of iterations.

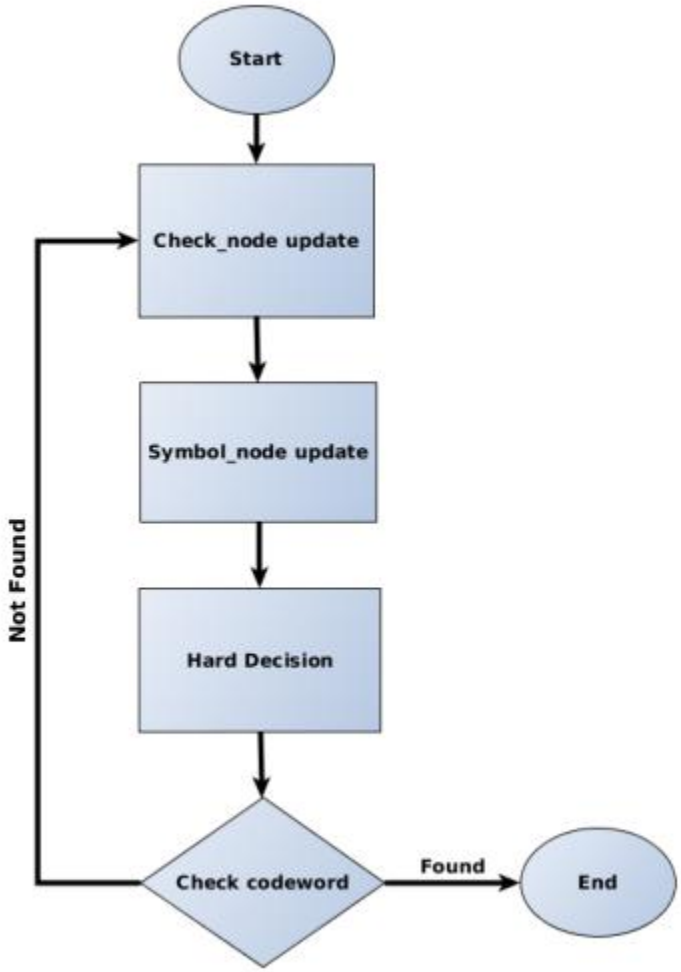


Fig.14 Decoding Flowchart

6.3 Decoder implementation using MATLAB and result

After gaining fundamental understanding of LDPC decoder algorithms and architecture, the next step taken was to perform and analyze a MATLAB implementation of belief propagation decoding algorithm of LDPC codes.

Getting the correct MATLAB code(in the Appendix) to use was the key in having a

working LDPC system and being able to verify its operation.

The decoding algorithm used in MATLAB(R2012b) implementation codes was first decided to be implemented in hardware. However, because decoding algorithm uses non linear functions, it was difficult to implement in the hardware.

There are basically four main blocks used in making up the decoder design-(1)the first block is the block that performs the calculations that happens at the variable nodes,(2)the next block is a memory block to store the edge values sent from the variable nodes to the check nodes,(3)then is the check node block where each check nodes update their current estimation,(4)the last block is where final decision is made.

The objective of this project was to explore the use of FPGA in the encoding and decoding process of LDPC codes. However the system could be implemented on MATLAB only and could not be tested on FPGA, this project stands to show the difficulties one may face when designing an LDPC system. This thesis also illustrates the performance criteria that are to be considered while developing an LDPC system.

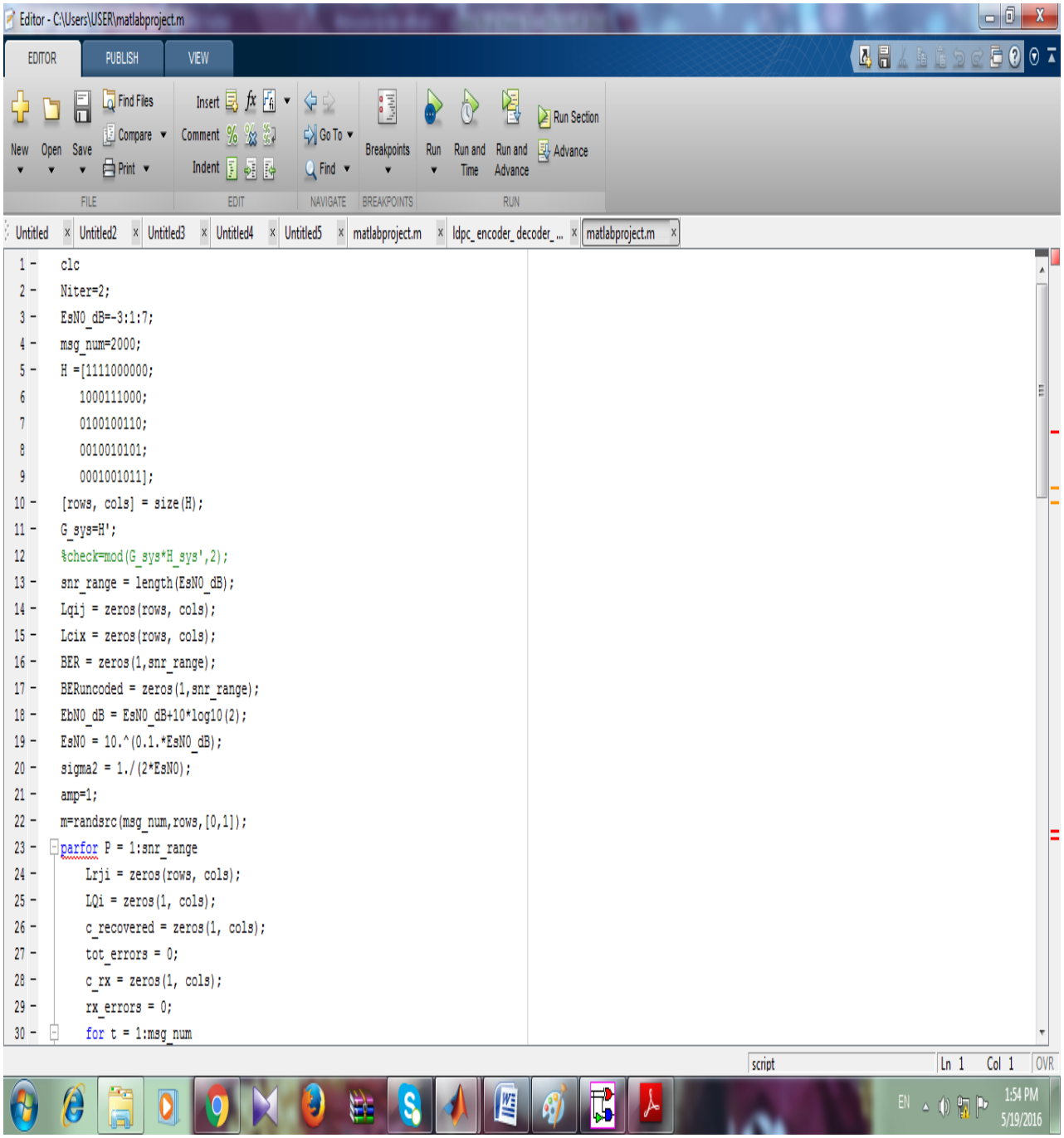


Fig.15 MATLAB implementation (1)

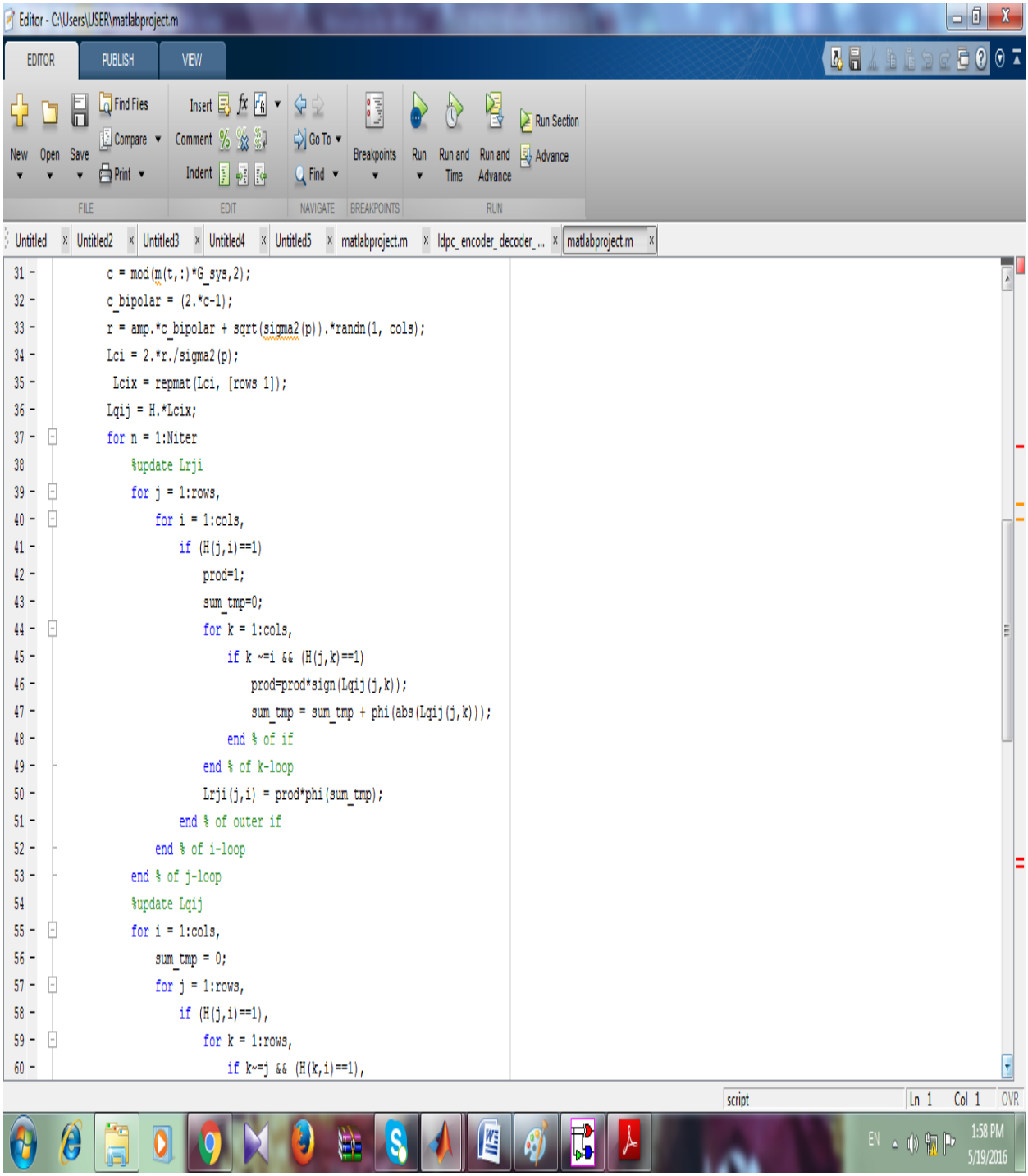


Fig.16 MATLAB implementation (2)

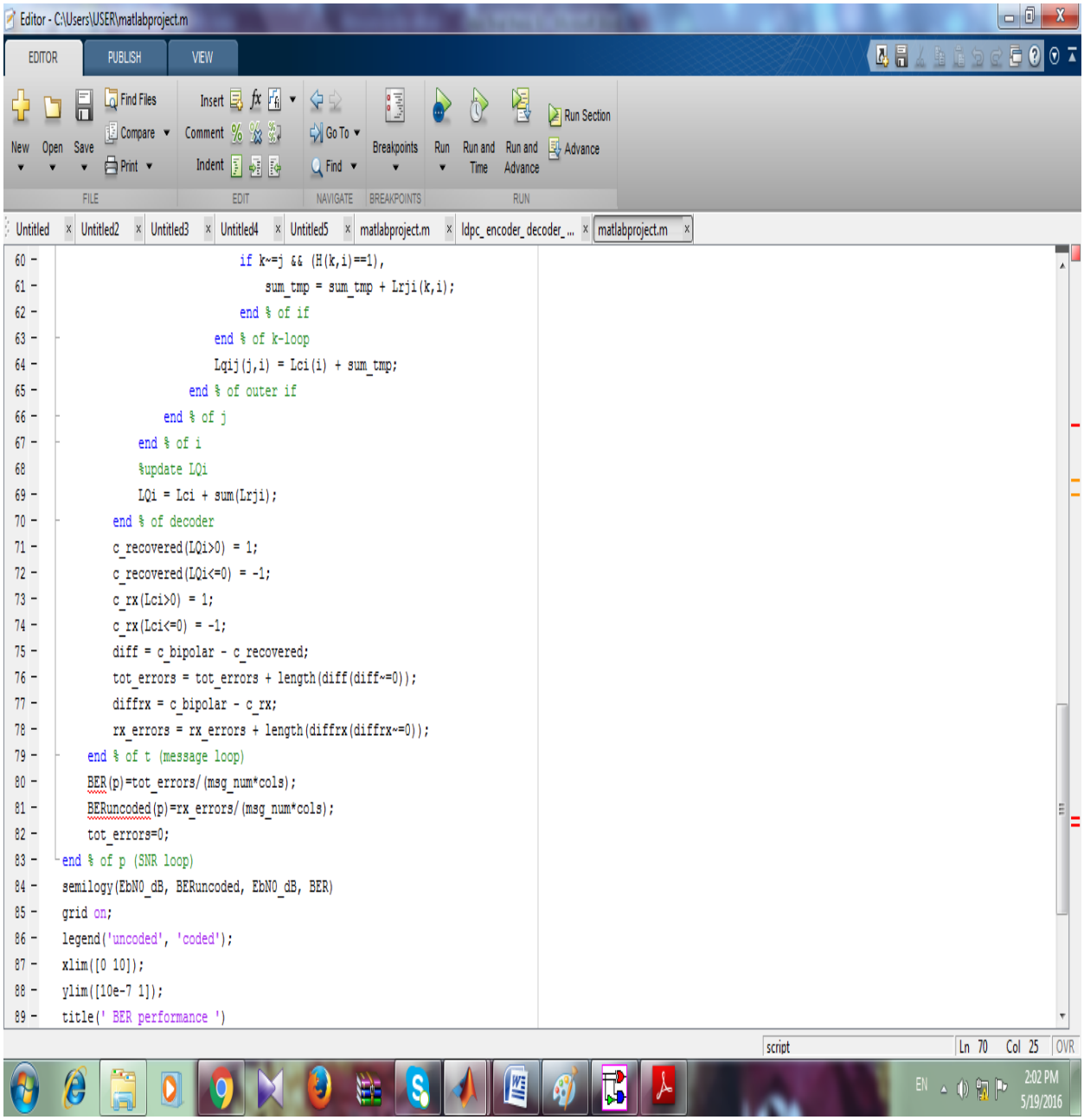


Fig.17 MATLAB implementation (3)

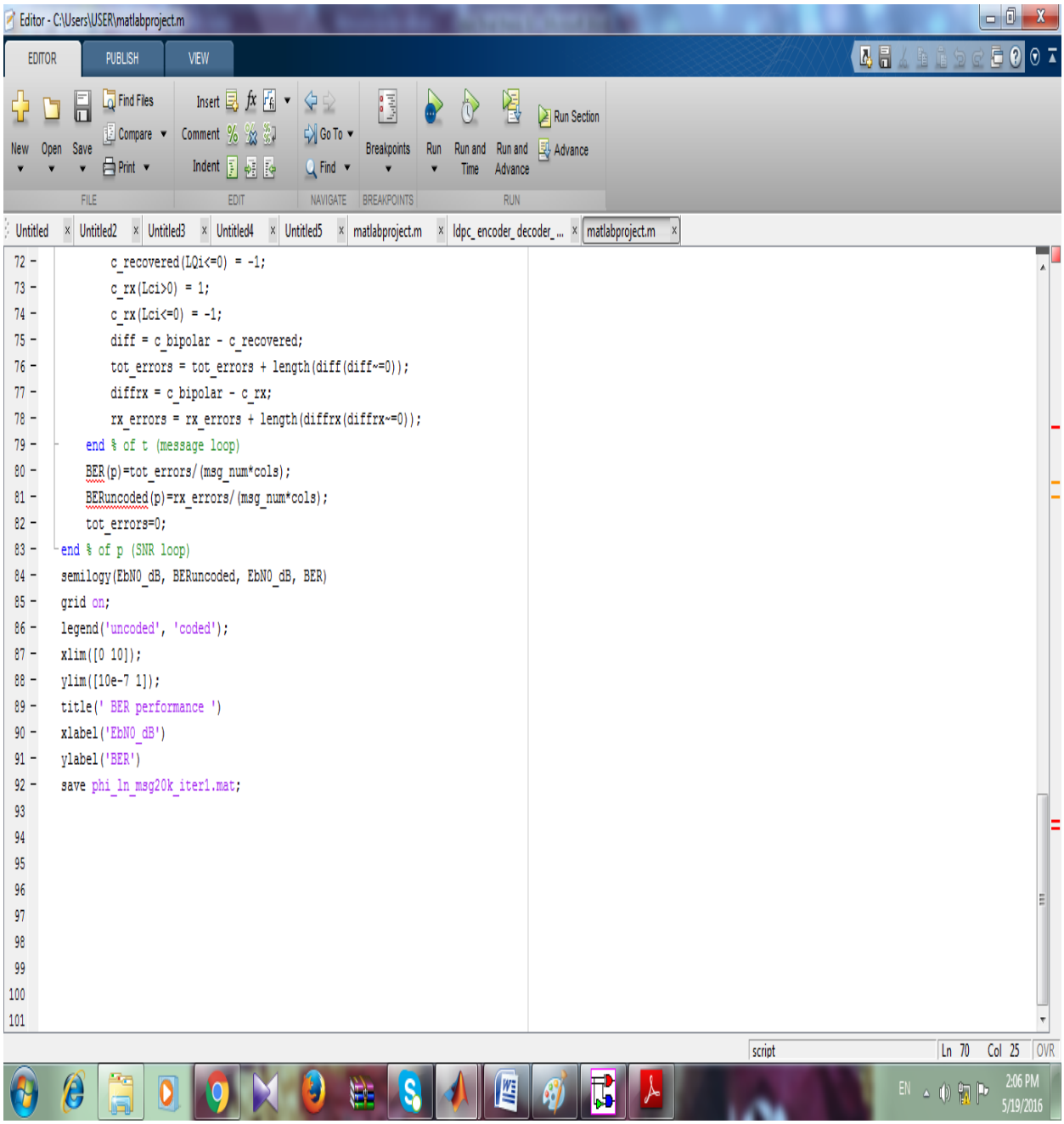


Fig.18 MATLAB implementation (4)

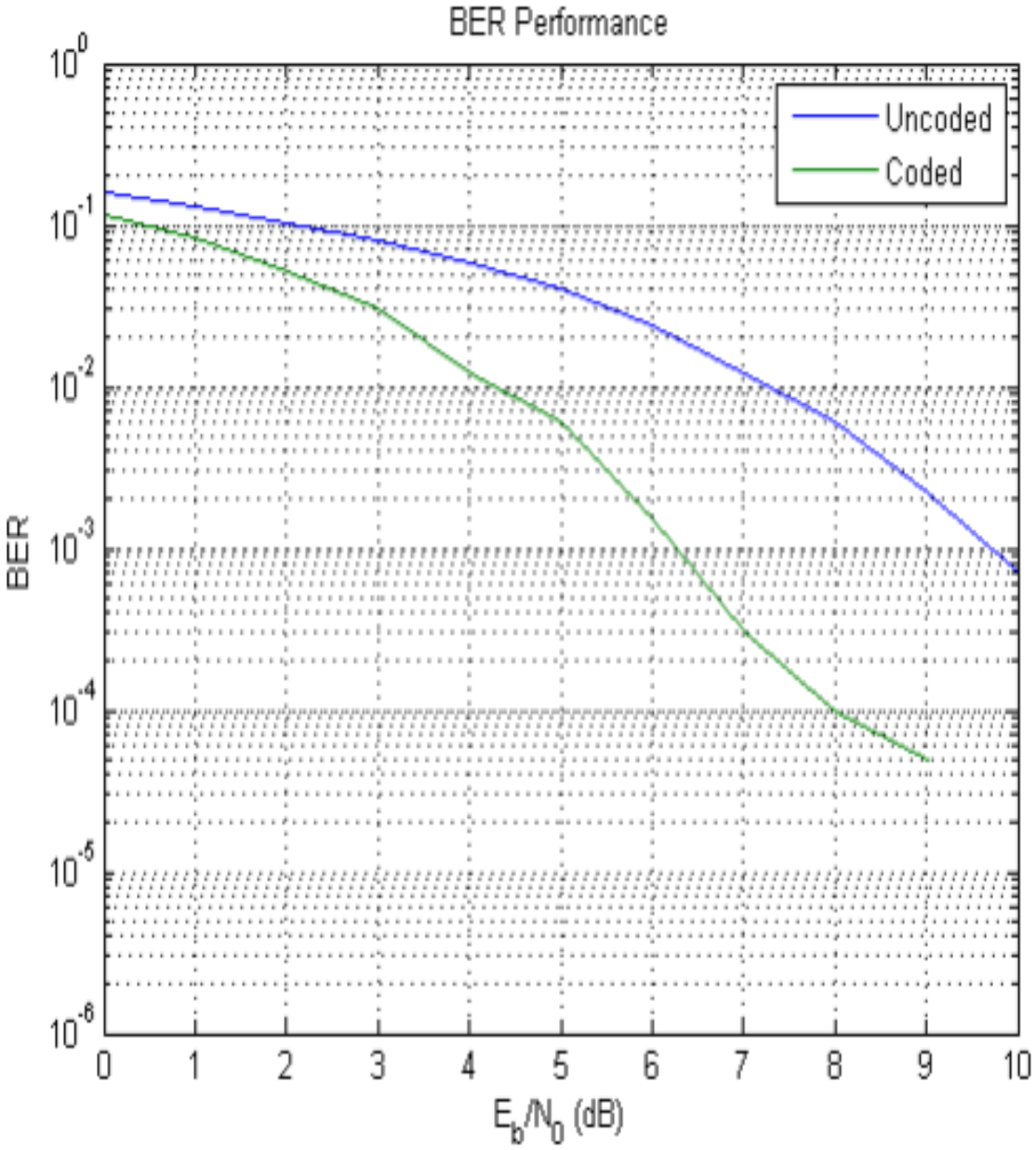


Fig.19 BER performance result

Chapter 7

Conclusion

&

Future scope

7.1 conclusion

In this project we have done analysis of LDPC codes and have studied different encoding and decoding algorithms.

The construction and hardware implementation of a LDPC system was the main objective of this thesis. The main objective was to construct good performing codes which are also easy to implement in hardware. Though we could achieve some of our objectives only. This work mainly focuses on digital implementation of encoder and MATLAB implementation of the decoding algorithm.

The conclusion from this project can be summarized as follows:

The error correcting performance of LDPC codes is near to Shannon's error correcting limit.

The error correcting capability of LDPC codes depends on various properties of parity check matrix such as matrix dimension, girth, regularity etc.

7.2 future scope

Some future work for a follow up to this project would include,

- reduction of hardware complexities.
- Besides hardware size and complexity, decoding latency is one of the critical factors for most applications. One major drawback of LDPC codes compared to Turbo codes is their low convergence rate. Turbo codes take on average 8-10 iterations to converge while LDPC codes typically need
- about 25-30 iterations to match the performance. A large number of iterations means longer decoding time. The overall decoding time could be

reduced by faster convergence of the decoding algorithm. That is another part were future work and improvement is possible.

- And few other changes could be in terms of speed , size, and power consumption.

APPENDIX

```
clc
Niter=2;
EsN0_dB=-3:1:7;
msg_num=2000;
H =[1111000000;
   1000111000;
   0100100110;
   0010010101;
   0001001011];
[rows, cols] = size(H);
G_sys=H';
%check=mod(G_sys*H_sys',2);
snr_range = length(EsN0_dB);
Lqij = zeros(rows, cols);
Lcix = zeros(rows, cols);
BER = zeros(1,snr_range);
BERuncoded = zeros(1,snr_range);
EbN0_dB = EsN0_dB+10*log10(2);
EsN0 = 10.^(0.1.*EsN0_dB);
sigma2 = 1./(2*EsN0);
amp=1;
m=randsrc(msg_num,rows,[0,1]);
parfor P = 1:snr_range
    Lrji = zeros(rows, cols);
    LQi = zeros(1, cols);
    c_recovered = zeros(1, cols);
    tot_errors = 0;
    c_rx = zeros(1, cols);
    rx_errors = 0;
    for t = 1:msg_num
```

```
        c = mod(m(t,:)*G_sys,2);
        c_bipolar = (2.*c-1);
        r = amp.*c_bipolar + sqrt(sigma2(p)).*randn(1, cols);
        Lci = 2.*r./sigma2(p);
         Lcix = repmat(Lci, [rows 1]);
        Lqij = H.*Lcix;
        for n = 1:Niter
            %update Lrji
            for j = 1:rows,
                for i = 1:cols,
                    if (H(j,i)==1)
                        prod=1;
                        sum_tmp=0;
                        for k = 1:cols,
                            if k ~=i && (H(j,k)==1)
                                prod=prod*sign(Lqij(j,k));
                                sum_tmp = sum_tmp + phi(abs(Lqij(j,k)));
                            end % of if
                        end % of k-loop
                        Lrji(j,i) = prod*phi(sum_tmp);
                    end % of outer if
                end % of i-loop
            end % of j-loop
            %update Lqij
            for i = 1:cols,
                sum_tmp = 0;
                for j = 1:rows,
                    if (H(j,i)==1),
                        for k = 1:rows,
                            if k~=j && (H(k,i)==1),
```

```
                                    sum_tmp = sum_tmp + Lrji(k,i);
                                end % of if
                            end % of k-loop
                            Lqij(j,i) = Lci(i) + sum_tmp;
                        end % of outer if
                    end % of j
                end % of i
                %update LQi
                LQi = Lci + sum(Lrji);
            end % of decoder
            c_recovered(LQi>0) = 1;
            c_recovered(LQi<=0) = -1;
            c_rx(Lci>0) = 1;
            c_rx(Lci<=0) = -1;
            diff = c_bipolar - c_recovered;
            tot_errors = tot_errors + length(diff(diff~=0));
            diffrx = c_bipolar - c_rx;
            rx_errors = rx_errors + length(diffrx(diffrx~=0));
        end % of t (message loop)
    end % of p (SNR loop)
```

REFERENCES

[1] R. G. Gallager, Low-Density Parity-Check Codes. Cambridge, MA: MIT Press, 1963.

[2] Ranjan Bose "*Information theory, coding and cryptography*", 3RD edition.

[3] Gabofetswe Alafang Malema "*LDPC construction and implementation*", The university of Adelaide, Australia, November 2007.

[4]Edward Liao1, Engling Yeo2, Borivoje Nikolić "*LDPC construction and hardware implementation"*, Department of Electrical Engineering & Computer Sciences, University of California, Berkeley, USA.

[5]LDPC research paper, Department of Electrical and Computer Engineering, Bradley University,05/2013 .

[6]T. Richardson and R. Urbanke, "The capacity of low-density parity check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[7]T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching low-density parity check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[8]M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.

[9]D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," in *Proc. 36th Allerton Conf. Communication, Control, and Computing*, Sept. 1998.

[10]R.M.Tanner "A recursive approach to low complexity codes" *IEEE Trans. Information Theory* ,pp.533-547 sept. 1981.

[11]D.MacKay and R.Neal " Good codes based on very sparce matrices" in *Cryptography and coding* 5th IMA conf. Springer 1995

[12]Error detection and correction, *http://en.wikipedia.org/wiki/Error_detection_and_correction* ,July 20, 2010

[13]J- Bhasker "Fundamentals of verilog" 3rd edition.

[14]LDPC codes using MATLAB: *http://sites.google.com/site/bsnugroho/ldpc* .

[15]William Y Ryan "An introduction to LDPC codes" , Department of electrical and computer engineering , University of Arizon, Aug.,2009.

[16]Behrouz A Farouzan – Data Communication, 4th edition.

[17]Das Subhashree "FPGA implementation of RS codes" , NIT Rourkela.

[18]N.Wiber "Codes and Decoding on general graphs", Ph.D dissertation, Linkoping University, Sweden, 1996.

[19] C. E. Shannon, “Certain results in coding theory for noisy channels,” Information and Control, vol. 1, pp. 6-25; September,1957.

[20] D. Slepian, “A class of binary signalling alphabets,” Bell Sys. Tech. J., vol. 35, pp. 203-234; January, 1956.

[21] P. Elias, “Coding for two noisy channels,” in “Information Theory,” C. Cherry, Ed., 3rd London Symp., September, 1955: Butterworths Scientific Publications, London, Eng., 1956.

[22] F. J. Bloom, et al., “Improvement of binary transmission by null-zone reception,” PROC. IRE, vol. 45, pp. 963-975; July,1957.

[23] R. M. Fano, “The Transmission of Information,” The Technology Press, Cambridge, Mass.; 1961.

[24] J. M. Wozencraft and B. Reiffen, “Sequential Decoding,” The Technology Press, Cambridge, Mass.; 1961.