FPGA IMPLEMENTATION OF LOW DENSITY PARITY CHECK CODES DECODER

Suresh Vijayakumar

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2009

APPROVED:

Armin R. Mikler, Major Professor and Graduate Coordinator
Shengli Fu, Major Professor
Philip Sweany, Committee Member
Krishna M. Kavi, Chair, Dept of Computer Science and Engineering
Bill Buckles, Associate Dean of the College of Engineering
Costas Tsatsoulis, Dean of the College of Engineering
Michael Monticino, Dean of the Robert B. Toulouse School of Graduate Studies

Vijayakumar, Suresh. FPGA implementation of low density parity check codes decoder. Master of Science (Computer Engineering), August 2009, 55 pp., 1 table, 13 illustrations, references, 50 titles.

Reliable communication over the noisy channel has become one of the major concerns in the field of digital wireless communications. The low density parity check codes (LDPC) has gained lot of attention recently because of their excellent error-correcting capacity. It was first proposed by Robert G. Gallager in 1960. LDPC codes belong to the class of linear block codes. Near capacity performance is achievable on a large collection of data transmission and storage. In my thesis I have focused on hardware implementation of (3, 6) – regular LDPC codes. A fully parallel decoder will require too high complexity of hardware realization. Partly parallel decoder has the advantage of effective compromise between decoding throughput and high hardware complexity. The decoding of the codeword follows the belief propagation alias probability propagation algorithm in log domain. A 9216 bit, (3, 6) regular LDPC code with code rate ½ was implemented on FPGA targeting Xilinx Virtex 4 XC4VLX80 device with package FF1148. This decoder achieves a maximum throughput of 82 Mbps. The entire model was designed in VHDL in the Xilinx ISE 9.2 environment.

ACKNOWLEDMENTS

I would like to express sincere gratitude to my major professors, Dr. Hao Li and Dr. Armin Mikler, my co-advisor Dr. Shengli Fu for their guidance on my research and my course works. They provided me with a lot of valuable advice and constructive criticism. I would also like to thank Dr. Sweany for being in the committee and helping me throughout my masters program. I also express my sincere gratitude Dr. Krishna Kavi, Chair of CSE department, for his constant support and motivation. I am thankful to my CSRL colleagues Cameron, Tommy, Tim, Pete and Paul for their generous support. I am also grateful to Anand Loganathan and Arvind Chandrasekaran for helping me during my graduate studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Motivation and Objective

The field of wireless communication has undergone a phenomenal growth in both scope and application. The need to transmit and receive information in more reliable way over a noisy channel has become an essential factor which determines the performance. There have been several major developments in the field of error correcting codes in the past and various coding techniques have been introduced, all of these techniques aim achieving reliable communication. Error correction is the ability to re-construct the original information which was transmitted. An error-correcting code is an algorithm for expressing a sequence of bits such that any errors which are introduced can be detected and corrected (within certain limitations) based on the remaining bits [49]. In recent past the low density parity check codes (LDPC) gained more attention and is considered as the important error-correcting codes for the coming years in the field of telecommunication and magnetic storage. There are many industrial standards which proposed the incorporation of LDPC codes. Some of them include digital video broadcast (DVB) satellite communication, wireless broadband, wireless local area networks and gigabit ethernet. Hardware realization of LDPC codes is considered as one of the most challenging tasks in the research community. This thesis is based on the hardware implementation of LDPC decoder. The hardware implementation was targeted on FPGA, because it has the advantage of flexibility over traditional ASIC implementation.

1.2. Channel Coding

1.2.1 Shannon Theorem for Channel Coding

Communication over noisy channels can be more effectively improved by the use of a channel code C, as introduced by C. E. Shannon in 1948. The theorem is stated as below [23]

"Let a discrete channel have the capacity C and a discrete source the entropy per second H. If H ≤ C there exists a coding system such that the output of the source can be transmitted over the channel with an arbitrarily small frequency of errors (or an arbitrarily small equivocation). If H > C it is possible to encode the source so that the equivocation is less than H."

According to the theorem, a noisy channel with channel capacity *C* and the rate *R* at which information is transmitted, then there are codes existing, which allow the probability of error at the receiver end to be made very small. This signifies that theoretically, it always possible to transmit message nearly without noise and error at any rate which is less than the limiting rate, *C*. The converse of this theorem is also equally important. The converse states that if $R > C$, an arbitrarily small probability of error is not achievable [23]. This means that all the codes will have an error probability greater than a certain specified level and it increases with the increase in the rate, *R*. Therefore, there is no guarantee that the data will always be transmit reliably across a channel at any rate which is beyond the channel capacity. The channel coding theorem is the single most important result of information theory. The theorem specifies the channel capacity *C* as a fundamental limit on the rate at which the transmission of reliable error free messages can take place over a discrete memory less channel [23]. However this theorem doesn't say how to reconstruct the error correcting method.

1.2.2 Communication Model

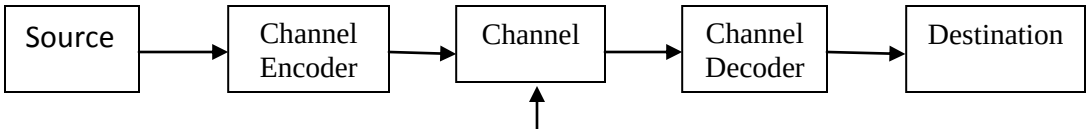

Figure 1.1 Block Diagram of Communication System

The above block diagram shows the typical communication model. The source block delivers information by the mean of sequences which are row vectors *x* of length *K*. The output of the encoder block is the codeword *c* of length *N*. The code rate of any LDPC codes is defined by the ratio $R = K/N$. The codeword *c* is sent over the channel and the received vector *y* is the received codeword, a distorted version of *c*. The output *y* from the channel depends upon the conditional probability density function *(pdf)*. We assume that the channel is memory less, then

(1) $$p(y \mid c) = \prod_{n=1}^{N} p(y_n \mid c_n)$$

If the channel is the additive white Gaussian noise (AWGN), and if the modulation is a binary phased shift keying (BPSK) modulation with the 0 → +A, 1 →−A mapping, then

we have:

(2) $$p(y \mid c) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y_n - (-1)^{c_n} \sqrt{E_s})^2}{2\sigma} \right)$$

Where $E_s = A^2$ is the energy of the symbol sent over the channel.

1.2.3 Optimal Decoding

The most important significance of the decoder is to find the codeword $\hat{c}$ which is the most probable to have been sent over the channel.

(3) $$\hat{c} = \underset{c' \in C}{\arg\max} \Pr(c = c' \mid y)$$

This is the word maximum a posteriori W-MAP decoder. Using Baye's rule the posterior probabilities $\Pr(c' \mid y)$ are expressed as

(4) $$\Pr(c \mid y) = \frac{p(y \mid c)\Pr(c)}{p(y)} = \frac{p(y \mid c)\Pr(c)}{\sum_{c \in C} p(y \mid c)\Pr(c)}$$

If the priori probabilities are Pr (c) are identical, then (3) could be represented as

(5) $$\hat{c} = \underset{c' \in C}{\arg\max}\ p(y \mid c = c')$$

This is called word maximum likelihood decoding. The function $p(y \mid c)$ is the likelihood function when *y* is fixed and it is a conditional pdf when *c* is fixed. The only way to achieve an optimal W-MAP decoder is to test each codeword. The W-MAP and W-ML decoders are two equivalent and optimal decoders if the source is equally probable. The Viterbi algorithm [24, 25] is an efficient W-ML decoder which eliminates many operations and leads to the best word (frame) error rate (FER), providing that the code has a trellis representation and that the source words are equally likely to happen. There are basically two types of decoding, they are

(i) *Soft decoding***:** The output samples y_n of the channel are not decided: they are the inputs of the decoder. Using the channel specifications, the decoder computes the probability for each y_n to be each one of the code-alphabet element. Then, the decoding process will base its decisions on the value of these probabilities.

(ii) *Hard decoding*: The output samples y_n of the channel are decided: each of them is associated with the most probable code-alphabet element. Then a processing is performed to try to detect and correct the transmission errors. This processing is made without using the knowledge of the probability set.

1.3 Error Correcting Codes

Error correcting codes are the techniques used for providing robust data transmission through imperfect channels by adding redundancy to the data [48]. There are two important classes of such coding methods: block and convolution codes. When the device at the receiving end does most of the detection and correction, then it is known as forward error correction. On considering block codes, the decoder detects errors and once detected, the decoder also performs the correction. This is one of the most important techniques used widely in today's communication systems and other wide range of digital applications. This kind of coding technique provides high performance at low cost.

1.3.1. The Shannon Capacity

The channel capacity of an AWGN channel is given by

(6) $$C = B\log_2(1+SNR) \quad [bits/\sec]$$

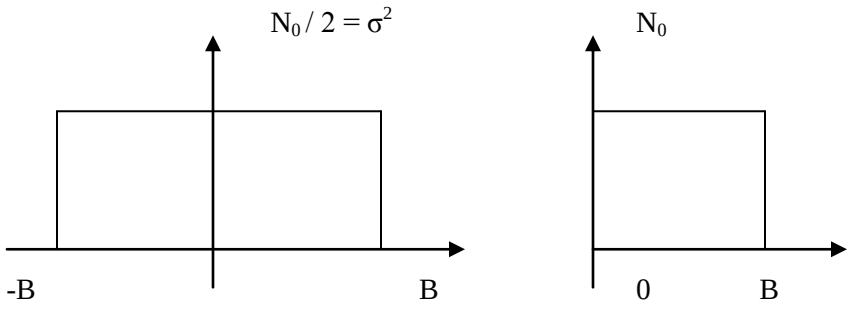

Figure 1.2 AWGN Noise Power Spectral Density

Here B is the channel bandwidth and SNR is the signal to noise ratio of the transmitted signal power P_S and the channel noise power P_N

Here

(7) $$P_N = N_0 B$$

(8) $$P_S = R_I E_B$$

Here N_0 denotes the one sided noise power spectrum density and E_B denotes the energy per information bit. The information rate R_I is expressed as

(9) $$R_I = \frac{R \log_2(M)}{T_S} \quad [bit/\sec]$$

Where R is the code rate, M is the size of the constellation of the modulation and T_S is the symbol time duration.

Substituting equations (8) and (7) in (6) we get

(10) $$C = B \log_2(1 + \frac{P_S}{P_N}) \quad [bits/\sec]$$

(11) $$C = B \log_2(1 + \frac{R_I E_b}{N_0 B}) \quad [bits/\sec]$$

Hence the Shannon theorem can be summarized as

If,

$R_I < C$ then reliable communication

$R_I > C$ then unreliable communication

Using spectral efficiency $\boldsymbol{\eta}$ the Shannon theorem is stated as

(12) $$\eta = \frac{R_I}{B}$$

Thus, the maximal information rate defines a maximum spectral efficiency $\boldsymbol{\eta}_{\max}$:

(13) $$\eta_{\max} = \frac{R_{I\max}}{B}$$

According to the definition of Shannon theorem, the maximal information rate is equal to capacity:

(14) $$R_{I\max} = B\log_2(1+\frac{R_{I\max}E_b}{N_0B})$$

(15) $$\frac{R_{I\max}}{B} = \log_2(1+\frac{R_{I\max}E_b}{N_0B})$$

(16) $$\eta_{\max} = \log_2(1+\eta_{\max}\frac{E_b}{N_0})$$

(17) $$\left(\frac{E_b}{N_0}\right)_{\min} = \frac{2^{\eta_{\max}}-1}{\eta_{\max}}$$

The Equation (17) is called Shannon bound.

1.3.2 The AWGN Channel

The bandwidth efficiency is not very useful to study only the properties of error correcting codes because it takes into account all the signals used in the transmission. According to the Nyquist theorem, a real signal which has a bandwidth B can be sampled at a rate of 2B samples per second without any inter-symbol interference. So the 2B samples are "independent"

and they are carried on 2B signal dimensions [dim] per second. The rate per signal dimension is defined by $R_{Id} = R_I/(2B)$ and its maximum by the capacity $C_d = C/(2B)$.

Thus,

(18) $$C_d = \frac{B}{2B}\log_2(1+\frac{2R_I E_b}{2N_0 B})$$

(19) $$C_d = \frac{1}{2}\log_2(1+\frac{2R_{Id} E_b}{N_0})$$

(20) $$C_d = \frac{1}{2}\log_2(1+\frac{2C_d E_b}{N_0})$$

Therefore the above equation is simplified as

(21) $$\left(\frac{E_b}{N_0}\right) = \frac{2^{2C_d}-1}{2C_d}$$

The capacity in expression (20) is by definition the maximization of the mutual information between the input and the output of the AWGN channel over the input channel probabilities [50]. The maximum occurs for a Gaussian distribution of the channel input

1.4 Linear Block Codes

1.4.1 Terminology

A linear block code C defined over the Galois field $GF(q)$ is a k-dimension vector subspace of $GF(q)^N$. The code C can be defined by the list of all the code words:

(22) $$C = \{c^i, i \in (0,.....2^{k-1})\}$$

It can be alternatively defined by a vector base B_C of K independent codeword, $c^i, i \in (0,.....k-1)$, which is not unique. The vector base B_C has itself many useful and equivalent representations:

Generator matrix G:

Whose rows are the vectors of the base B_C (so G is a $K \times N$ matrix):

Parity-check matrix H:

Which is a $(N-K) \times N$ matrix with elements in $GF(q)$: $C = \{c^{(i)}/c^{(i)}.H^t = 0\}$. The parity check matrix is a concatenation of $M = (N-K)$ rows denoted by pc_m. Each row pc_m of H is a parity check equation on some bits of the codeword. The bits implied in the parity-check pc_m are the non-zero entries of the $m-$th row of H. H is the orthogonal complement of C: $GH^t = 0$.

Tanner graph

A Tanner graph [26] is a bipartite graph, where the elements of a one type can be connected to the elements of a second type, but not to the same type. In a Tanner graph for binary block codes, the elements of the first type are the variable nodes denoted by vn_n and the elements of the second type are the check nodes denoted by cn_m. Each variable node vn_n is associated with one code symbol c_n and each check node cn_m is associated with the m-th parity check constraint pc_{m} of H. A variable node vn_n is connected to a check node cn_m if and only if $H(m, n)$ has a non-zero entry.

1.4.2. Decoding Based on Iterative Algorithm

In coding theory the iterative algorithm is one where the algorithm calculates the distribution of variables in a graph model. It is called with different names based in the context.

They are:

a) Sum-product algorithm

b) Belief propagation algorithm

c) Message passing algorithm

The term “message passing” refers usually refers to all such iterative algorithms. The works on the principle of passing the values of real vector messages through the edges of the connected nodes in a graphical mode. Specifically, in trees: a node sends a message to an adjacent node if it has received messages from all its adjacent nodes and if it hasn’t sent one already. Hence during the first iteration, the algorithm passes messages from all leaf nodes to every lone node that is adjacent to those corresponding leaves; this continues until all the messages have been passed exactly one, this describes the term propagation. On the termination of the algorithm, the variable margin is simple the product of all the incoming messages from all its adjacent nodes.

CHAPTER 2

LOW DENSITY PARITY CHECK CODES

2.1 History

Low-density parity-check (LDPC) codes are class of linear block codes which provide near Shannon capacity performance. LDPC codes were invented by R. G. Gallager [1] in 1962. He also proposed an iterative decoding algorithm and this was applied in decoding of this code. He named these codes low-density parity-check (LDPC) codes since the parity-check matrices had to be sparse to perform well. But still LDPC codes have been ignored for a long time due mainly to the requirement of high complexity computation, if very long codes are considered. In 1981 Tanner generalized the LDPC codes and introduced a graphical representation of LDPC codes. This representation is now called Tanner graphs [26]. In 1993, C. Berrou et al. invented the turbo codes [27] and they used this iterative decoding algorithm for the decoding purpose. They observed a great performance of iterative decoding with the turbo codes. In 1995, D. J. C. MacKay and R. M. Neal [28] rediscovered the LDPC codes, and established a bridge between their iterative algorithms to the algorithm proposed by Pearl [29], from the artificial intelligence community (Bayesian networks). M. Sipser and D. A. Spielman [30] used the algorithm proposed R. G. Gallager in decoding the expander codes. Hence, LDPC codes are the convergence of two major techniques in the channel coding. They are graph model based code and iterative decoding

2.2 Representation of LDPC Codes

LDPC codes are mainly represented in two ways

(i) Matrix representation

(ii) Graphical representation

2.2.1 Matrix Representation

LDPC codes can also be applied to non-binary alphabets. Here we consider only binary LDPC codes to make things simpler. Since LDPC codes are class of linear block codes they can be described as certain *K* dimensional subspace of *C* of the vector space F^n_2 of binary *n*- tuples over the binary field F_2. Hence the basis is expressed as $B = \{g_0, g_1, \ldots g_{k-1}\}$, and this spans over *C*.

Therefore each $c \in C$ can be expressed as $c = u_0 g_0 + u_1 g_1 + \ldots + u_{k-1} g_{k-1}$ for some $\{u_i\}$; thus $c = uG$, where $u = [u_0\ u_1\ \ldots\ U_{k-1}]$. Here *G* is the called the *k x n* generator matrix. The *(n-k)* dimensional null space $C^{\perp}$ of *G* comprises all vectors $x \in F^n_2$ for which $xG^T = 0$ and is spanned by basis $B^{\perp} = \{h_0, h_1, \ldots h_{n-k-1}\}$. Therefore $cH^T = 0$ where *H* is known as the parity check matrix of dimension *(n-k) x n*. Since it performs m= n-k parity checks on the received codeword is called parity check matrix.

LDPC codes are represented by parity check matrix, the *H* matrix which contains low density of 1's. If the parity check matrix *H* contains exactly *j* 1's in each column and exactly *k* 1's in each row, then it is called a regular LDPC code.

For example let's consider an example of (10, 5) linear block code with *j* =3 and *k*= 6. The *H* matrix for this code is represented as shown below

(22)
$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Figure 2.1: Parity Check Matrix of a (10, 5) Linear Block Code.

2.2.2 Graphical Representation

LDPC codes are graphically represented using Tanner graphs. Tanner graphs also called as bipartite graphs of LDPC code is represented using nodes and connected by edges. There are two types of nodes

(i) Check nodes (*cn*)

(ii) Variable nodes (*vn*)

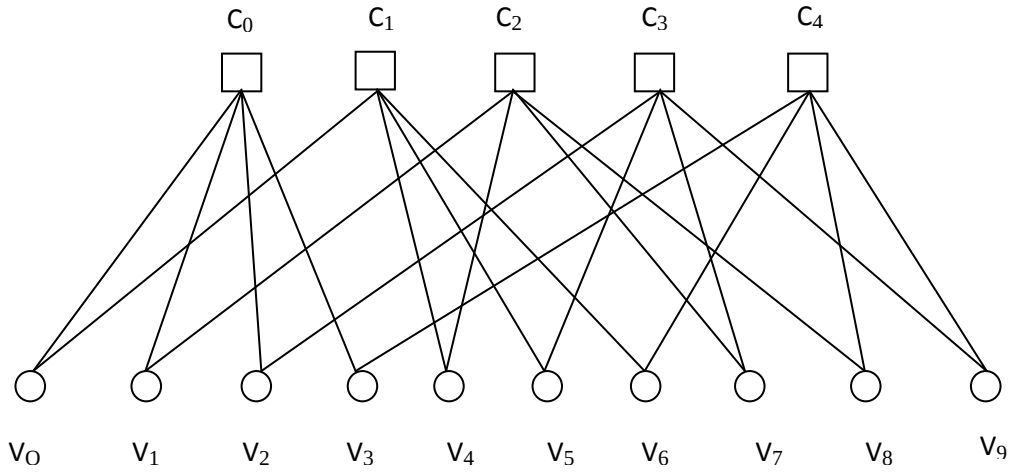

Figure 2.2: Tanner Graph for the (10, 5) Linear Block Code.

Check nodes (*cn*) represent the rows of the parity check matrix and the variable nodes (*vn*) represent the columns. The connection between the check nodes and variable nodes are determined by the elements of the *H* matrix. Whenever the element h_{ij} in the *H* matrix is a 1, then there exits an edge between the check node *j* and variable node *i*. The Tanner graph corresponding to the parity check matrix in (22) is shown in the above figure. In the above figure the variable nodes v_0, v_1, v_2, v_3 are connected to check node c_0, since in the *H* matrix the elements $h_{00} = h_{01} = h_{01} = h_{01} = 1$. And similarly this hold good for all other check nodes.

2.3 Classes of LDPC Codes

R. Gallager defined an (N, j, k) LDPC codes as a block code of length *N* having a small fixed number (*j*) of ones in each column of the parity check *H*, and a small fixed number (*k*) of ones in each rows of *H*. This class of codes is then to be decoded by the iterative algorithm. This algorithm computes exact a posteriori probabilities, provided that the Tanner graph of the code is cycle free. Generally, LDPC codes do have cycles [31]. The sparseness of the parity check matrix aims at reducing the number of cycles and at increasing the size of the cycles. Moreover, as the length *N* of the code increases, the cycle free hypothesis becomes more and more realistic. The iterative algorithm is processed on these graphs. Although it is not optimal, it performs quite well. Since then, LDPC codes class have been enlarged to all sparse parity check matrices, thus creating a very wide class of codes. The LDPC codes are classified into two types

(i) Regular codes

(ii) Irregular codes

2.3.1 Regular Codes

In the Gallager's original LDPC code design, there are a fixed number of ones in both the rows (*k*) and the columns (*j*) of the parity check matrix: it means that each bit is implied in (*j)* parity check constraints and that each parity check constraint is the exclusive-OR (XOR) of (*k*) bits. This class of codes is referred to as regular LDPC codes. The Tanner graph shown in figure 2.1 is an example of regular LDPC code. In that example each variable node has two edges connected to the check nodes. This exactly corresponds to the fact that each column in the *H* matrix has two 1s. Similarly each check node is connected to 4 variable nodes, corresponding to the *H* matrix where each row has four 1s.

And in this thesis I have implemented a (3, 6) regular LDPC code. Here (3, 6) corresponds to the number of 1s in column (*j*) and number of 1s in a row (*k*).

2.3.2 Irregular LDPC Codes

Irregular LDPC codes do not have a constant number of non-zero entries in the rows or in the columns of H. They are specified by the distribution degree of the bit $\lambda(x)$ and of the parity check constraints $\rho(x)$, using the notations of Luby et al. [5], we have

(23) $$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1}$$

(24) $$\rho(x) = \sum_{i=2}^{d_c} \rho_i x^{i-1}$$

Here λ_i denotes the proportion of non-zero entries in the *H* matrix which belongs to the columns of weight *i*. Similarly ρ_i denotes the proportion of non-zero entries in the *H* matrix which belong to the rows of weight *i*.

If Γ is the number of non-zero entries in *H*, then $\lambda_i \Gamma$ is the total numbers of ones in columns of weight *i*. So the proportion of columns of weight *i* is given by

(25) $$\tilde{\lambda}_i = \frac{\Gamma\lambda_i / i}{\sum_j \Gamma\lambda_j / j} = \frac{\lambda_i / i}{\sum_j \lambda_j / j}$$

Similarly the proportion of rows of weight *i* is given by

(26) $$\tilde{\rho}_i = \frac{\Gamma\rho_i / i}{\sum_j \Gamma\rho_j / j} = \frac{\rho_i / i}{\sum_j \rho_j / j}$$

2.4 Code Rate

The LDPC code rate R is defined as

(27) $$R \geq R_d = \frac{1-M}{N}$$

Here R_d is the design code rate. $R_d = R$ if the parity check matrix is considered to be of full rank. Miller and Cohen [43] have shown that as N increases, the parity-check matrix is almost sure to be full rank. Hereafter, we will assume that $R = R_d$ unless the contrary is mentioned. The rate R is then linked to the other parameters as shown below,

(28) $$R = \frac{\sum_i \rho_i / i}{\sum_i \lambda_i / i} = 1 - \frac{j}{k}$$

Hence in general for random constructions, when j is odd, we have

(29) $$R = 1 - \frac{M}{N}$$

When j is even we have

(30) $$R = 1 - \frac{M-1}{N}$$

2.5 LDPC Codes Construction Methods

Constructions of LDPC codes is the design, of a particular LDPC parity check matrix H. The design of H is the moment when the asymptotical constraints (the parameters of the class you designed, like the degree distribution, the rate) have to meet the practical constraints (finite dimension, girths). Two techniques exist in the literature:

(i) Random and

(ii) Deterministic

The design compromise is that for increasing the girth, the sparseness has to be decreased yielding poor code performance due to a low minimum distance. On the contrary, for high minimum distance, the sparseness has to be increased yielding the creation of low-length girth, due to the fact that *H* dimensions are finite, and thus, yielding a poor convergence of the belief propagation algorithm.

2.5.1 Random Construction

The first constructions of LDPC codes were random ones. The parity check matrix is the concatenation and/or superposition of sub-matrices; these sub-matrices are created by processing some permutations on a particular (random or not) sub-matrix which usually has a column weight of 1. R. Gallager's construction for example is based on a short matrix H_0. Then j matrices $\pi_i(H_0)$ are vertically stacked on H_0, where $\pi_i(H_0)$ denotes a column permutation of H_0. Regular and irregular codes can be also constructed like in (Luby et al. 2001) where the 2 sets of nodes are created, each node appearing as many times as its degree's value. Then a one to one association is randomly mapped between the nodes of the 2 sets. In [32] D. MacKay compares random constructions of regular and irregular LDPC codes: small girth has to be avoided, especially between low weight variables. All the constructions described above should be constrained by the girth's value. Still, increasing the girth from 4 to 6 and above is not trivial; some random constructions specifically address this issue. In [33], the authors Campello and Modha generate a parity check matrix optimizing the length of the girth or the rate of the code when M is fixed N increases.

2.5.2 Deterministic Construction

Random constructions of LDPC codes are pretty simple and don't have many constraints. The problem is that they do not guarantee that the girth will be small enough. So either post-

processing or more constraints are added for the random design, yielding sometimes much complexity. To circumvent the girth problem, deterministic constructions have been developed. Moreover, explicit constructions can lead to easier encoding, and can be also easier to handle in hardware. Two branches in combinatorial mathematics are involved in such designs: finite geometry and balanced incomplete block design (BIBSs). They seem to be more efficient than previous algebraic constructions which were based on expander graphs [34, 35, 36]. In [32] the authors Mackay and Davey designed high rate LDPC codes based on Steiner systems. Their conclusion was that the minimum distance was not high enough and that difference set cyclic (DSC) codes should outperform them, as in Lucas et al. 2000 [37], where they are combined with the one step majority logic decoding. In [38] the authors Kou, Lin, and Fossorier present LDPC code constructions based on finite geometry, like in [39] (Johnson and Weller) for constructing very high rate LDPC codes. Balanced incomplete block designs (BIBDs) have also been studied in Ammar et al. [40]; Vasic et al. [42]; Johnson and Weller [41]. The major drawback for deterministic constructions of LDPC codes is that they exist with a few combinations of parameters. So it may be difficult to find one that fits the specifications of a given system.

CHAPTER 3

DECODING OF LDPC CODES

3.1 LDPC Software

Radford M. Neal, Dept. of Statistics and Dept. of Computer Science, University of Toronto has designed a software package [44] that simulates the low density parity check codes (LDPC). This software contains various modules like parity check module, encoding, transmission and decoding. This software package was very useful for my thesis since it gave me a very good understanding over the working of LDPC codes.

3.1.1 Creation of Parity Check Matrix

This module of the LDPC codes software can be used to create parity check matrix or *H* matrix of different sizes. It deals with only linear block codes of binary vectors. The set of valid codewords for a linear code can be specified by giving a parity check matrix, *H*, with *M* rows and *N* columns [45]. The codewords are represented by vectors, *x*, which is of length *N*. To be a valid codeword this vector, *x* should satisfy the condition $Hx=0$, where all arithmetic is performed using modulo-2 arithmetic. Every row of the parity check matrix, *H* represents a parity check on a subset of the bits in the codeword. This matrix is then stored in a file by the software in sparse format which is specified by the software itself. The there two different methods that we can choose from the software to create the parity check matrix. They are

1) Evencol method
2) Evenboth method

In *evencol* method all the columns in the parity check matrix have equal number of ones. Every column of the parity check matrix, the algorithm places a specified number of 1s in positions selected at random. The only constraint that is to be considered during this process is

that these 1s are placed in distinct rows. In *evenboth* method the numbers of 1s in all rows are equal and similarly numbers of 1s in all columns are equal. The position of theses ones in the *H* matrix are randomly decided by pseudo randomness algorithm.

3.1.2 Encoding Technique

In order to transmit the information bits, a mapping from bit vector *s*, the source vector of length *K* bits, to a codeword represented as *x*, of length $N > K$. This software considers only linear mapping, which is expressed as,

(31) $$x = G^T s$$

Here *G* is known as the generator matrix. For a code with parity check matrix *H*, whose codewords satisfy $Hx=0$, the generator matrix must satisfy $HG^T=0$ [46]. According to this software, the number of rows in the parity check matrix, *M*, is always equal to *N-K*. The encoding schemes followed by this software are always systematic, where *K* bits of the source vector, *s* are copied without making any changes to it, to a subset of the *N* bits of the codeword, *x*. Finally the remaining *M=N-K check bits* of the codeword, *x* are then set so that it results in a codeword. On considering any linear code, there exists a systematic encoding technique, which decides on the choice of which bits of the codeword to be the message bits. The very straightforward technique is to rearrange the codeword bits in such a way that the message or information bits always come at the beginning. The first *K* columns of the *K* x *N* generator matrix will always be the identity matrix. The codeword is divided into parts. The first part comprises of the *M* check bits, *c*. The second part of the codeword is formed by the *K* information bits, *s*. Thus dividing the parity check matrix into an *M* x *M* matrix *A* occupying the first *M* columns of *H* and an *M* x *K* matrix *B* occupying the remaining columns of *H* [46]. The criteria of a codeword, *x*, to satisfy all parity checks is represented as

(32) $$Ac + Bs = 0$$

On simplifying the above equation provided *A* is a non singular matrix we have,

(33) $$c = A^{-1}Bs$$

The above equation defines how the check bits can be computed. This software follows three different methods in creating a generator matrix for the given vector of information bits. They are

i) Dense representation

ii) Mixed representation

iii) Sparse representation

Once the generator matrix is created by using one of the three methods mentioned above, then the encoding of the information bits is done.

3.1.3 Transmission through Channel

Once the codeword is formed by encoding the information bits using the generator matrix, it can be transmitted through a simulated channel by adding random noise to the channel. This software supports three different types of channel simulation.

i) Binary symmetric channel

ii) Additive white Gaussian noise channel - AWGN

iii) Additive white logistic noise channel - AWLN

In binary symmetric channel the bit sent results in the bit received which differs with some error probability, *p*. It is same for both 0 bits and 1 bit. In the AWGN channel, the received data each time is always equal to the data which was originally sent plus some Gaussian noise with mean zero and standard deviation, *s*. For this software, the data sent is -1 for a 0 bit and +1 for a 1 bit

[44]. The AWLN channel is very similar to the AWGN channel, but the noise is generated from logistic distribution.

3.1.4 Decoding Technique

The codewords that was transmitted over the simulated channel are decoded from the received data on the basis of likelihood of the possible codeword [47]. Likelihood is defined as the probability of receiving the data that was actually received if the codeword is the one which was actually sent. In simple words likelihood is the ratio of probability of the bit being 1 by probability of the bit being 0. This software follows two different methods in decoding the codeword. They are

- Exhaustive enumerated decoding
- Probability propagation or belief propagation

The exhaustive enumerated decoding finds the all possible codeword from the source message or information bits using the generator matrix. Then it checks which one satisfies the parity check, and the one which best does is finally the decoded codeword.

With the parity check matrix, H and the likelihood ratios for each bit, the probability propagation method finds the most probable values for the bits of the codeword. The algorithm uses only the parity check matrix for the code, whose columns correspond to codeword bits, and whose rows correspond to parity checks, and the likelihood ratios for the bits derived from the data [47]. The goal of the algorithm is to find the probability of each bit of the transmitted codeword being 1.

3.2 Log Domain and Probability Domain

The Decoding of the codeword could be done in two domains, probability domain and log domain. This software follows decoding in probability domain. Log domain decoding is

same as the probability domain, but the likelihood ratios of each bit are expressed in log forms. This ratio is known as log-likelihood ratio. The likelihood ratio is expressed as shown below in the equation (34). It is the ratio of the probability of the bit being 0 to the probability of the bit being 1.

$$l(c_i) = \frac{\Pr(c_i = 0 \mid y)}{\Pr(c_i = 1 \mid y)} \tag{34}$$

The log-likelihood ratio is natural log of the above expression.

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid y)}{\Pr(c_i = 1 \mid y)} \right) \tag{35}$$

The log domain decoding approach is followed in the hardware implementation of the LDPC codes. Direct implementation of the probability propagation algorithm consumes high complexity of hardware and it will be very difficult to implement it since it involves complex multiplications of the probabilities of each bit of the codeword. Hence to reduce this complexity, the hardware implementation follows the log domain. The usage of log-likelihood ratio reduces complex multiplication in to sums, thus reducing the hardware complexity.

3.3 Implementation of Log Domain Algorithm

In my thesis, before I implement the log domain based decoding in hardware, I verified the accuracy and working of the log domain based algorithm in software. I used these software modules and designed a log domain based decoding in C. I used most of the functions from this software and implemented this algorithm. This helped me greatly in the hardware implementation of this algorithm. For accuracy purposes I compared the results obtained from both the log domain based and probability domain based algorithm and found to be same. The exact flow and working of these algorithms are explained in detail in following topics.

3.4 Decoding Algorithm

3.4.1 Belief Propagation Algorithm

Apart from introducing the LDPC codes, Gallager also proposed a decoding algorithm that is typically near optimal. Since then, many people have discovered many related algorithms. The algorithm is commonly called with different names such as message passing or sum-product or belief propagation algorithm. This algorithm iteratively performs the computation of the distribution of the variables in graph based models and come under different names as mentioned in 1.4.2. In this thesis the algorithm that was adopted for the decoder design of LDPC codes is belief propagation algorithm which was proposed by Gallager. Log domain implementation of belief propagation was followed due to its advantage of less hardware complexity. One best example of sum-product algorithm is the iterative decoding algorithm for turbo codes.

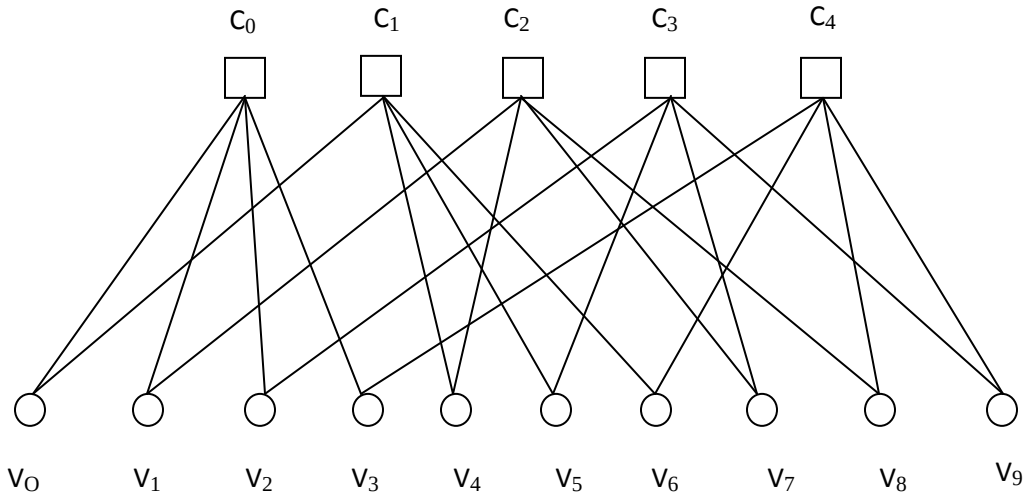

Figure 3.1 Tanner Graph Representation of (5, 10) Parity Check Matrix

Consider the Tanner graph shown in the Figure 3.1 for an LDPC code. This is also called as a bipartite graph defined by a very sparse *M x N* parity check matrix, *H*. As explained in section 2.2.2, there are two types of nodes in this graph, check nodes (C_n) and variable nodes (V_n). The *N* variable nodes correspond to the section of codeword bits and the *M* check nodes

correspond to the section of parity check bits. Similarly check nodes represent the rows and the variable nodes represent the columns of the parity check matrix. The connections between these two nodes represent the non-zero entry in the *H* matrix. During the decoding process, information is sent along the edges of the graph. It is very important that the graph is sparse, because this helps in better iterative processing. All though this algorithm only leads to a suboptimal solution, it generally attains near optimal solution and best performance. Throughout this thesis, an additive white Gaussian channel is considered as the channel medium through which messages are transmitted.

3.5 Decoding Procedure

The decoding domain followed here is log domain and hence the algorithm is called log belief propagation algorithm. The notations and definitions used in this algorithm are adopted from the work of Zhang and Parhi [6]. Here, let *H* denote the *M* x *N* sparse parity check matrix of the LDPC code and $H_{i,j}$ denote the non-zero entry of the matrix, *H* at the position (*i, j).* The set of bits *n* that participate in parity check *m* are defined as $\mathcal{N}(m) = \{ n: H_{m,n} = 1\}$. Similarly the set of parity checks *m* that participate with bit *n* as $\mathcal{M}(n) = \{ m: H_{m,n} = 1\}$. The set $\mathcal{N}(m)$ with the bit *n* excluded is represented as $\mathcal{N}(m)\backslash n$, and the set $\mathcal{M}(n)$ with the parity check *m* excluded is represented as $\mathcal{M}(n)\backslash m$. The inputs to this algorithm are the prior probabilities of the bit being 0 and the bit being 1.

They are represented as

(36) Probability of bit being 0 = $P_n^0 = P(x_n = 0)$

(37) Probability of bit being 1 = $P_n^1 = P(x_n = 1) = 1 - P_n^0$

Where $n = 1\ldots..,N$.

3.5.1 Initialization

This is the first step of the decoding procedure. In this step the intrinsic message for each bit is calculated for each bit using the prior probabilities. These intrinsic messages are nothing but the log likelihood ratios of a bit explained in section 3.2. It is defined as

(38) $$\gamma_n = \log \frac{P_n^0}{P_n^1}$$

For each $(m, n) \in \{(i, j) \mid H_{i,j} = 1\}$, compute

(39) $$\alpha_{m,n} = sign(\gamma_n) \log \left(\frac{1+e^{-|\gamma_n|}}{1-e^{-|\gamma_n|}} \right)$$

Where

(40) $$sign(\gamma_n) = \begin{cases} +1, & \gamma_n \geq 0 \\ -1, & \gamma_n < 0 \end{cases}$$

Here $\alpha_{m.n}$ is called the extrinsic message. This value is calculated for each check in the parity check matrix, *H*. This value will be same for all the non-zero entries in the *H* matrix in each column. Since this value is computed from the intrinsic message which corresponds to the variable nodes, all checks in a particular column will be same. This is how the initialization of log-BP algorithm is done.

3.5.2 Check Node Computation

The next step is the check node computation, which is carried out for each check node. All the incoming messages from all the variable nodes to which a check node is connected are gathered together and the following computation is performed.

For each $(m, n) \in \{(i, j) \mid H_{i,j} = 1\}$, compute

(41) $$\beta_{m,n} = \log\left(\frac{1+e^{-\alpha}}{1-e^{-\alpha}}\right) \prod_{n' \in N(m)\backslash n} sign(\alpha_{m,n'})$$

Where

(42) $$\alpha = \sum_{n' \in N(m)\backslash n} | \alpha_{m,n'} |$$

Here $\beta_{m,n}$ is the extrinsic message that is computed for each check node and then passed to the variable nodes to which it is connected.

3.5.3 Variable Node Computation

Very similar to check node computation, variable node computation is done for each variable node. In this step all the extrinsic messages obtained from the check nodes to which a variable node is connected are gathered and the following computation is performed.

For each $(m, n) \in \{(i, j) \mid H_{i,j} = 1\}$, compute

(43) $$\alpha_{m,n} = sign(\gamma_{m.n}) \log\left(\frac{1+e^{-|\gamma_{m,n}|}}{1-e^{-|\gamma_{m,n}|}}\right)$$

Where

(44) $$\gamma_{m,n} = \gamma_n + \sum_{m' \in N(n)\backslash m} \beta_{m',n}$$

For each *n*, the algorithm updates the posterior log-likelihood ratio λ_n.

(45) $$\lambda_n = \gamma_n + \sum_{m \in M(n)} \beta_{m,n}$$

3.5.4 Hard Decision

This is the final step of the decoding procedure where we decide whether the decoded bit is 0 or 1 based on the posterior log-likelihood ratio computed in equation (45). This step performs the hard decision on $\{\lambda_1, \ldots\ldots, \lambda_n\}$ to obtain the decoded codeword. The decoded codeword is denoted as shown in the below equation.

(46) $$\hat{X} = \{\hat{x}_1, \ldots\ldots, \hat{x}_n\}$$

The hard decision on decoding the codeword bits is done using the following criterion.

(47) $$\hat{x}_n = 0 \;\; if \;\; \lambda_n > 0 \;\; and$$
$$\hat{x}_n = 1 \;\; if \;\; \lambda_n \leq 0$$

Thus if the posterior log-likelihood ratio for a particular bit in the codeword is greater than 0 then that bit id decoded as 0. Similarly if the posterior log-likelihood for a particular bit is less than or equal to 0, then that bit is decoded as 1. These variable node and check node computations are iteratively done until all the parity checks of the parity check matrix is satisfied, i.e :

(48) $$H.\hat{x}_n = 0$$

Once the above equation is satisfied the algorithm terminates, otherwise it iterates until the maximum number of iterations specified.

CHAPTER 4

DECODER DESIGN

4.1 LDPC Code Design

Construction of the low density parity check (LDPC) code is the first major thing to be done in the design of LDPC code decoder. A LDPC code is known as *(j, k)* – regular LDPC code if each variable node has a degree of *j* and each check node has a degree of *k,* this means that each variable node is connected to *j* check nodes and each check node is connected to *k* variable nodes [7]. The code rate of a regular LDPC code is defined as 1 –*j/k*, provided the parity check matrix is considered to have full rank. In this thesis, a 9216 bit codeword, with the code rate of – 1/2 (3, 6) – regular LDPC code is constructed. Here 3 denotes the variable node degree and 6 the check node degree. The construction of LDPC code is typically random. As explained in section 2.5, there are two different methods of LDPC codes construction, random and deterministic. Random construction is chosen for its simplicity and sparseness.

In the decoding process messages are exchanged along the edges and computed at variable and check nodes. If the constructed LDPC code is a good design then the algorithm will result in bits confluence exponentially to correct bits after certain number of iterations. To understand the efficient constructions of LDPC code there are two terminologies involved.

They are

i) Cycle

ii) Girth

A cycle in the Tanner graph is defined as a closed path that traversals through a set of *g* variable nodes and *g* check nodes without going through the same edge again. The length of this cycle is 2*g*. The length of cycles which are short is known as Girth.

If there are too many short cycles in the graph, then the belief propagation decoding algorithm will converge very slowly and may even result in wrong decoded results. Hence to increase the Tanner graph girth is very important in the construction of the LDPC code. To design LDPC codes that can be easily realized in VLSI implementation is very important. Several methods [6-11] have been developed which talks about the joint design of code construction and the hardware realization. The basic idea proposed by all these methods is to construct a base matrix of size M_b x N_b, in which each non zero element is expanded by a square permutation matrix of size L x L. This resulting M x N matrix is known as the expanded matrix. To obtain a code word length of 9216 bits, the constructed parity check matrix should have 9216 columns and 4608 rows which correspond to the variable nodes and check nodes. Direct mapping of such a big parity check matrix to the hardware is called fully parallel decoder. But designing such a decoder will be highly complex and consume more area. Hence partly parallel decoder design is opted. In this type of design, first a base matrix is constructed and then this base matrix is mapped to the hardware, which in turn consumes less area. Then this base matrix is expanded to a bigger matrix by a factor known as folding factor. Such expanded base matrix is the parity check matrix. The construction of the base matrix and the expanded matrix is explained in the following sections.

4.1.1 Base Matrix

A (3, 6) – regular LDPC code with the base matrix of size 18 x 36 is constructed in this thesis. This base matrix is the parity check matrix which will be expanded by a particular folding factor. Since it is a (3, 6) regular LDPC code, every column of this matrix will have 3 checks, i.e. 3 non zero entries. Similarly every row will have 6 checks. The LDPC software explained in section 3.1 was used to construct this base matrix. The construction of this matrix is purely random. This software accepts many arguments like size of the matrix to be generated, check

node degree, variable node degree and the method of constructions. This base matrix is constructed with following parameters

i) Size – $M_b \; x \; N_b$ = 18 x 36

ii) Variable node degree - j = 3

iii) Check node degree – k = 6

iv) Method – evenboth

$$H = \begin{bmatrix}
000011100000000001000001000000001000 \\
000101000000101000000010000000000010 \\
010000100000100001000000000100000001 \\
001000000110000000000000010000010100 \\
000000010011000000000000000011000001 \\
000000000001000100011000001100000000 \\
011000000000000000000000000001101001 \\
000000001000010000000011000010010000 \\
110100000000000010000011000000000000 \\
000000001100000100100010010000000000 \\
000000000010000000101000010000010010 \\
000000000000000000010000010110100100 \\
001010000100000010001100000000000000 \\
000000010001010000000100101000000000 \\
100100000000011110000000000000000000 \\
100011100001000000000000000000100000 \\
000000000000010101001100000000000010 \\
000000011000000000100000000001001100
\end{bmatrix}$$

Figure 4.1 (3, 6)-Regular LDPC Code – 18 x 36 Base Matrix –Dense Format

0: 4 5 6 17 24 32
1: 3 5 12 14 22 34
2: 1 6 12 17 27 35
3: 2 9 10 25 31 33
4: 7 10 11 28 29 35
5: 11 15 19 20 26 27
6: 1 2 29 30 32 35
7: 8 13 22 23 28 31
8: 0 13 17 23 24
9: 8 9 15 18 23 26
10: 10 18 20 25 31 34
11: 19 25 27 28 30 33
12: 2 4 9 16 20 21
13: 7 11 13 21 24 26
14: 0 3 13 14 15 16
15: 0 4 5 6 12 30
16: 14 16 18 21 22 34
17: 7 8 19 29 32 33

Figure 4.2 (3, 6)-Regular LDPC Code – 18 x 36 Base Matrices –Sparse Format

With these above mentioned parameters the base matrix is constructed. Since the method used to construct was evenboth method, the software generated a (3, 6) regular LDPC code. This matrix is then used to design the decoder architecture, which means that the Tanner graph of this matrix is directly mapped to the hardware implementation. But to achieve higher codeword length we expand this base matrix using the folding factor.

4.1.2 Folding Factor

Consider the base matrix shown in figure 4.1, each element in this base matrix can be substituted by a square permutation matrix or sub matrix. All checks, i.e. 1's are expanded by L x L identity matrix and similarly all zeros are expanded by L x L zero matrix. Thus the size overall expanded matrix is 18L x 36L. This value of L, the size of the sub matrix is known as folding factor.

4.1.3 Expanded Matrix

The base matrix shown in Figure 4.1 can be expanded to a bigger matrix using the folding factor defined in the previous section. But in order to improve the performance of the LDPC codes, the Tanner graph girth should be increased. In other words, the graph should not contain too many short cycles which reduce the performance of the LDPC code. Hence to achieve better performance and increased girth, the L x L identity matrix which will be substituted in base matrix, has its columns cyclically shifted by an amount $P_{i,j}$. There are different methods to determine the cyclically shifted value of this identity matrix. The method followed here is the one proposed by Zhang and Parhi [6].

(49)
$$I_{x,y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Consider the identity matrix in equation (49), this identity matrix can be substituted to a 1 at position *(i, j)* in the base matrix with the cyclically shifted value P computed by the following expression

(50)
$$P = (i-1).j$$

For example the cyclically shifted identity matrix for a check at position (2, 3), for the matrix in equation (49) is:

(51) $$P = (2-1).3 = 3$$

The cyclically shifter identity matrix with P equal to 3 is as show in the equation (52).

(52) $$I_{2,3} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Thus every check in the base matrix is replaced by the L x L cyclically shifted identity matrix, and every zero is replaced by L x L zero matrix. Such an expanded matrix is called the parity check matrix.

4.2 Finite Precision

The intrinsic messages and the extrinsic messages are probability ratios which are real numbers. The mapping of real numbers to the hardware is one of the challenging tasks. Real numbers can be represented in two ways, fixed point notation and floating point notation. Floating point notation is the most commonly used format to represent real numbers. The single precision floating point is 32 bit wide, with one bit for sign, 8 bits for exponent and 23 bits to represent the fractional part. The floating point format implementation of extrinsic and intrinsic messages in the decoder design will result in high hardware complexity. Since the log-likelihood ratio of every received bit of the codeword requires 32 bit vector, the floating point notation of real number is not followed in this decoder design. The best way to represent the extrinsic and intrinsic messages is fixed point notation. This gives the advantage of choosing how many bits of

precision we need for the design. Hence the hardware complexity could be greatly reduced by using fixed point notation.

4.2.1 Quantization

The number of bits used to represent the messages can influence both decoding performance and the hardware complexity, hence quantization of the real numbers which represent the extrinsic and intrinsic messages are very important. Let *(q: f)* represent the quantization scheme; here *q* is the total number of bits used in the fixed point notation, in which *f* represent the number of bits used for fractional part. In this design there are two quantization schemes followed.

They are

i) (10: 5) Quantization scheme for intrinsic messages (prior log-likelihood ratios)

ii) (8:5) quantization scheme for extrinsic messages.

One bit is used to represent the sign; four and two bits are used to represent the magnitude of intrinsic and extrinsic messages respectively. The fractional part of both intrinsic and extrinsic messages is represented by 5 bits of precision. All extrinsic messages that are exchanged between the variable nodes and check nodes follow this quantization scheme. Since this design follows five bits of precision, the decoding performance is better and reduces the number of iterations to decode the codeword.

4.2.2 Look up Table - LuT

The Log Belief Propagation decoding algorithm uses the below function *f(x)* expressed in equation (53) for all the check nodes and variable nodes computations. This function can easily be implemented in the design using a LuT. This LuT can be implemented as a combinational block in hardware.

(53) $$f(x) = \log\left(\frac{1+e^{-|x|}}{1-e^{-|x|}}\right)$$

Here x is the value to look up in the table. Since the quantization scheme is known and it is constant the range for the above function can be computed and stored in the LuT.

4.3 Decoder Architecture

The iterative belief propagation algorithm is inherently fully parallel. This gives the advantage of great decoding speed. This high speed decoding implementation in hardware is one of the most crucial issues faced during the design of the LDPC decoder. The most efficient way of implementation, is the direct mapping of belief propagation algorithm to hardware. All check nodes and variable nodes are assigned to individual check node and variable node processors and are connected through an interconnection network which reflects the Tanner graph. By completely using the parallelism of belief propagation algorithm, a fully parallel decoder can achieve very high decoding throughput and speed. The main drawback of such a fully parallel design is that with the increase in the codeword length, the LDPC code length increases which in turn reflects in the hardware complexity. Thus a high speed partly parallel decoder with appropriate trade-offs between hardware and decoding throughput is desired.

4.3.1 Partly Parallel Decoder

Based on the parity check base matrix a (3, 6) – regular LDPC code, a partly parallel decoder is designed. The overall decoder architecture is shown in Figure 4.3. There are three major units in this decoder design. They are

i) Processing elements - PE

ii) Check node unit – CNU

iii) Shuffle network and address generator – AG

This decoder is designed with the base matrix described in section 4.1.1. It has 36 variable node units (VNU) and 18 check node units (CNU) which correspond to the columns and rows of the base matrix. Since it is (3, 6) – regular LDPC code, each VNU is connected to 3 check nodes and each CNUs is connected to 6 VNUs. Every VNU is associated with 5 different RAM blocks. These 5 RAM blocks together with the VNU form one processing element (PE). Hence there are 36 PEs in this decoder design. Since each VNU is associated with 3 CNUs, it has 3 RAM blocks to store the extrinsic messages exchanged between each CNU separately. The five different RAM blocks in each PE are as follows

i) INIT RAM – Used to store intrinsic messages

ii) RAM 1 – Used to store extrinsic messages between VNU and 1st CNU

iii) RAM 2 – Used to store extrinsic messages between VNU and 2nd CNU

iv) RAM 3 – Used to store extrinsic messages between VNU and 3rd CNU

v) DEC RAM – Used to store the decoded message.

As described in section 4.1.2, the folding factor *L*, is used to expand this base matrix. Since each check is expanded by *L* x *L* cyclically shifted identity matrix, it has *L* variable nodes and *L* check nodes. These *L* variable nodes and check nodes comprise one variable node group VG and one check node group CG respectively. Each VG is assigned to one VNU and similarly each CG is assigned to one CNU. Thus achieving the folding factor *L*, the expanded variable nodes is 36*L* and check nodes is 18*L*. In other words, *L* variable nodes are assigned to one VNU for variable node computation and *L* check nodes are assigned to one CNU. Each PE unit is associated with one address generator (AG), this AG is used to access the memory locations of the RAMs in it. And the shuffle network is a bidirectional realize that connectivity between the VNUs and CNUs reflecting the Tanner graph of the base matrix. Each RAM block in the PE has *L* memory

locations. These L memory locations are used to store the extrinsic and intrinsic messages that are exchanged between the L variable nodes in VG and its neighboring check nodes in CG. The nodes in the VG are denoted as V_d and the messages associated with this node is always store at address *d-1*. The type design strategy followed in this design greatly simplifies the control logic for generating the memory access address. This decoder design completes one decoding iteration in $2L$ clock cycles. In the first L clock cycle the decoding performs the entire check node processing and in the second L clock cycle it performs the variable node processing. During the check node processing the decoder performs the check node computation and also updates the extrinsic messages in the RAMs of the PEs. As mentioned in section 4.2.1, all the intrinsic and extrinsic messages are quantized to 8 bits. The architecture of the CNU and VNU is a direct adaptation of the CNU and VNU architectures presented by Zhang and Parhi [6] for a (3, 6) regular LDPC code. Figure 4.4 shows the data paths between one VNU and its 3 neighboring CNUs together with the bidirectional shuffle network that realizes the Tanner graph connectivity.

4.3.2 Check Node Processing

The decoder performs all the check node computations in the check node processing stage. Here it realizes the extrinsic message exchange between the neighboring nodes. During the beginning of check node processing, the RAMs in PE contain the 9 bit hybrid data. This hybrid data includes one bit hard decision and 8 bits of variable-to-check extrinsic messages associated with all L variable nodes in VG. At every clock cycle the decoder performs *read – exchange– compute – exchange – write* operations to convert one variable-to-check extrinsic message in each RAMs to its check-to-variable extrinsic message.

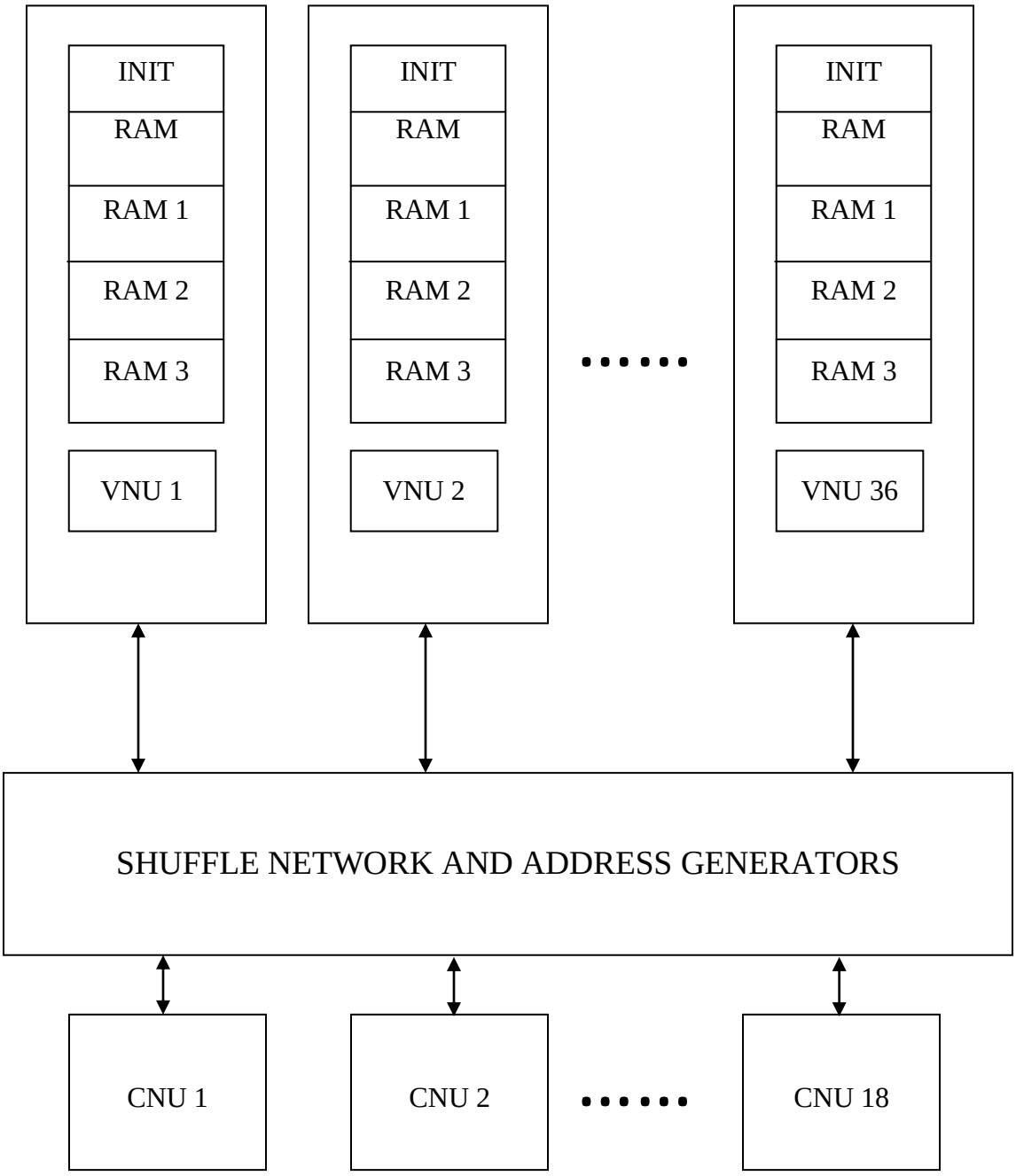

Figure 4.3 The Principal (3, 6) Partly Parallel Decoder Architecture

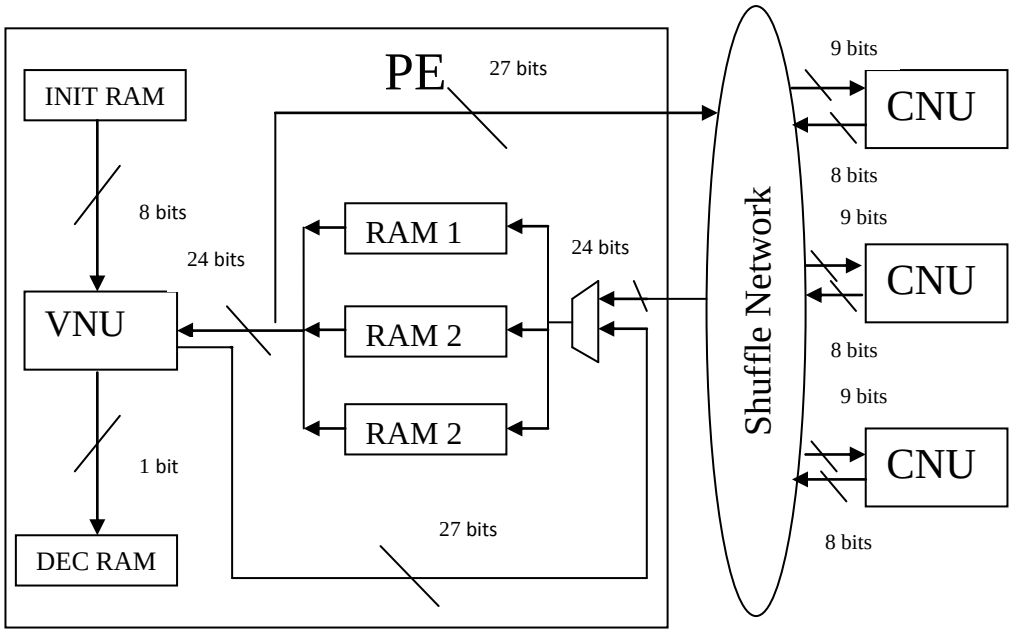

Figure: 4.4 Data Path of Iterative Decoding

The check node processing involves following operations

i) *Read:* Reads 9 bit hybrid data from each RAM in PE which stores the extrinsic messages

ii) *Exchange:* Each 9 bit hybrid data is passed from the VNUs to CNU through the shuffle network.

iii) *Compute:* Check node computation is performed with all 6 9-bit received variable-to-check extrinsic messages, thus generating 6 8-bit check-to-variable extrinsic messages.

iv) *Exchange:* Sends the generated check-to-variable extrinsic messaged to PEs.

v) *Write:* Writes the check-to-variable extrinsic messages to the RAMs in the PEs.

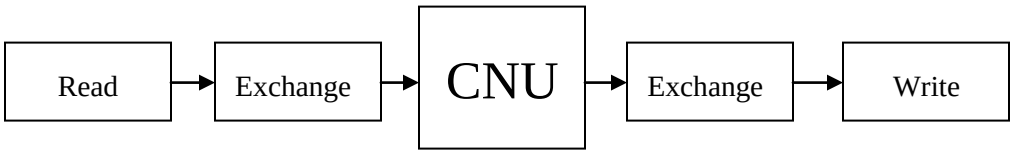

Figure 4.5 Check Node Processing

These above mentioned operations summarize the whole check node processing. The exchange of extrinsic messages between CNUs and VNUs realized with two distinct set of

wires with opposite direction. In other words there are two separate wires to carry variable-to-check extrinsic messages and check-to-variable extrinsic messages.

4.3.3 Variable Node Processing

Variable node processing is very similar to check node processing. Here it performs check node computations with the 8-bit extrinsic messages gathered from its three neighboring check nodes and generates 3 9-bit variable-to-check extrinsic messages.

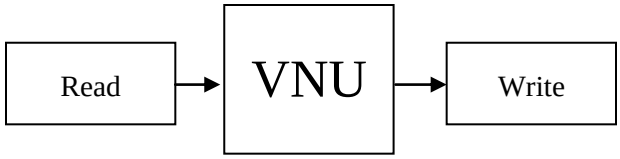

Figure 4.6: Variable Node Processing

At the beginning of the variable node processing the RAMs in the PE hold the check-to-variable node extrinsic messages which were computed in the first *L* clock cycle.

At each clock cycle it performs *read-compute-write* operations. They are summarized as follows

i) *Read:* Reads the 8-bit check-to-variable extrinsic messages from the three RAMs associated in its PE.

ii) *Compute:* Performs the variable node computation and generates the 9-bit hybrid variable-to-check extrinsic message with one bit hard decision contained in it.

iii) *Write:* The variable-to-check messages are written back to the same memory location from where it was read. Since both the extrinsic and intrinsic messages are associated with the same variable node are stored in the same memory location, one binary counter in enough to generate the address for all RAMs in that PE.

4.3.4 Shuffle Network and Address Generator

The shuffle network together with the AG is the one which realizes the connectivity among the VNUs and CNUs. This network exactly follows the connectivity that exists between the variable nodes and check nodes of the Tanner Graph of the constructed base matrix. The extrinsic messages associated with the node $V_{d \text{ are}}$ always stored at *d-1*. Exploiting the explicit structure of the *H* matrix, the implementation of the AG and the shuffle network could be achieved easily. The implementation of the AG can be done using a simple binary counter. This binary counter starts counting with and offset value at every check node processing. This offset value is corresponds to the function $P_{i,j}$, by which the cyclic shifting is done. Similarly the binary counter starts counting from zero during variable node processing.

4.4 Check Node Unit Architecture

Every CNU has *L* check nodes assigned to it. It performs one check node computation at

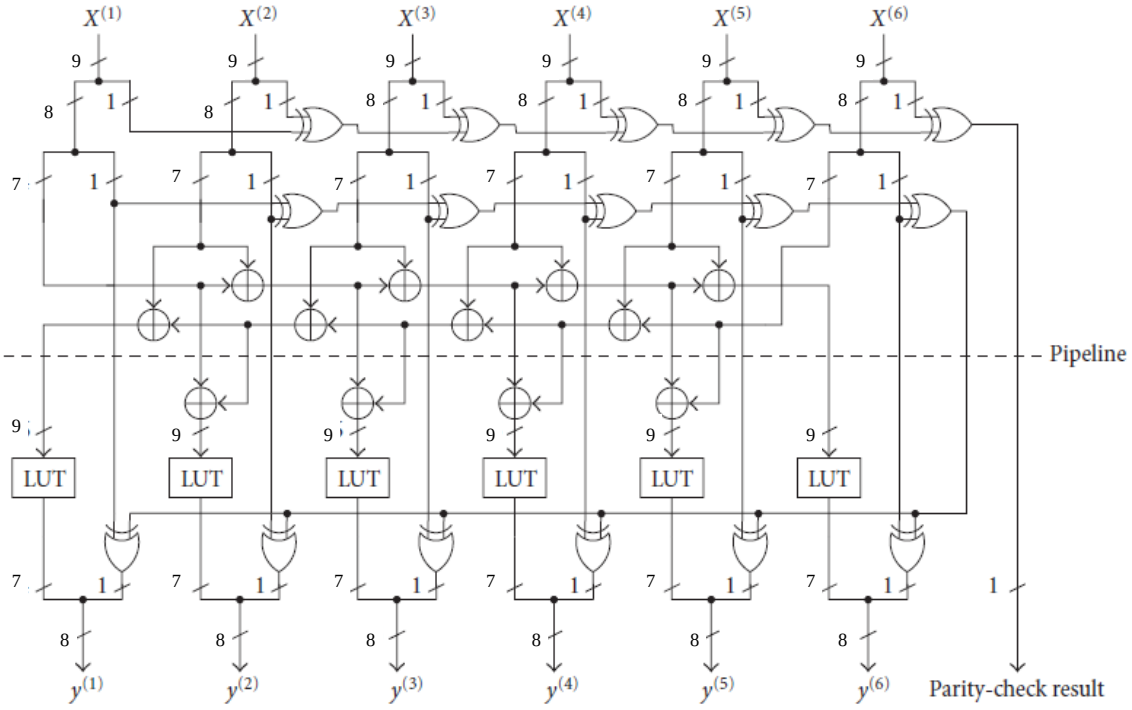

Figure 4.7 CNU Architecture – Taken from Zhang and Parhi

one clock cycle. It takes *L* clock cycles to complete all the check node computations of *L* check nodes. The CNU architecture implemented in this design is the architecture proposed by Zhang and Parhi [6] for a (3, 6) regular LDPC code. The overall architecture of the CNU is shown in the figure 4.7 [6]. Implementation of CNU is fairly simple and can be realized using combinational logic block. The CNU has 6 9-bit inputs, the variable-to-check extrinsic messages coming from its neighboring six variable nodes. Then the 9-bit hybrid input is split into one bit hard decision and 8 bit variable-to-check extrinsic message data. Then this 8-bit data is separated from the sign bit and then goes to the adder. The hard decision bit which was split initially is used to check the parity by XORing all the 6 hard decision bits. The architecture shown above is fairly the direct implementation of log belief propagation algorithm. This design follows the check node computation exactly the way defined in the algorithm. LuTs are used to implement the function *f(x)* that is used in the algorithm.

4.5 Variable Node Unit Architecture

Similar to the CNU architecture the VNU architecture is also implemented using adders and LuTs.

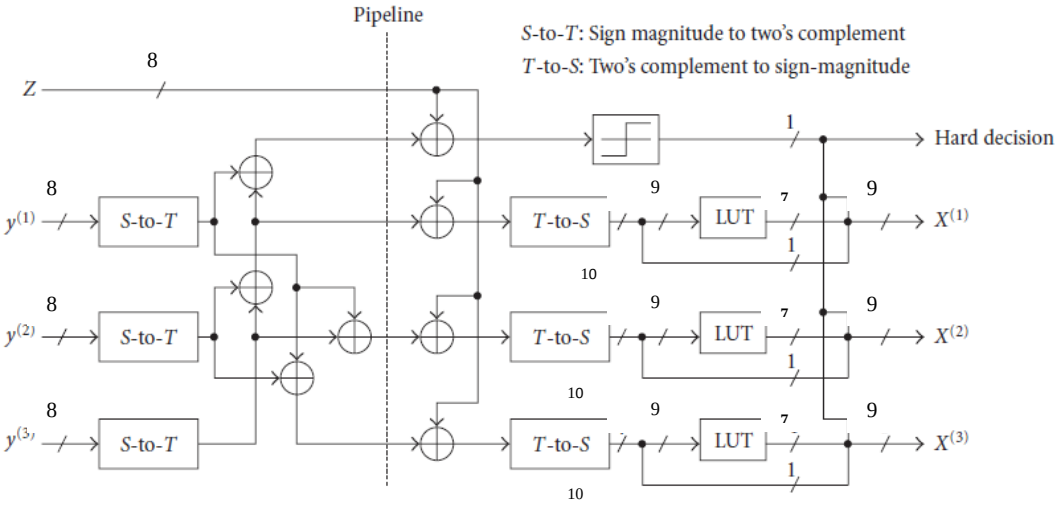

Figure 4.8 VNU Architecture – Taken from Zhang and Parhi

In addition to adders and LuTs, the VNU architecture has sign-magnitude to two's complement converter and two's complement to sign-magnitude converters. The summation performed in the variable node computation is sign dependant and hence the sign-magnitude format of data bits is converted into two's complement format which facilitates the summation of all the extrinsic messages. Figure 4.8 [6], shows the overall architecture of VNU. This VNU architecture is also a direct adaptation of the architecture proposed Zhang and Parhi [6].

CHAPTER 5

FPGA IMPLEMENTATION AND RESULTS

5.1 LDPC Code Construction

The construction of the low density parity check codes (LDPC) was achieved with the help of the LDPC software and MATLAB. The base matrix of size 18 x 36 was generated using the LDPC Software. As explained in section 4.1.1, this software takes few arguments like, *n*-bits, *m*-checks, and method and distribution degree. This base matrix is constructed with following parameter values

i) Size – M_b x N_b = 18 x 36

ii) Variable node degree - j = 3

iii) Check node degree – k = 6

iv) Method – evenboth

This base matrix is expanded to a bigger matrix using the folding factor, L. In this implementation, the folding factor value is L = 256. Thus achieving a codeword of length 9216-bit. Such an expanded matrix has 4608 rows and 9216 columns, which corresponds to 9216 bits and 4608 checks. This expansion was implemented using MATLAB. A Matlab program was created to expand the base 18 x 36 base matrix to 4608 x 9216 expanded matrix by replacing all zeros by a 256 x 256 zero matrix and all 1's by cyclically shifted 256 x 256 identity matrix. This program also takes care of cyclically shifting the identity matrix bases on the position of the check and then substitutes it in the base matrix.

5.2 AWGN Channel Simulation

The simulation of the transmission of bits through additive white Gaussian noise (AWGN) channel was done by the LDPC software. The expanded parity check matrix which

was constructed is given to the LDPC software to create its generator matrix *G*. Then the encoding of the source messages is done as explained in section 3.1.2. The encoded bits are then transmitted through the AWGN channel simulated by the LDPC software. The noise level of the channel represented by the standard deviation is chosen to be 0.4. Thus the data is received with the noise variance of 0.4. This received data is then converted into a fixed-point notation format, which is then given to the decoder input.

5.3 FPGA Implementation

This 9216-bit (3, 6)-regular LDPC code decoder is implemented using Xilinx Virtex-4 XC4VLX80 device with package FF1148. The entire decoder was designed in VHDL. Xilinx ISE 9.1 was used for developing, simulating and synthesizing the decoder model. The variable node unit (VNU) and check node unit (CNU) architectures were designed using VHDL procedures and functions. This device contains 200 on-chip block RAMs. Each block RAM is a dual port 18Kb, fully synchronous RAM. In this design, each processing elements (PE) has 3 RAMs and one INIT RAM, each has 256 memory locations of size 9 bits. Each of these RAMs requires 2286 bits to store 256 x 9 bit data. Each PE requires 4 RAMs of total size of 9144. Hence one dual port -18Kb block RAM of this device can be configured in two 9-bit single-port RAM. Hence every PE uses two 18Kb dual port RAM for its four RAMs and INIT RAM. Out of 200 available on-chip block RAMs, this design uses 72 block RAMs. Since the DEC RAM is a 256 x 1 bit RAM, it is implemented using a distributed RAM.

All the adders used in CNU and VNU architectures are implemented by ripple carry adder type, since this type of adder is very suitable for Xilinx FPGAs. All the LuTs used in both CNUs and VNUs are 170 x 7-bit ROM. These are easily implemented using Xilinx IP cores instantiation. According to this design, at every clock cycle the decoder performs both read and write

operations. Let *W* be the primary clock frequency, the design requires 2 x *W* as clock signals to RAMs to achieve read and write operations in single clock cycle. This clock signal is generated using DCM-digital clock manager of the device.

5.3 Results

The Xilinx ISE 9.1 project navigator tool is uses for simulation, timing analysis and synthesis purposes. The decoded output is written to file at the end of simulation. The static timing analysis tool reports the maximum clock at which the decoder can work. In this design the maximum clock frequency that is achieved is 48MHz. If the decoder performs *r* decoding iterations, then the total clock frequency is $2r.L + L$. Thus the maximum decoding throughput achieved will be

$$Throughput = \frac{W.36.L}{2r.L + L} \tag{54}$$

Here *L* is the folding factor, *W* is the clock frequency and *r* is the number decoding iterations. Since this decoder is designed with 5 bits of precision, the maximum number of iterations is 10, which is very less when compared to other implementations. Therefore the maximum throughput achieved can be computed by substituting the values of *L, r and W* in equation (54).

$$Throughput = \frac{W \times 36 \times L}{(2r \times L) + L} = \frac{W \times 36}{2r + 1}$$
$$\Rightarrow \frac{48 \times 36}{(2 \times 10) + 1} = 82Mbps \tag{55}$$

Thus a decoding throughput of 82 Mbps is achieved. The device utilization is shown in the Table 5.1. These results are obtained on synthesizing the design.

Table 5.1 Device Utilization Summary

RESOURCE	**NUMBERS**	**UTILIZATION RATE**
Slices	19,482	54 %
LUTs	27,336	33%
Block RAMs	72	36 %
Slice Flip Flops	16,892	22%
Bonded IOBs	92	12%
DCM	1	8%

CHAPTER 6

CONCLUSION

6.1 Conclusion and Future Work

Low Density Parity Check (LDPC) Codes has wide range of applications in the wireless field due to its high capacity performance. Thus the (3, 6)-regular LDPC code decoder is implemented on FPGA targeting Xilinx Virtex 4 XC4VLX80 device with package FF1148. The decoder was designed using VHDL in Xilinx ISE 9.1 EDA tool. The codeword length of 9216-bit is implemented with decoding throughput of about 82Mbps. The detailed decoder design and architecture are presented in previous chapter and the results are also explained. This research helped me to gain knowledge in three different fields namely, digital wireless communication, coding theory and digital design.

My current and future work is on LDPC codes for digital video broadcast using WiMax standard 802.16e at SAI Technology, Inc, Santa Clara US. I am working together with a team which investigates on optimizing the data transmission over wide range wireless medium, given that the transmitted data is only video signal.

BIBLIOGRAPHY

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. IT-8, no. 1, pp. 21–28, 1962.

[2] R. G. Gallager, *Low-density parity-check codes*, MIT Press, Cambridge, Mass, SA, 1963.

[3] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[4] M. C. Davey and D. J. C. MacKay, "Low-density parity check codes over GF(*q*)," *IEEE Communications Letters*, vol. 2, no.6, pp. 165–167, 1998.

[5] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low density parity-check codes using irregular graphs and belief propagation," in *Proc. IEEE International Symposium on Information Theory*, p. 117, Cambridge, Mass, USA, August 1998.

[6] T. Zhang and K. K. Parhi, "An FPGA implementation of (3, 6)-regular low-density parity-check code decoder," in *Proc. EURASIP. Applied Signal Processing. Syst.*, Jun. 2003, pp. 530–542.

[7] T. Zhang and K. K. Parhi, "VLSI implementation-oriented (3, k)-regular low-density parity-check codes," in *Proc. IEEE Worksh. Signal Process. Syst.*, Sep. 2001, pp. 5–36.

[8] E. Boutillon, J. Castura, and F. R. Kschischang, "Decoder first code design," in *Proc. 2nd International Symposium on Turbo Codes and Related Topics*, pp. 459 –462, Brest, France, September 2000.

[9] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, JohnWiley & Sons, New York, USA, 1999.

[10] D. E. Hocevar, "LDPC code construction with flexible hardware implementation," in *Proc. Int. Conf. Commun.*, 2003, pp. 2708–2712.

[11] A. Delvarathinam, E. Kim, and G. Choi, "Low-density parity-check decoder architecture for high throughput optical fiber channels," in *Proc. Int. Conf. Comp. Design*, 2003, pp. 520–525.

[12] Narayanan, K.R., Xiaodong Wang, and Guosen Yue.. "LDPC code design for MMSE turbo equalization." *Proceedings of the IEEE Information Theory Workshop*. Oct. 2002

[13] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[14] D. J. C. Mackay, S. T. Wilson, and M. C. Davey, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Comm.*, vol. 47, pp. 1449–1454, Oct. 1999.

[15] T. Tian, C. Jones, J. D. Villasenor, and R. D.Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. Int. Conf. Commun.*, vol. 5, May 2003, pp. 3125–3129.

[16] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2001, pp. 25–29.

[17] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Efficient implementation of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2001, pp. 1036–1036.

[18] T. Tian, C. Jones, J. D. Villasenor, and R. D.Wesel, "Construction of irregular LDPC codes with low error floors," in *Proc. Int. Conf. Commun.*, vol. 5, May 2003, pp. 3125–3129.

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

[20] T. Richardson, “Error floors of LDPC codes,” in *Proc. 41st Allerton Conf. Commun. Contr. Comput.*, Oct. 2003, pp. 1426–1435.

[21] M. M. Mansour and N. R. Shanhhag, " High-throughput LDPC decoders ", *IEEE Trans. on VLSI Systems*, vol. 11, no. 6, pp. 976-996, December 2003.

[22] IEEE Std 802.16e-2005, “IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems”, Feb. 28th, 2006.

[23] Shannon, C.E. *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press. 1948

[24] Viterbi, A.J. “Error bound for convolutional codes and an asymptotically optimum decoding algorithm.” *IEEE Transactions on Information Theory* no. 13: pp 260–269. April 1967.

[25] Forney, G.D. “The Viterbi algorithm.” *Proceedings of the IEEE* 61:268–278. March 1973.

[26] Tanner, R. “A recursive approach to low complexity codes.” *IEEE Transactions on Information Theory.* no. 27: pp 533–547. Sept 1981.

[27] Berrou, C., A. Glavieux, and P. Thitimajshima. “Near Shannon limit error-correcting coding and decoding: Turbo-codes.” *International Conference on Communication.* May 1993,

[28] MacKay, D.J.C, and R.M. Neal.. “Good codes based on very sparse matrices.” *5th IMA Conference on Cryprography and Coding.* Berlin, Germany: Springer. 1995

[29] Pearl, J. *Probabilistic Reasoning in Intelligent Systems* : *Networks of Plausible Inference*. San Mateo: CA : Morgan Kaufmann. 1988.

[30] Sipser, M., and D.A. Spielman.. “Expander codes.” *IEEE Transactions on Information Theory* no 42: pp. 1710–1722. Nov 1996

[31] Etzion, Tuvi, Ari Trachtenberg, and Alexander Vardy.. “Which codes have cycle-free Tanner graphs ?” *IEEE Transactions on Information Theory*, vol. 45. Sept. 1999

[32] MacKay, D.J.C, S.T. Wilson, and M.C. Davey. "Comparison of constructions of irregular Gallager codes." *Transaction on Communications* no 47: pp 1449–1454. Oct 1999.

[33] Campello, J., and D.S. Modha.. "Extended bit-filling and LDPC code design." *Global Telecommunications Conference* pp : 985–989. Nov 2001.

[34] Lafferty, J., and D. Rockmore.. "Codes and iterative decoding on algebraic expander graphs." *International Symposium on Information Theory and its Applications*. Honolulu, Hawaiin, U.S.A. Nov 2000.

[35] Rosenthal, J., and P. O. Vontobel. "Constructions of regular and irregular LDPC codes using Ramanujan graphs and ideas from Margulis." *Proceedings of the IEEE International Symposium on Information Theory*. 4. June 2001.

[36] MacKay, David J.C., and Michael S. Postol.. "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check Codes." *Electronic Notes in Theoretical Computer Science*, Volume 74. Elsevier. 2003.

[37] Lucas, R., M.P.C. Fossorier, Yu Kou, and Shu Lin. "Iterative decoding of one step majority logic deductible codes based on belief propagation." *Communications, IEEE Transactions* on no .48 pp :931–937. June 2000.

[38] Kou, Y., S. Lin, and M.P.C. Fossorier "Low-density parity-check codes based on finite geometries : A rediscovery and new results." *Transactions on Information Theory* no 47 pp: 2711–2736. November 2001.

[39] Johnson, S.J., and S.R.Weller.. "High-rate LDPC codes from unital designs." *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM '03*. Dec. 2003

[40] Ammar, B., B. Honary, Y. Kou, and S. Lin.. "Construction of low density parity check codes: a combinatoric design approach." *Proceedings of the IEEE International Symposium on Information Theory*. July 2002

[41] Johnson, S.J., and S.R.Weller. "Resolvable 2-designs for regular low-density parity-check codes." *Communications, IEEE Transactions* no 51 pp:1413 – 1419. Sept. 2003

[42] Vasic, B., I.B. Djordjevic, and R.K. Kostuk. "Low-density parity check codes and iterative decoding for long-haul optical communication systems." *Lightwave Technology*, Journal no 21 pp :438 – 446. Feb. 2003

[43] Miller, G., and G. Cohen. "The rate of regular LDPC codes." *IEEE Transactions on Information Theory*, no 49 pp: 2989–2992. Nov 2003.

[44] LDPC software,
http://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/index.html

[45] LDPC software - parity check matrix,
http://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/pchk.html

[46] LDPC Software – encoding techniques,
http://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/encoding.html

[47] LDPC Software – decoding techniques,
http://www.cs.toronto.edu/~radford/ftp/LDPC-2006-02-08/decoding.html

[48] Bernard Skalar "What every communications engineer should know about error-correction coding" *Communications Design Conference,* April 2004.

[49] Baylis, J. *Error Correcting Codes: A Mathematical Introduction.* Boca Raton, FL: *CRC Press*, 1998.

[50] Jianming Wu. "Performance analysis for LDPC-coded modulation in MIMO multiple-access systems", *IEEE Transactions on Communications*, July 2007.