High Throughput, Fully Parallel, Pipelined
FPGA Implementation of LDPC Decoder

By Maarij Raheem

A thesis submitted in fulfillment of the requirements for the degree of Masters of Science in the Embedded System Research Group, Graduate School of Science and Engineering, Karachi Institute of Economics and Technology

Dr Husain Parvez, Faculty Advisor

Dec 02, 2016
Declaration of Authorship

I Maarij Raheem hereby declares that this thesis High Throughput, Fully Parallel, Pipelined FPGA Implementation of LDPC Decoder and the work presented in it are my own.

Signed: _______________

Dated: ________________
Abstract

Forward error correction Techniques have extensive applications in wireless LANs, deep space communications, digital broadcasting, and cellular communications. Low-Density Parity Check decoders (LDPC) have made their way in communications systems, because of robust performance in error correction and strong competency to parallel implementation in hardware.

In this thesis we have proposed a FPGA based architecture of LDPC decoder by exploring the high parallelism, flexibility and the computation speed of the FPGAs. The proposed design is fully parameterized to adopt any variation of LDPC decoder. The key characteristics of the proposed design is the streaming input support, low latency, moderate resource utilization and a very high throughput of 65Gbps. The previous highest throughput for FPGA implementation of LDPC decoder was 16.2 Gbps. We have shown that high throughput can be achieved by implementing pipelining in the decoder such that the design can handle streaming input support. Also reducing the routing complexities in the design and using appropriate design parameters helped increasing the throughput. In this thesis we have also implemented a fixed point implementation strategy for resource optimization. The proposed design is also tested and verified in hardware.
“The less I know the, the less I know, the less I know,

The more I know, the more I know, the less I know.”

- Unknown.
I would like to dedicate this work,

to my beloved parents,

who are real motivation behind my every success.
Acknowledgement

I owe a debt of gratitude to all who supported me in this research work. I would like to thank my supervisor Dr. Husain Parvez (in capital letters) for his support and encouragement during the whole period of this thesis and during my fellowship under his supervision. I would also like to mention PAF KIET management especially Sadat sahib for guiding me through the policies and formalities of the system. Also, I am thankful to ICT R&D for providing me stipend and funding for my MS. I would also like to name my colleagues Baqar Raza, Ali Asghar, Amin Qureshi, Waqar Ahmed, Adeel Amin and Ali Umair for sharing their piece of knowledge and feedback throughout the course work and this research work. Lastly, I am thankful for the unmatchable love of my family who remained patient and supportive for the entire period of research.
Contents
Declaration of Authorship................................................................. 2
Abstract ............................................................................................. 3
Acknowledgement ............................................................................... 6
Table of figures .................................................................................. 9
Chapter 1 Introduction ......................................................................... 11
Chapter 2 Literature Review ................................................................. 14
Chapter 3 Low Density Parity Check Decoder ........................................ 17
  3.1 Hard Decision Decoders .............................................................. 19
  3.2 Soft decision decoders ................................................................ 19
    3.2.1 Sum Product decoding Algorithm ......................................... 20
    3.2.2 Min Sum Decoding Algorithm .............................................. 21
  3.3 LDPC decoding Flow .................................................................. 22
  3.4 Applications of LDPCs ................................................................. 23
    3.4.1 Wireless and Mobile Communications .................................... 23
    3.4.2 Deep Space Communications .............................................. 24
    3.4.3 Satellite Communications .................................................. 24
    3.4.4 Military Communications .................................................. 24
    3.4.5 Data Storage ....................................................................... 24
Chapter 4 Hardware Implementation ..................................................... 25
  4.1 LDPC Top ................................................................................. 25
    4.1.1 Architecture of LDPC top level module ............................... 25
    4.1.2 Latency of LDPC top level module ...................................... 28
    4.1.3 Parameterization of top level module for LDPC ................. 29
  4.2 Message Quantizer ..................................................................... 29
  4.3 Check Nodes ............................................................................. 30
    4.3.1 Architecture of Check Nodes .............................................. 30
    4.3.2 Latency ............................................................................. 32
    4.3.3 Parameterization for top level module for check nodes ....... 32
  4.4 Bit nodes .................................................................................. 32
    4.4.1 Bit nodes Architecture ...................................................... 33
4.4.3 Bit nodes designing, Parameterization and reducing the hardware cost

4.4.4 Parameters for bit nodes

4.5 Check Message translator

4.6 Bit Message Translator

4.7 Syndrome

4.7.1 Architecture

4.7.2 Latency of Syndrome Block

4.7.3 Parameterization of Syndromes

4.7.4 Resource optimization of Syndrome

Chapter 5 Results, Analysis and Commentary

5.1 Error Correction Performance

5.2 FPGA Implementation

5.2.1 Intel FPGA Synthesis and Implementation

5.2.2 Xilinx FPGA Synthesis and Implementation

5.3 Comparison with other implementations

5.4 Effects of Parameters on Resource Utilization VS Decoding Performance

5.4.1 Architecture of LDPC Decoder

5.4.2 Message Width

5.4.3 Pipelining

Chapter 6 Conclusion

6.1 Conclusion

6.2 Future work

Bibliography
Table of figures

Figure 1: Characteristics and parameters of FPGA based LDPC decoding system .............................................. 12
Figure 2: Tanner graph representing bit nodes and check nodes ...................................................................... 17
Figure 3: Flow chart of Generalized LDPC Decoding Algorithm ...................................................................... 23
Figure 4: Block diagram of LDPC top level module .......................................................................................... 27
Figure 5: Block diagram of Top level of Check nodes ..................................................................................... 30
Figure 6: Block diagram of Top level of Bit nodes ........................................................................................... 34
Figure 7: BER vs SNR (db) graph for 128 bits Codeword decoder LDPC .......................................................... 41
Figure 8: Throughput comparison between implementations discussed above. ............................................. 44
Figure 9: Comparison between timing closures of different hardware implementations of LDPC discussed above..................................................................................................................... 44
Figure 10: Resource Utilization comparison between LDPC decoder FPGA implementations discussed above .............................................................................................................................................. 45
Table of tables

Table 1: table showing various parameters and characteristics of hardware implementations available in literature .......................................................... 16
Table 2: BER vs SNR (db) table for 128 bits Codeword decoder LDPC.......................................................... 41
Table 3: FPGA resource summary for LDPC implementation for Arria 10 device .............................................. 42
Table 4: FPGA resource summary for LDPC implementation for Vertex Ultra scale device......................... 43
Table 5: Table showing comparison between different fully parallel hardware implementations of LDPC .......................................................... 43
Chapter 1 Introduction

The importance of FECs can be guessed by the fact that currently a lot of active research within the communications community is dedicated on Forward Error Correction (FEC) codes. Errors in the transmitted information over noisy communication channels and systems can be corrected by employing FEC techniques. The focus of the latest research in the domain of FECs is to design a coding schemes that may approach to Shannon capacity and to reduce its complexity to make its hardware solution more cost effective.

Low-Density Parity Check (LDPC) codes is a type of FECs and were proposed by Gallager in 1962 [1], at that time LDPC was thought to be too complex to realize its implementation and simulation. Therefore these codes remained untouched for as long as two and a half decades until 1996 when LDPC was rediscovered by Neal and Mackay [2]. Thanks to the many fold improved computation power available today, LDPC codes have become a vital part of many commercialized communication systems, including CCSDS [3], DVB-S2 [4], WiMAX [5], and Wi-Fi [6].

LDPC codes have numerous attractive features that make them very appealing for putting them into practice. The LDPC can be simplified to use low complexity equations, with a little loss in decoding efficiency. These low complexity equations reduce the implementation cost of computational hardware. LDPC are the iterative decoding codes that are considered to be the nearest to the theoretical limits for the higher codeword lengths [7].

To design the LDPC on FPGA we need to implement the basic design blocks like bit nodes, check nodes, syndrome etc. which are relatively simple. But designing the complete decoding system is a matter of composite compromises between various system characteristics like hardware resource utilization, processing throughput, processing latency, bandwidth efficiency, correction capability, design flexibility and run time reconfiguration. The compromises between these characteristics are ruled by some design
parameters like maximum iterations for decoding, block lengths, and the type of LDPC code used, and most importantly the architecture of the LDPC decoding hardware. The relation between these characteristics and parameters is shown in Figure 1 [8].

![Figure 1: Characteristics and parameters of FPGA based LDPC decoding system](image)

Parameters are the factor that can be changed at the design time. Whereas the characteristics are the attributes of the design ruled by the parameters. The goals of our thesis is to present an architecture that can offer the following characteristics:

1. To increase the processing throughput
2. Reduce the processing latency
3. Keep the hardware requirements as low as possible
4. Make the design flexible enough so that it can adopt any LDPC decoder with least possible effort
The structure of this thesis is as follows. Chapter 2 discusses the literature review on the FPGA implementation of LDPC decoders. Chapter 3 discusses the LDPC decoding algorithm and its applications. Chapter 4 discusses the proposed FPGA based LDPC Architecture in detail. And explains the pipelining, processing latency, parameterization and resource optimization in details. Chapter 5 discusses the results and provides analysis and commentary on the results. Chapter 6 concludes the thesis with recommendation for the future work.
Chapter 2 Literature Review

It is a challenging task to compare all the LDPC decoding system based on FPGA designs present in published research works. This chapter provides a summary of current literature present on the main attributes involved in LDPC decoding systems based on FPGA. Both industrial and academic published designs are compared, discussed, characterized and illustrated the related performance trade-offs in depth.

The survey paper [8] is the detailed survey of all the FPGA implementations of LDPC. The paper discusses the implementations in the light of various parameters like LDPC code, Algorithm, Architecture and No. of iterations. And also discusses the characteristics of the implemented hardware liked flexibility, bandwidth efficiency, transmission energy efficiency, processing energy efficiency, Hardware requirements, latency and throughput. We have taken the comparisons discussed in this paper as a standard to compare our results with the results already published in literature.

In literature both fully parallel and semi parallel implementation of LDPC decoders can be found. In [9] [10] [11] [12] [13] we have fully parallel FPGA implementations but none of these designs supports streaming input support because of which the throughput completely depends upon the latency. The lesser the latency the higher will be the throughput. Other factors that contribute in throughput calculations is the clock speed. Number of iterations and the block length.

In [9] Bit-Serial approximated decoder is implemented on FPGA. This is a fully parallel architecture in which bits are passed between the nodes in serial manner. This is done to avoid routing complexity. Also this technique can improve the decoding performance by increasing the quantization bits of the messages.
In [10] implementation results for a (1024, 512) are discussed for fully-parallel LDPC decoder. The author claims a throughput of 650 Mbps for Altera Stratix IV device at 61 MHz which is increased for Xilinx up to 709 Mbps. This design has maximum number of iterations equal to 32 which has increased the latency many fold, therefore throughput was reduced.

In [11] it is claimed to have a throughput of 6 Gbps with a block length of 1200 and 10 number of iterations at 100 MHz. The design was very resource expensive and 45% of hardware resources were consumed for the implementation of single iteration of LDPC decoder.

The designs described in [12] and [13] are from the same author who enhanced his results for the journal publications. Therefore we will only discuss the better implementation. At maximum he has achieved 16.2 Gbps at 188 MHz using simplified message passing algorithm which is also commonly known as bit-flipping algorithm. He did not share the hardware resource utilizations but his design takes 3.8 cycles on average to correct a codeword.

There are some semi parallel architectures like [14] and some FPGA based industrial designs offered by companies like Blue rum, Unicore, Iprium and Trellis ware are worth mentioning here. But we are not discussing the details of their design since they are using semi parallel architecture to target specific standards and we are targeting fully parallel architecture of LDPC.

In [15] discussed the fixed point implementation and bit clipping techniques which we applied in our design to optimize our design with minimal hardware resources. This paper showed that in most scenarios four bits are enough to get a performance close to ideal for wide span of SNR ratios.

Following is the table showing hardware implementation of some fully parallel LDPC decoders.
<table>
<thead>
<tr>
<th>ref</th>
<th>Code Length</th>
<th>Clk MHz</th>
<th>Device</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Throughput</th>
<th>LUTs stated</th>
<th>Registers stated</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9]</td>
<td>480</td>
<td>61</td>
<td>Altera, Stratix</td>
<td>Min-sum with simplified check update function</td>
<td>15</td>
<td>481 M*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[10]</td>
<td>1024</td>
<td>212</td>
<td>Xilinx, Vertex 4</td>
<td>Stochastic</td>
<td>NA</td>
<td>353 M*</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: table showing various parameters and characteristics of hardware implementations available in literature

These implementation will be used later for comparison with our hardware implementation.
Chapter 3 Low Density Parity Check Decoder

LDPC codes are block codes and as the name suggests it contains parity check matrices. LDPC has very sparse parity check matrices which will be called H matrix in this literate. Sparseness of H matrices are essential for iterative decoding codes to reduce the complexity and to make it linear only with the length of the block code. LDPC codes can be best represent graphically using tanner graphs. There are two types of nodes in the tanner graph. One is called bit nodes and the other is called the check nodes. The bit nodes present in the LDPC is equal to the block length of the codeword whereas the check nodes are equal to the no. of parity equations of LDPC. In other words there as many bit nodes as the number of columns in the parity check matrix. And there are as many check nodes as the number of rows in the parity check matrix. Bit nodes and check nodes are joined by an edge if and only if that bit is present in the respective parity check equation of that check node.

![Tanner graph representing bit nodes and check nodes](image)

If an H matrix has same number of 1s in all column called $w_c$ and another same number of 1s in all rows called $w_r$. Then such a parity check matrix is called regular parity check matrix.
Before moving to the next section we will discuss some important properties of error correcting codes.

Firstly, it should be noted that the error correcting codes are divided into two parts, block codes and convolutional codes.

For block codes encoding can be described as follows; the information symbols to be sent are divided into blocks of $k$ symbols, then these $k$ symbols are encoded to $N$ symbols where the code rate (or the ratio of the number of information symbols to total number of symbols) is $k/N$. The decoding process is the reverse of this case, where the received information is chopped into blocks of $N$ symbols and decoded to $k$ information symbols.

The number of bit positions in which two codewords differ is called the Hamming distance between them. The minimum Hamming distance or just minimum distance of a code, $d_{\min}$, is defined as the smallest Hamming distance between any pair of codewords in the code. In general, a code with minimum distance $d_{\min}$ can definitely detect $t$ errors whenever

$$t < d_{\min}$$  \hspace{1cm} (1)

The smaller the code rate $r$, the smaller the subset of $2^N$ binary vectors that are codewords and so the better the minimum distance that can be achieved by a code with length $N$. The importance of the code’s minimum distance in determining its performance is reflected in the description of error correction codes by the three parameters $N$, $K$, $d_{\min}$.

The job of the decoder is to detect if the received codeword has been corrupted by the channel or not and if possible it should correct that codeword. There are two types of Decoders for LDPC which are as following:
3.1 Hard Decision Decoders
To decode LDPC codewords using hard decision decoders first we check the syndrome. If the code word is incorrect. We try to find out the correct codeword. To find the correct code work each parity equation recommends a value for all the bits present in that parity check equation. These recommended binary values are called check messages.

Figure representing an update for bit message

Now each bit has a number of check messages these check messages has been passed to bit nodes. These bit nodes collects all the check messages for their respective bits and selects the majority of the value that is recommended by the check nodes. for example if a bit has 5 check messages and 2 of them is recommending 1 and other 3 check messages are recommending zero than the bit node select 0 as the bit update bit message. Now these bit messages are passed to the check nodes again, and this process will be repeated until the codeword has been corrected or the process has been repeated for the maximum number of iterations. This algorithm is also called bit flipping algorithm. It is based on the assumption that if a bit is flipped in a large number of parity check equations no other incorrect bit each of them will calculate a correct value for that bit.

3.2 Soft decision decoders
There are different variations of soft decision decoding algorithms for LDPC. Basically soft decision decoding algorithms are derived from the Bit Flipping decoding algorithm but in these algorithms the decisions are based on messages that are probabilities of the bits to be 1 or 0 instead of using binary bit values, hence the algorithm is known as soft decision decoding. In this thesis we are going to discuss the two soft decision decoding algorithms i.e. Sum product (SP) decoding algorithm and Min sum (MS) decoding Algorithm.
3.2.1 Sum Product decoding Algorithm

The SP decoding is basic derived from the Bit Flipping decoding algorithm and is considered to be the ideal decoding algorithm for LDPC. All the other variations of soft decision algorithms are derived from the SP decoding algorithm because this algorithm is very complex to implement for real time communication system. So other derived algorithms from SP algorithms are basically the simplification of this algorithm to realize the hardware implementation of LDPC decoders on hardware at the cost of some decoding efficiency.

Like bit flipping algorithm there are two parts of sum product algorithm the check message update and bit messages update.

The check messages $E_{ji}$ from $j^{th}$ check node to the $i^{th}$ bit node is the opinion of $j^{th}$ check node on the bit $c_i$ to be 1. This means $E_{ji}$ gives the probability that $c_i = 1$ will satisfy the $j^{th}$ parity check equation. Here it is also important to mention that $E_{ji}$ doesn’t not exist if $i^{th}$ bit is not included in the $j^{th}$ parity check equation. The probability that a parity-check equation is satisfied if $c_i = 1$ is the probability that an odd number of the bits in that parity-check equation are 1s:

$$P_{ji}^{ext} = \frac{1}{2} - \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2 P_{ji})$$  \hspace{1cm} (2)

Log likelihood ratios are used to represent the metrics for a binary variable by a single value (1.2):

$$L(x) = \log \frac{p(x = 0)}{p(x = 1)}$$  \hspace{1cm} (3)

The sign of $L(x)$ provides a hard decision on $x$ and the magnitude $|L(x)|$ is the reliability of this decision.

Log likelihood ratio for the extrinsic information from the $j^{th}$ check node to $i^{th}$ bit node can be expressed as:
\[ E_{j,i} = L(p_{\text{exit}}^{ji}) = \log \frac{p(1 - p_{\text{exit}}^{ji})}{p_{\text{exit}}^{ji}} \]  

Now substituting equation 2 in equation 4 we have:

\[ E_{j,i} = \log \frac{\frac{1}{2} + \frac{1}{2} \prod_{i' \in B_{j,i}, i \neq i}(1 - 2P_{j,i})}{\frac{1}{2} - \frac{1}{2} \prod_{i' \in B_{j,i}, i \neq i}(1 - 2P_{j,i})} \]  

This equation can be simplified to

\[ E_{j,i} = \log \frac{1 + \prod_{i' \in B_{j,i}, i \neq i} \tanh(M_{j,i'}/2)}{1 - \prod_{i' \in B_{j,i}, i \neq i} \tanh(M_{j,i'}/2)} \]  

Where

\[ M_{j,i} \equiv L(P_{j,i'}) = \log \frac{1 - p_{j,i'}}{p_{j,i'}} \]

The value of each \( i \)th bit message can be calculated by the summation of all the check messages connected to that bit node. And the input LLR \( R_i \). We call the set of all check messages connected to the \( i \)th bit as \( A_i \).

\[ M_i = \sum_{j' \in A_i} E_{j',i} + R_i \]

The equation 8 is called the bit node updated equation for SP decoding Algorithm whereas equation 15 is the check node update equation for SP decoding Algorithm.

### 3.2.2 Min Sum Decoding Algorithm

Min sum decoding Algorithm is the simplified from of sum product decoding algorithm. In min sum decoding algorithm there is no change in the bit node update algorithm. But the check node equation can be drive from the equation 15.

Equation 15 can be re write as:
\[ E_{j,i} = 2 \tanh^{-1} \prod_{i' \in B_j, i \neq i} \tanh \left( \frac{M_{j,i'}}{2} \right) \]  

(9)

The term \( M_{j,i'} \) can be factorized as follows:

\[ M_{j,i'} = \alpha_{j,i'} \beta_{j,i'} \]  

(10)

Where,

\[ \alpha_{j,i'} = \text{sign} \, M_{j,i'} \]  
\[ \beta_{j,i'} = |M_{j,i'}| \]

Using equation 10 we can write,

\[ \prod_{i' \in B_j, i \neq i} \tanh \left( \frac{M_{j,i'}}{2} \right) = \prod_{i' \in B_j, i \neq i} \alpha_{j,i'} \prod_{i' \in B_j, i \neq i} \beta_{j,i'} \]  

(11)

Using equation 11 we can rewrite equation 9 as,

\[ E_{j,i} = \prod_{i' \in B_j, i \neq i} \alpha_{j,i'} \, 2 \tanh^{-1} \prod_{i' \in B_j, i \neq i} \tanh \left( \frac{\beta_{j,i'}}{2} \right) \]

(12)

This equation can be simplified as,

\[ E_{j,i} = \prod_{i' \in B_j, i \neq i} \text{sign} \, M_{j,i'} \, \min_{i' \in B_j, i \neq i} |M_{j,i'}| \]  

(13)

3.3 LDPC decoding Flow

The flow of LDPC decoding algorithm will remain same for all different types and variations of decoding algorithm. The flow chart for almost all the LDPC decoding algorithms is presented in Figure 3.
3.4 Applications of LDPCs

Main application areas of error detecting/correcting codes can be given as.

3.4.1 Wireless and Mobile Communications

Error correcting techniques are widely used in mobile communication systems to cope with the perturbations caused by interference, noise, multi path fading, shadowing, propagation loss, etc. in the wireless channel. In GSM; Cyclic Redundancy Codes (CRC) are used for error detection, block and convolutional codes are applied for error correction. In CDMA2000; convolutional codes and turbo codes are used for error correction. In 3G; both convolutional and turbo codes are supported for error correction. Low-Density Parity-Check (LDPC) codes are the standard error correcting codes for many wireless communication protocols such as WiMAX (802.16e) and WLAN (802.11).
3.4.2 Deep Space Communications

Since space and the atmosphere is the channel for these communication systems, the space radiation is the most effective disturbance to the signal which can be modeled as Additive White Gaussian Noise Channel (AWGN). Although the noise sources are very limited, communication from outer space would be impossible because of the high propagation loss due to the distance.

Thus, error correction is the only sensible way to communicate with signal powers as low as in space communication case. First of the codes that is used for deep space communications is (32, 6, 16) Reed-Muller in the Mariner spacecraft. After that, mostly convolutional codes and Reed-Solomon (RS) codes are used for space communications, with improvements in rate, gain, or time performance.

3.4.3 Satellite Communications

For digital satellite TV, RS codes had been used for a long time, however currently replaced by modern codes such as LDPC and Turbo codes.

3.4.4 Military Communications

In military communications, besides the natural interference and noise, the communication system also needs to deal with the intentional enemy interference; therefore, the need for error correcting codes is obvious.

3.4.5 Data Storage

RS codes is widely used for error correction in storage systems such as CDs, DVDs and hard discs. Single Error Correcting Double Error Detecting (SECDED) codes, which correct single errors and detect double errors, are widely used in many data storage units with RS codes or Hamming codes.
Chapter 4 Hardware Implementation

In this chapter we will discuss the hardware implementation in details. In 4.1 we will discuss the top level module of LDPC. Then we will discuss the sub blocks i.e. quantizer, check nodes, bit nodes, check message translator and bit message translator, syndrome and Codeword extractor one by one.

4.1 LDPC Top

LDPC top is the wrapper module of all the sub blocks of LDPC. It stitches all the sub blocks according to the parameters at the compile time. In Figure 4 block diagram of LDPC top is given.

4.1.1 Architecture of LDPC top level module

The top level module accepts the BN bits wide binary Codeword, and outputs a BN bits wide corrected Codeword.

The binary code word is first changed into the soft bit message. This is done in the Message quantizer block. In message quantizer apriori probabilities are assigned to the bits depending upon their initial binary values if a bit is 1 it is assigned a negative apriori value and vice versa. The output of the message quantizer are initial or raw bit messages. These bit messages which are BN*MW wide are then feed into bit message translator.

It is the responsibility of the bit message translator to arrange all the expected input bit messages for all the check nodes in the ascending order of check nodes. For example if check nodes at maximum expects four bit messages then bit message translator puts respective four bit messages for each check node on a bus in such a way that first four bit messages belongs to first check node. That is first 4*MW wide part of the bus is feed into the first check node. Similarly the second 4*MW wide part of the bus is feed into the second check node. And it continues for all the check nodes. Hence the bus between bit message translator and check nodes top is CNW x CN wide. Where CNW is defined as the check node width which
is MW * Max bits per check node. If a check node expects less than maximum bits than the MSBs of its respective bits will remain unused. The output of the bit message translator is feed into the check nodes top module.

The check node top module is the wrapper module for check node modules. Check node module receives its input bit messages and outputs the corresponding check messages for each input bit message. These check messages are put on a bus by check node top in the similar order of bit messages. Hence the input and output bus of the check nodes top is CNW * MW wide. The output check messages by check nodes top are feed into the check message translator.

The check message translator receives the CNW*MW wide bit messages and arranges all the check messages in such a way all check messages for a corresponding bit node is put together. For example if a bit node at maximum expects 8 bit messages, then first eight bit messages placed on the output bus will belong to the first bit node and similarly second eight bit messages will belong to second bit node. This will go on for all the BN bit nodes in the ascending order of bit nodes. The output of check message translator are feed into bit nodes top module.
Figure 4: Block diagram of LDPC top level module
The bit nodes top module accepts the arranged check messages in the ascending order of bit nodes such that all the check messages belonged to a single bit node are put together. This module outputs the updated bit messages and hence completes the calculation of the given iteration. Bit nodes top is the wrapper module for all the bit node modules. Each bit node module accepts check messages for its particular bit node and outputs a single bit message for that bit node. All the output bit messages from the output bit nodes are out as the output of bit nodes top. Syndrome will be calculated on the output bit messages of bit nodes. Also the output of the bit nodes will be feed into the next iteration, otherwise if it is the output of the last iteration it will be feed into the messages extractor.

Message extractor module accepts the bit messages of the last iteration and outputs the binary Codeword which is the corrected Codeword and hence the output of the LDPC top module.

4.1.2 Latency of LDPC top level module

The latency of LDPC top level module is the sum of latencies of all the sub modules. In the form of the formula the latency can be calculated as following.

\[
L_{LDPC} = L_{MQ} + (L_{BMT} + L_{CNT} + L_{CMT} + L_{BNT}) \times \text{No. of iterations} + L_{ME}
\] (14)

In our case message quantizer, bit message translator, check message translator and message extractor modules do not have any latencies. They are just a wrapper to various connection or simple combinational logics. Therefore the equation can be reduced as following.

\[
L_{LDPC} = (L_{CNT} + L_{BNT}) \times \text{No. of iterations}
\]

Since the latency of check nodes top is 3 cycles and latency of bit nodes top is 3 cycle, and assuming 4 decoding iterations. The total latency of the module is as following.

\[
L_{LDPC} = (3 + 3) \times 4
\]
4.1.3 Parameterization of top level module for LDPC

LDPC top level module has been parametrized for the following parameters.

1. Message width
2. Number of bit nodes
3. Number of check nodes
4. Number of iterations
5. Negative and positive apriori probabilities

And LDPC top level module is also parameterized for any decoding matrix using following parameters.

6. Maximum bit messages per check nodes
7. Maximum check messages per bit
8. Number of bit messages per check node
9. Number of check messages per bit node

4.2 Message Quantizer

This is the first module in the data path of LDPC module. The input of this module is BN bits wide input Codeword. The output of this module is the initial bit messages that are BN x MW bits wide bus. This module assigns apriori log likelihood values to the bits depending upon their values. If the bit value is 1 it will be assigned negative apriori log likelihood values. If the bit value is 0 it will be assigned positive apriori log likelihood values. The latency of this module is 0. The latency of 0 cycles is used to optimize the resources. The latency of this module can be increased to any number without effecting the pipelining of the LDPC top level architecture. This module is parameterized for any length of Codeword.
It is also parametrized for negative and positive apriori values and message widths of the output bit messages.

4.3 Check Nodes

Check nodes is one the main functional unit of LDPC hardware implementation. The block diagram of check nodes is given in Figure 5.

![Check Nodes Top](image)

*Figure 5: Block diagram of Top level of Check nodes*

4.3.1 Architecture of Check Nodes

Check nodes top is the top level module for check nodes. This module accepts the arranged bits messages and outputs the check messages for all the corresponding bit messages. The input bit message is CNW wide for each check node. CNW is defined as the product of maximum number of bit message
per check node and message width of the check messages. Therefore the size of the input and output
bus of check nodes top module is CNW * CN.

Check nodes top is the wrapper of check node modules. There are CN number of check nodes in the
check nodes top module. Each check node module works independently with other check nodes, and all
the check nodes are working in parallel. Each check node module can accept any number of check
messages which can be controlled by the corresponding parameter. The input bus of check nodes is
CNW bits wide. In which only BITS * MW bits are active. Where BITS are defined as the number of bit
messages for the check node. The output of the check node module is also CNW bits wide. In which only
BITS * MW bits are active.

Check node module is the wrapper of check node element modules. Each check node element has the
same number of inputs as its wrapper but each check node element calculates only one check message.
Therefore check nodes element can be considered as the basic building block for each check nodes
module. Each check node module has as many check nodes element modules as the number of input bit
messages to that check node. Each check node element calculates a check message for a particular bit.

The functionality of the check nodes element depends upon the variation of algorithm which is to be
implemented. Since we have implemented Min-Sum algorithm therefore our implementation is
according to the min sum algorithm. We have to implement the equation 15.

\[
E_{m,n} = \prod_{n' \in N(m)/n} sgn(F_{n'}, m) \cdot \min_{n' \in N(m)/n} |F_{n'}, m| 
\]  

(15)

The equation 15 has two parts of the equation. The first part of the equation decides sign of the output
check message. Whereas the second part of the equation decides the magnitude of the equation. This
has been implemented in hardware with the help of two different combinational logic. The first part has
been implemented simply using XOR gate. The second part that is the magnitude of the equation is calculated by taking the absolute magnitude of all the bit messages. The absolute magnitude can be calculated by subtracting the negative value from the maximum possible and adding 1 in the result. Once all the values are converted to their absolute values then minimum can be calculated using a mux.

4.3.2 Latency

One cycle is required to calculate the absolute values for the incoming bit messages.

Another cycle is required to calculate the minimum values. And calculating the sign of the output by using XOR gates for all the MSBs of incoming bit messages.

And still another clock cycle is required to unify the magnitude and sign into a single message.

Thus a total of 3 clock cycles are required to complete the computation of check nodes.

4.3.3 Parameterization for top level module for check nodes

Check nodes top level module has been parametrized for the following parameters.

1. Message width
2. Number of check nodes

And check nodes top level module is also parameterized for any decoding matrix using following parameters.

3. Maximum bit messages per check nodes
4. Number of bit messages per check nodes

4.4 Bit nodes

Bit nodes are relatively simple to implement. But somewhat hard to parameterize. Since we need to implement LDPC for research purpose it will be beneficial for us to make a parameterizable hardware. In
4.4.1 Bit nodes Architecture

The function of bit nodes is to calculate the bit messages from check messages. Each check node sends a message to each of their connected bit nodes (see tanner graph). The message from the check node to the bit node declares the likelihood of that bit to be 1 (to satisfy the parity check equation) as determined by that check node. Each bit node collects all its respective check messages and calculates the likelihood of its respective bit of being 1. It actually sums all the check messages and previous bit message value [16]. The value of the bits can be determined by the sign of bit messages, positive value suggests that the bit value is 0 and vice versa. Bit nodes are implemented in hardware using two modules.

1. Bit nodes top

   The bit nodes top is the wrapper of all the bit nodes. It is parameterized for any length of Codeword and message width. It is also parameterized for any given matrix using two parameters:

   a. Maximum check messages per bit

   b. Number of check messages per bit

   The second parameter is the sum of all the 1s in each column in the given H matrix. The second parameter is the maximum value of the list of 2nd parameter. The second parameter defines how many valid check messages each bit node has, whereas the first parameter defines the width of the bit node input interface for check messages. The width of input check message in every bit node will simply be the product of message width and the maximum number of the input check messages.

   The block diagram of Bit node module is given below:
In order to feed correct inputs to all the bit nodes, we pre-arranged the check messages. This is done in check message translator, hence it is the responsibility of check message translator to arrange check messages in such a way that all the check messages for a particular bit node is arranged together in ascending order of bit nodes. Since we can have variable number of check messages for different bit nodes therefore we need to have a list of number of check messages each bit node has. We will call this parameter as NO_CHK_MSGS_PER_BIT. To populate this list we will manually enter the values of check nodes associated with each bit. We will keep the width of all the populated values constant and call it NO_CHK_MSGS_PER_BIT_WIDTH and it will be determined by the highest value to be entered in the list.

Let’s call this new parameter as MAX_CHK_NODES_PER_BIT. So NO_CHK_MSGS_PER_BIT_WIDTH is \( \log_2 (\text{MAX_CHK_NODES_PER_BIT}+1) \). Since \( \log_2 (8) \) is 3 and the value of 8 can be expressed in minimum 4 bits in binary, therefore we need to add an additional 1 to cater this corner case for the values that are integer power of base 2. Now the width of the check messages bus can be calculated as
CODE_WORD_LENGTH*MAX_CHK_NODES_PER BIT*MESG_WIDTH.

For example: if the CODE_WORD_LENGTH is 4 and MESG_WIDTH = 6 and NO_CHK_MSGS_PER_BIT = {3’d3,3’d5,3’d3,3,d3} then MAX_CHK_NODES_PER_BIT = 5 and NO_CHK_MSGS_PER_BIT_WIDTH = 3. And the check messages bus width will 4*5*6 = 120. So we have assigned the same bus width for check messages of all Codeword bits i.e. 5*3 bits but some of the bits of the check messages bus will remain unused since all the check nodes do not have max no. of check messages.

2. Bit node

The bit node module is the leaf module of bit nodes block. It takes check messages and the previous value of bit message of respective bit node. The module is parameterized for any number of check messages and for any message width. This module perform addition of the input check messages and previous value of bit message. The result of the addition is truncated to the applied message width following the rules of fixed point arithmetic.

4.4.3 Bit nodes designing, Parameterization and reducing the hardware cost

Bit nodes are designed in such a manner to have a fully parallel pipelined implementation of LDPC with a design approach to achieve the following:

1. Reduce the latency
2. Reduce the hardware cost
3. And increase the throughput

We have used a unique method here to optimize the bit messages hardware and parameterize it at the same moment. We took the advantage of a property of synthesis tools i.e. those constant signals that do not have any contributions to the outputs are removed in synthesis. Therefore we used a 32*MESG_WIDTH wide register, say check_word register and assigned the parameterized check messages bits to the LSBs of check_word register. Then individually sum all the bits of the check_word in a single
equation. Now since the MSBs are never assigned any value therefore in synthesis unused MSBs will be removed. Also since check messages are parameterized by the parameter NO_CHK_MSGS_PER_BIT, therefore the whole equation is now parameterized. Also the output of bit message will be calculated in a single cycle. Then we need an extra 2 cycle to round the sum of the bit message and truncate it back to the width MESG_WIDTH. This code will be synthesized with the most optimized resources and at the same it is parametrized for any numbers of check messages up to 32 (the size of check_word register). The size of check_word register is actually good enough for almost all the standard LDPC decoders. If we need more elements we can simply increase the size of check_word register and append the individual addition of increased bits in the equation. The sum of check messages is signed sum. Hence this design is achieving both the required parameterization as well as hardware resource optimization.

4.4.4 Parameters for bit nodes

1. Codeword length
2. Message width

This module is also parameterized for any LDPC decoding matrix using following parameters.

3. Number of check messages per bit
   It is an array/list of number of check messages each bit has. For example if the CODE_WORD_LENGTH is 10 and all 10 bits have 3 check messages it should be {10(2’d3)}.
4. Maximum check nodes per bit
   This is the highest value listed in array/list "NO_CHK_MSGS_PER_BIT". In the above example all the bits has 3 check messages so it should be 3.
5. Number of check messages per bit width
This is log2 (MAX_CHK_NODES_PER_BIT) and defines the width of each element in the array/list "NO_CHK_MSGS_PER_BIT".

In the example given in parameter description of NO_CHK_MSGS_PER_BIT
MAX_CHK_NODES_PER_BIT = 3 therefore NO_CHK_MSGS_PER_BIT_WIDTH should be 2.

4.5 Check Message translator

Check message translator accepts check messages from the check nodes top and rearrange them so that they can be accepted by bit nodes top. Check message translator is just a connection box and its job is to route the check messages to appropriate bit nodes. There is no logic involved in the check messages translator. Since the routing logic highly depends up on the various parameters and the LDPC parity check matrix therefore this block is not parametrized. Instead this block is generated using a script. Because using a script to generate the Verilog file for this block is a lot more convenient a flexible. Since this block does not contain any logic therefore it does not utilizes any hardware resource except for routing. Also the latency of this block is zero clock cycles since it only contains wires.

4.6 Bit Message Translator

Bit message translator accepts bit messages from the bit nodes top of previous iteration or message quantizer block for the first iteration and rearrange them so that they can be accepted by check nodes top. Bit message translator is just a connection box and its job is to route the bit messages to appropriate check nodes. There is no logic involved in the bit message translator. Since the routing logic highly depends up on the various parameters and the LDPC parity check matrix therefore this block is not parametrized. Instead this block is generated using a script. Because using a script to generate the Verilog file for this block is a lot more convenient a flexible. Since this block does not contain any logic therefore it does not utilizes any hardware resource except for routing. Also the latency of this block is zero clock cycles since it only contains wires.
4.7 Syndrome

Syndrome is the block that evaluates if the code word has been corrected by the decoder or not. Since all the codewords have the fixed decoding latency and all the codewords pass through the complete pipeline of all decoding iterations. Therefore early stop support is not required in the decoder because of this block is placed at the last decoding iteration just to confirm if the codeword is correct or not.

4.7.1 Architecture

The top level block of syndrome is called syndrome top and which is the wrapper to all the leaf modules of syndrome. The syndrome top module takes the quantized bit messages and passes those bit messages to the syndrome instances. There are as many syndrome module instances as the number of parity check equations. Each syndrome module is emulates a parity check equation and XORs all the input messages to check if the equation is satisfied or not. If all the parity check equations are satisfied by the codeword then all syndrome top modules should flag the error as zero. If any or more equations are not satisfied than syndrome top should set the error flag as 1.

4.7.2 Latency of Syndrome Block

The latency of syndrome block is a combinational block and its latency is 0 cycles.

4.7.3 Parameterization of Syndromes

Syndrome top level module has been parametrized for the following parameters.

1. Number of check nodes

And syndrome nodes top level module is also parameterized for any decoding matrix using following parameters.

2. Maximum bits per check nodes

3. Number of bits per check nodes
4.7.4 Resource optimization of Syndrome

The syndrome block is a combinational block and does not utilizes much of resources. In syndrome module all the bits included in a particular check node are XORed and in syndrome top level module all the output status of syndrome modules is ORed to generate the error flag.
Chapter 5 Results, Analysis and Commentary

The performance of error correction hardware is measured from two aspects. Which are as following.

1. Error correction Performance
2. Hardware Utilization and Throughput

Although this thesis is focused on hardware implementation of LDPC decoder but hardware implementation is incomplete without verification of the Error correction performance. Therefore we have analyzed our design for both the aspects and the details are given in the subsections 5.1 and 5.2.

5.1 Error Correction Performance

For the implemented LDPC hardware error correction performance is evaluated using MATLAB based simulations. The steps to perform performance analysis are as follows:

1. A number of samples of information messages are generated randomly and then encoded using our LDPC codeword generator model.
2. Than random Gaussian noise is added to the generated codeword to emulate channel noise.
3. Then our LDPC model is used to decode these messages after being transmitted over a noisy channel.
4. The performance was evaluated using bit error rate (BER) vs Signal to noise ratio (SNR) graph as a parameter of merit.
5. This setup is repeated for different values of SNR.
6. We have used 500 message samples for each SNR value. For Sum Product decoding algorithm we have used 30 iterations to calculate the BER to get the ideal decoding performance from our decoders. Whereas for Min-Sum algorithm we have used 4 iteration and 4 bits wide messages to evaluate the performance of the decoder we have implemented on FPGA.
The results are shown below:

<table>
<thead>
<tr>
<th>S. No.</th>
<th>SNR</th>
<th>BER for 128 bits codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Db</td>
</tr>
<tr>
<td>1</td>
<td>5.2</td>
<td>0.302953846</td>
</tr>
<tr>
<td>2</td>
<td>5.9</td>
<td>0.255076923</td>
</tr>
<tr>
<td>3</td>
<td>7.8</td>
<td>0.167692308</td>
</tr>
<tr>
<td>4</td>
<td>8.8</td>
<td>0.133292308</td>
</tr>
<tr>
<td>5</td>
<td>9.5</td>
<td>0.110940625</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0.100430769</td>
</tr>
</tbody>
</table>

*Table 2: BER vs SNR (db) table for 128 bits Codeword decoder LDPC*

The above table can be represented in the graph as below.

*Figure 7: BER vs SNR (db) graph for 128 bits Codeword decoder LDPC*

In the above graph we can see the comparison of Min-sum algorithm implemented on the FPGA vs the Sum Product algorithm which is considered to be the ideal decoding algorithm for LDPC decoders. We can see that our decoder is performing well for a wide range of SNR values. Also by analyzing the above graph we can state that performance of MS decoder implemented on FPGA is comparable to SP product algorithm which is considered to be ideal decoding performance.
5.2 FPGA Implementation

We have analyzed our design for both Xilinx and Intel FPGAs. For the performance evaluation we have selected the latest state of the art FPGAs. And for the sake of comparison between two different vendors, we have make sure that both the FPGA should be based on same Fabrication technology. Therefore we have selected 20nm FPGAs for both Xilinx and Intel. We have selected Arria10 series FPGA for Intel and Vertex 7 series FPGA for Xilinx. The details of the implementation results are given in the sections 5.2.1 and 5.2.2

5.2.1 Intel FPGA Synthesis and Implementation

We have selected Arria10 part number 10AS066N3F40E2SGE2 FPGA for evaluation purposes. This is a mid-range FPGA of mid-range series of Intel FPGAs i.e. Arria10 series. This FPGA was available with us and we successfully verified our design on this FPGA. The synthesis results are acquired using Quartus Prime 16.0. The summary of resource utilization report of the FPGA is shown in the table below.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Variation</th>
<th>ALMS</th>
<th>REGS</th>
<th>DSP</th>
<th>F(_{\text{max}}) MHz</th>
<th>Throughput Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td></td>
<td>251680</td>
<td>1006720</td>
<td>2274</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>128-bit</td>
<td>33,016</td>
<td>16519</td>
<td>0</td>
<td>218.91</td>
<td>28.02</td>
</tr>
<tr>
<td>2</td>
<td>256-bit</td>
<td>66,026</td>
<td>31359</td>
<td>0</td>
<td>189.11</td>
<td>48.412</td>
</tr>
</tbody>
</table>

Table 3: FPGA resource summary for LDPC implementation for Arria 10 device

The above table shows that our design is synthesizable for very fast clock speeds and higher throughput within reasonable resources.

5.2.2 Xilinx FPGA Synthesis and Implementation

We have selected Vertex UltraScale part number xcvu190fglb2104-3-e FPGA for evaluation purposes. These synthesis results are acquired using Vivado design suite 2016.1. This device is a 20nm. In all these
Number of iterations are set to 4. The summary of resource utilization report of the FPGA is shown in the table below.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Variation</th>
<th>LUTs</th>
<th>Flip-Flops</th>
<th>DSP</th>
<th>$F_{\text{max}}$ MHz</th>
<th>Throughput Gbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td>1074240</td>
<td>2148480</td>
<td>1800</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>128-bit</td>
<td>43067</td>
<td>16068</td>
<td>448</td>
<td>275.48</td>
<td>35.26</td>
</tr>
<tr>
<td>2</td>
<td>256-bit</td>
<td>86809</td>
<td>31995</td>
<td>896</td>
<td>256.54</td>
<td>65.67</td>
</tr>
</tbody>
</table>

*Table 4: FPGA resource summary for LDPC implementation for Vertex Ultra scale device*

The above table shows that our design is synthesizable for very fast clock speeds and higher throughput within reasonable resources. Moreover we can see that Vivado design suite utilized some DSP48 resources in this design because of which we got even higher clock speeds as compared to ALTERA.

Since there are a couple of adders in the Xilinx DSP48 blocks therefore it is utilizing these DSP blocks to map the addition operation of bit nodes.

5.3 Comparison with other implementations.

For the sake of comparison we are reproducing the Table 1 with our design characteristics and parameters.

<table>
<thead>
<tr>
<th>ref</th>
<th>Code Length</th>
<th>Clk MHz</th>
<th>Device</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Throughput Gbps</th>
<th>LUTs stated</th>
<th>Registers stated</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9]</td>
<td>480</td>
<td>61</td>
<td>Altera, Stratix</td>
<td>Min-sum</td>
<td>15</td>
<td>481 M*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[10]</td>
<td>1024</td>
<td>212</td>
<td>Xilinx, Vertex 4</td>
<td>Stochastic</td>
<td>NA</td>
<td>353 M*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[12]</td>
<td>648</td>
<td>188</td>
<td>Xilinx, Vertex 5</td>
<td>Message Passing</td>
<td>3.8</td>
<td>16.2 G</td>
<td>14239</td>
<td>5963</td>
</tr>
<tr>
<td>128 Altera</td>
<td>128</td>
<td>218.91</td>
<td>Altera, Arria10</td>
<td>Min-sum</td>
<td>4</td>
<td>28.02</td>
<td>33,016</td>
<td>16519</td>
</tr>
<tr>
<td>256 Altera</td>
<td>256</td>
<td>189.11</td>
<td>Altera, Arria10</td>
<td>Min-sum</td>
<td>4</td>
<td>48.412</td>
<td>66,026</td>
<td>31359</td>
</tr>
<tr>
<td>128 Xilinx</td>
<td>128</td>
<td>275.48</td>
<td>Xilinx, Vertex US</td>
<td>Min-sum</td>
<td>4</td>
<td>35.26</td>
<td>43067</td>
<td>16068</td>
</tr>
<tr>
<td>256 Xilinx</td>
<td>256</td>
<td>256.54</td>
<td>Xilinx, Vertex US</td>
<td>Min-sum</td>
<td>4</td>
<td>65.67</td>
<td>86809</td>
<td>31995</td>
</tr>
</tbody>
</table>

*Table 5: Table showing comparison between different fully parallel hardware implementations of LDPC*
In the Table 5 we can see that all our hardware implementations outperformed all previous implementations in terms of throughput comparison. This is highlighted in the Figure 8.

![Figure 8: Throughput comparison between implementations discussed above.](image1)

This throughput cannot be achievable without efficient hardware architecture designing and pipelining of the decoder. This can be evident that our design is synthesizable for very high clock frequencies. This comparison can be seen in the Figure 9.

![Figure 9: Comparison between timing closures of different hardware implementations of LDPC discussed above.](image2)
It can be seen in the Table 5 that our hardware utilizes a lot more hardware resources in comparison with previous implementations. This cost is unavoidable to get higher throughput and error correction performance. But still with the evaluation of FPGA technology we can afford this luxury as we have much more logic available in the FPGAs available today. The resources utilized by the Arria10 FPGA for 256 bits codeword decoding take only 26% of the overall logic available in the FPGA. Since Arria 10 is a midrange FPGA device and this suggests that the FPGAs available today are capable of handling LDPC decoder with higher block lengths and streaming input capabilities. The resource comparison is highlighted in the Figure 10.

![Hardware Resource Utilization](image)

*Figure 10: Resource Utilization comparison between LDPC decoder FPGA implementations discussed above*

5.4 Effects of Parameters on Resource Utilization VS Decoding Performance.

There are various parameters that effects the performance and hardware utilization of the LDPC decoder. These parameters also dictates the error correction performance of the decoder. Therefore the tradeoff between the decoding performance and the hardware utilization is Important. In this
sections we will discuss the importance of different parameters and their effects on the both the decoding performance and hardware utilizations.

5.4.1 Architecture of LDPC Decoder

Architecture of LDPC decoder is the most important aspect with respect to hardware implementation of LDPC. It is the architecture that defines the throughput of the LDPC design. It also plays an important part in the hardware resource utilizations of the FPGA. In this implementation we have used fully parallel architecture because our goal was to achieve the maximum throughput with our implementation.

5.4.1 Number of Iterations

Number of iterations is also one of the most important parameter when it comes to hardware implementation. If all the iterations are pipelined in the hardware it means more hardware resources are required but if same hardware is used to evaluate all the iterations then it means higher the number of iterations lower will the decoding throughput. In this implementation we have used 4 iterations as we have observed that MS algorithm converges faster than SP algorithm therefore 4 iterations are a good tradeoff between FPGA resources and decoding performance.

5.4.2 Message Width

Message width is also one of the important parameters in LDPC hardware implementation. Higher the message width higher will be the resource utilization and higher will be the decoding accuracy. The message width parameter and fixed point implementation go hand in hand. In [15] it is shown how to use the fixed point implementation intelligently so that we can achieve almost similar performance with four bits wide messages as with higher width of messages.
5.4.3 Pipelining

Pipelining is one of the most important parameter with respect to FPGA implementation. In this architecture it is shown we can reduce the hardware resource with by compromising the message width and number of iterations so that we can implement different hardware for different iterations such that each iteration can work in parallel with others. In this way we can make it possible for the decoder hardware to accept streaming input. This will increase the throughput of the decoder many folds.
Chapter 6 Conclusion

In this chapter I have concluded the achievements of this thesis and the improvements that are due in the present architecture and hardware implementation for better decoding performance and efficient hardware resource utilization.

6.1 Conclusion

In this Thesis our main focus was on the hardware implementation of LDPC codes. We have seen that the latest FPGAs are large enough to handle fully parallel implementation of various moderate length of LDPC decoders. We have also proposed a highly parallel, pipelined hardware architecture for LDPC codes which can support streaming decoding which enables the decoder to outperform all previous decoder implemented on FPGAs as per our knowledge. We have also taken full advantage of inherent parallelism of the LDPC algorithm to reduce the latency of the decoding block. Our design can take as low as 6 clock cycles per iteration at 200MHz clock frequency which makes the total latency of 0.03µs per iteration. With all these improvement we have showed that a very high throughput up to 65 Gbps is achievable within reasonable resource utilization with a single instantiation of the decoder. Also we have shown that our architecture is fully parameterized and any LDPC code can be adopted at the compilation time.

6.2 Future work

The pipelined architecture we have proposed is very flexible and some modifications to this architecture can make it run time reconfigurable as well. Run time reconfigurable computing capability with high throughput and low latency can make them a lot more cost effective in comparison to ASICs for low scale commercial productions. Also some advanced variations of LDPC Min-Sum algorithm like
normalized Min-sum discussed in [17] can be applied for the same architectural framework to get some performance gain in error correction capability of the decoder.

Performance gain can be achieved by implementing even higher blocks of codewords. The results of our FPGA implementations shows that both Xilinx and Intel FPGAs are capable of mapping higher lengths decoding blocks.

The architecture given in this thesis only discusses fully parallel implementation for very high throughput applications. The same architecture can be used to optimize the resource utilization for the applications where very high throughput is not required. We can achieve this by implementing run time reconfiguration ability and semi parallel implementation. Evaluation of such a semi parallel architecture with pipeline implementation for decoding iterations for applications that does not required high throughput is also included in the future works.
Bibliography


