Master Thesis

Low Power LDPC Decoder design for 802.11ad standard

By:
Sergey Skotnikov

Supervisors:
Nicholas Preyss
Alessandro Cevrero
Matthew Weiner
Preface

Working and writing my thesis in exchange at University of California, Berkeley was a great opportunity and I would like to thank from the bottom of my heart Professor Borivoje Nikolic and Professor Yusuf Leblebici for providing it to me. There was never such a resourceful and enriching time in my life and the last 6 months were an unforgettable experience that I wouldn’t have had if not for them.

I would also like to thank Professor Andreas Burg and Nicholas Preyss for supervising the project and for their guidance in this endeavor. A separate gratitude goes to Matthew Weiner who was always there when I needed any help and to all the staff and students at Berkeley Wireless Research Center for their friendliness and support.

Lastly, I would like to thank my family for having been there for me and I felt their presence and care even from the other side of the planet. It is by knowing how much they are proud of me, no matter what I do, that I strive for perfection and excellence in my life.

Sergey Skotnikov
Contents

Preface ................................................................................................................................. i
List of Figures ...................................................................................................................... iv
List of Tables ......................................................................................................................... vi
Chapter 1. Introduction ......................................................................................................... 1
  1.1 Abstract ........................................................................................................................ 1
  1.2 Task .............................................................................................................................. 1
  1.3 Organization ................................................................................................................ 2
Chapter 2. Theory ................................................................................................................... 3
  2.1 Basic Signal Processing Theory .................................................................................. 3
    2.1.1 Shannon Limit ......................................................................................................... 3
    2.1.2 Signal Encoding and Decoding .............................................................................. 3
    2.1.3 Generator and Parity Check Matrices ................................................................... 5
    2.1.4 Soft and Hard Decoding ....................................................................................... 6
  2.2 LDPC Codes ................................................................................................................. 8
    2.2.1 General Notions ..................................................................................................... 8
    2.2.2 Sum Product Decoding ......................................................................................... 9
    2.2.3 Iterative Schedule Representation ....................................................................... 14
Chapter 3. Existing Architecture ......................................................................................... 15
  3.1 LDPC Decoder Architecture ...................................................................................... 15
    3.1.1 Overall Architecture ............................................................................................ 15
    3.1.2 Structured LDPC Matrices .................................................................................. 17
  3.2 Existing Design ............................................................................................................ 18
    3.2.1 Decoding Matrices ............................................................................................... 18
    3.2.2 Overall Design ..................................................................................................... 20
    3.2.3 Variable Node ....................................................................................................... 21
    3.2.4 Check Node .......................................................................................................... 23
    3.2.5 Pipelining ............................................................................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.6</td>
<td>Operating Results</td>
<td>27</td>
</tr>
<tr>
<td>3.2.7</td>
<td>Power Consumption</td>
<td>28</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Simulated Improvements</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>General Notions</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation Parameters</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Reduced Precision</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Dynamically Reduced Precision</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Dynamically Removed Marginalization</td>
<td>38</td>
</tr>
<tr>
<td>4.6</td>
<td>Reduced Marginalization</td>
<td>42</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Implemented changes</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>Verilog</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Wiring</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>Control and Memory</td>
<td>53</td>
</tr>
<tr>
<td>5.4</td>
<td>Reduced Marginalisation</td>
<td>53</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Results and Discussion</td>
<td>56</td>
</tr>
<tr>
<td>6.1</td>
<td>Resulting Tables</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Verilog remake Comparison</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Reduced Marginalisation Comparison</td>
<td>57</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusion and Future Work</td>
<td>58</td>
</tr>
</tbody>
</table>

References

i
List of Figures

Figure 1 Message over AWGN channel with and without encoding .................................................. 4
Figure 2 Generator and Parity-Check Matrices in canonical form ..................................................... 6
Figure 3 Hard Decoding Detector Slicing .............................................................................................. 7
Figure 4 Soft Decoding Detector Slicing .............................................................................................. 7
Figure 5 LDPC H-Matrix and corresponding Tanner Graph ................................................................. 9
Figure 6 Sum Product Algorithm. From [9] ......................................................................................... 10
Figure 7 Check Node Simplified Sum-Product Algorithm example .................................................... 13
Figure 8 LDPC Decoder Fully Parallel and Fully Serial Structures mapped from the same H-Matrix. From [1] ......................................................................................................................... 15
Figure 9 Variable wiring for parallel-serial design .............................................................................. 16
Figure 10 All-zero Matrix .................................................................................................................. 17
Figure 11 1-shifted Identity Matrix .................................................................................................... 17
Figure 12 Regular Decoding Matrix .................................................................................................. 17
Figure 13 802.11ad LDPC decoding matrices .................................................................................... 18
Figure 14 Merging of Rows for 802.11ad Rate 5/8 Matrix ................................................................ 19
Figure 15 Overall 802.11ad LDPC Decoder Design. From [1] (altered) .............................................. 20
Figure 16 Variable Node internal Structure From [1] .......................................................................... 22
Figure 17 Check Node Sign Computation XOR tree from [1] .............................................................. 23
Figure 18 Check Node Compare Select Block Tree from [1] ................................................................. 24
Figure 19 Full Check Node Design Optimised for 802.11ad Matrices and Row Merging From [1] ....... 25
Figure 20 No-pipelining Decoding Schedule From [1] (altered) .......................................................... 26
Figure 21 Pipeline Register Placement (in blue) .................................................................................. 26
Figure 22 13/16 Matrix Pipelining From [1] ....................................................................................... 27
Figure 23 Lower-rate Matrices Pipelining (3/4, 5/8, 1/2) from [1] ....................................................... 27
Figure 24 Power Consumption Distribution for 802.11ad Decoder from [6] ........................................ 28
Figure 25 Shannon Limit on Eb/No vs. generic LDPC Decoder performance with variable block length (di)
From [5] ............................................................................................................................................... 31
List of Tables

Table 1 802.11 Decoding Matrices Properties ................................................................. 19
Table 2 Original Decoder Results From [1] ........................................................................ 28
Table 3 802.11ad LDPC Decoder Register Power Consumption Breakdown From [1] .......... 29
Table 4 Variable-to-Check Node Wiring as Inferred from Rate 1/2 Matrix............................. 51
Table 5 Variable to Check Node Optimised Wiring ............................................................ 52
Table 6 LDPC Decoder comparison at synthetized frequencies and voltages ....................... 56
Table 7 LDPC Decoder comparison at 0.8V and 150 MHz .................................................. 56
Table 8 LDPC Decoder comparison at 0.8V and 75 MHz .................................................... 56
Chapter 1. Introduction

1.1 Abstract

In signal transmissions the goal is always to send the message at the highest information rate with lowest amount of errors possible. In wireless technology Shannon theorem postulates that the reliable transmission of the signal is possible above a certain signal to noise ratio (SNR). The reliability of the transmission is dependent on the encoding and decoding scheme of the network.

Low-Density Parity Check (LDPC) codes present a high performance at the limit of the theoretical maximum for reliable transmission. They achieve high bit rates at low SNR with low bit-error rate (BER) and are considered to be one of the best decoding algorithms. With the push for the 60 GHz transmission band, rises the necessity for the fast and reliable decoder. However at a high bit-rate such decoders process a lot of information and therefore consume a lot of power.

LDPC decoders in question suffer from a large wiring overhead, and at high bit rates (above 1 Gb/s) they consume more than the desirable amount of power for such a circuit (> 50 mW). The advances in this area are important as the decoder is often used in mobile devices where the longevity of battery life is paramount.

This work focuses on adapting and modifying the existing LDPC decoder design in order to lower the power consumption without sacrificing the excellent performance required for a high transmission rate. The decoder is being rewritten from scratch and several solutions are modeled and implemented to test the changes in power consumption.

The decoder in question is an improved version over the standard version featuring serial-parallel design, extensive pipelining and adaptable wiring. The current work is aiming to adapt the structure for a very specific 802.11ad standard, streamlining the components in an attempt to gain better performance from the circuit.

1.2 Task

The goal of this research is to find and implement power reducing techniques on a high-throughput Low Density Parity Check Decoder optimized for 802.11ad standard. In order to achieve the goal, the design had to be rewritten in Verilog and a tradeoff between loss of performance and reduced power consumption investigated. Special attention was paid to reducing the number of power-hungry registers.
in the design. The final performance is compared to the original design and its Verilog version, conclusions are drawn on the methods used and possible further investigations.

1.3 Organization

In this work I will first discuss the basics of signal processing theory in Chapter 2, including the Shannon theorem and the need for encoding and decoding. I will then move on to discussing the coding algorithms and in particular the LDPC parity check matrices and their design. The decoding algorithm will be discussed in detail as it forms the basis for developing the decoder hardware.

The existing architecture review follows, with detailed description of the blocks within the decoder. The goal is to have a clear vision of the design and how it relates to the decoding matrices as well as understanding the existing modifications and new solutions for improved efficiency.

Chapter 3 is focused on the original design. It describes the working of a generic LDPC decoder and the already existing innovations in the current design. The chapter demonstrates the link between the theoretical algorithm and its hardware implementation. It will allow to show the state of art and provide a basis for my research, modifications and improvements.

At first the potential hardware improvements were simulated using a decoder emulator in C++. Those various tests and their results are reported in Chapter 4. Only the most successful or important tests are discussed, as well as the reasons why they were made and how they can be implemented into the design.

In Chapter 5, I’ll discuss the actual designed modifications to the decoder that got beyond the realm of simple simulations. Those include a revamped and simplified wiring scheme, internal nodes modifications and tweaking the marginalisations.

The focus of the research being the improvement of the power consumption of the decoder, the results are reported in Chapter 6 where the original design is first compared to its rewritten Verilog version and then they are both compared to the improved version of the decoder with reduced marginalisation.
Chapter 2. Theory

2.1 Basic Signal Processing Theory

2.1.1 Shannon Limit

The wireless transmission of a signal over an AWGN (Additive White Gaussian Noise) channel is a subject of study by various researchers. The research became of great importance within the last decade with the rise of mobile and smartphone use as well as a proliferation of various modes of wireless communication between devices almost to the point of saturation of the available spectrum (e.g. Wi-Fi, 3G and LTE networks etc.).

In 1948 Shannon published what might be the most important paper in the field of signal processing which first introduced the concept of the Shannon limit for transmission over an AWGN-channel. Shannon’s theorem states that for many common classes of channels there exists a channel capacity $C$ such that there exist codes at any rate $R < C$ (in bits per second) that can achieve arbitrarily reliable transmission in which the error rate limit goes to zero, whereas no such codes exist for rates $R > C$. In other words if $R > C$ the probability of an error at the receiver increases with no upper bound, however if $R < C$ there exists an encoding/decoding algorithm that would allow the transmission to be reliable. (The theorem doesn’t include the rare case of $R = C$).

The theorem was first introduced by Shannon in [2] and its proof can be seen in [4]. We’re only interested in the final result of the theorem as it forms the core of the research into decoding algorithms.

The Shannon theorem postulates that for a band-limited AWGN channel, the capacity $C$ in bits per second (b/s) depends on only two parameters, the channel bandwidth $W$ in Hz and the signal-to-noise ratio $SNR$, as follows:

$$C = W \log_2 (1 + SNR) \text{ b/s}$$

Therefore for every channel of a certain bandwidth there exists a hard limit on transmission speed.

The capacity of the channel expressed in the Shannon formula represents the net rate of information bits without the redundant bits introduced by the coding scheme.

2.1.2 Signal Encoding and Decoding

The transmission of information over the wireless channel is a non-deterministic (unreliable) process. The following example illustrates the need for encoding.
Figure 1 shows the transmission of an information word over an AWGN-channel. The AWGN-channel, as it follows from its name, is characterized by the white Gaussian noise it introduces to the signal that passes through it. The information byte (here: 110011) is transmitted without encoding in top design. Should a noise be present on a channel high enough to cause an uncertainty at the receiver, i.e. for the low-SNR signal, the received byte would get its information bits flipped in certain places, making the transmitted signal incorrect. In this case, without any possibility to restore the original information, the received signal produces an error and prevents the correct operation of the system.

With the presence of the encoder and the decoder (the added/redundant bits from the encoder are not shown on bottom image) the recovery of the correct signal can be performed using various decoding methods and therefore weaker signals can still be interpreted correctly even when certain bits are received at a wrong value.

Encoding is an operation performed on the information stream before the transmission which adds the redundant bits into the message. Therefore each codeword contains information bits, which are actual useful data that is transmitted, and redundant bits, which are the bits that are introduced by the encoding schemes to improve the transmission reliability. The decoder on the receiver side is necessary to iteratively restore the original codeword even if certain bits were unreliably transmitted over the channel due to the presence of the redundancy. The chosen algorithm is called the error correcting code (ECC).

The most common types of ECCs are repetition codes, Hamming Codes, turbo and LDPC codes. The information about these codes can be found in [4].

The three characterizing parameters that are used to describe ECCs are the length, dimension and Hamming distance.

**Length** (denoted \( n \)) defines the total number of bits in the codeword after the encoding. In a 2-bit message and encoding, this translates into a codeword \( n \)-tuple.
**Dimension** (*denoted k*) is the number of binary n-tuples that constitute the code. The parameter k identifies the number of information bits in the codeword and consequently each code has \(2^k\) possible codewords. For example: to encode a 4-bit message, we can have \(4^2 = 16\) possible permutations of the information bits and need 16 codewords to cover all of them.

The **Hamming distance** (*denoted d*) is the minimum number of bits that separate the two closest codewords in the code. The Hamming distance is an indicator of the robustness of the code. The higher is the Hamming distance between the two codewords, the less chance there is to confuse the two and achieve wrong results at the decoder for high SNR. The minimum Hamming distance is equal to the smallest Hamming weight of the non-zero codeword in the code.

The standard notation for linear codes is an \((n, k)\)-notation that determines the parameters of the code. The examples of such linear codes are the \((n, 0)\) – all 0-vector code, which is a trivial code and \((n, n)\) which includes all the possible permutations of the n-tuple and therefore is called the universe code.

The example of the \((5, 2)\)-code is given below. The number of codewords is \(2^k = 2^2 = 4\), including the all-zero and all-one codewords. The following constellation can be derived:

\[
\begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1
\end{pmatrix}
\]

In which the two topmost bits are the information bits for each possible information word ((00), (11), (10), and (01)) and the three bottom bits are redundant bits. The Hamming distance in this case is \(d = 2\) which is the weight of the third codeword.

The biggest challenge for the ECCs is to attain the Shannon limit, i.e. to allow the information rate to be close to the theoretical maximum with the probability of the error at the receiver being arbitrarily small.

### 2.1.3 Generator and Parity Check Matrices

The generator matrix is a basis for a linear code and is used to form all the possible codewords. A linear \((n, k)\)-code has a \(k \times n\) generator matrix as it translates all possible k-tuple information bits into n-tuple codewords. The following definition applies:

For a linear \((n, k)\)-code \(C\) and a generator matrix \(G\) every n-tuple \(q\) of the code is obtained by:

\[
q = \vec{c}G
\]

where \(\vec{c}\) is a row vector of information bits.
Every codeword which constitutes the alphabet of the code [4] is generated by multiplying the incoming information stream by the generator matrix.

The parity check matrix (denoted H) is the generator matrix of the dual code of C where the dual code of C (denoted here as C’) is defined in such way so that the product of a word from C and its dual C’ is always 0:

\[ C' = \{ w \in F_q^n \mid < w, q > = 0, \forall q \in C \} \]

\( F_q^n \) is the finite field of \( n \)-tuples for an alphabet of size \( q \). Further discussion of finite fields can be viewed in [4] and is not a subject of this study.

The parity check matrix is a dual of the generator matrix and can be derived from it. Every linear code possesses a generator matrix and a parity check matrix. A linear \( (n, k) \)-code has an \((n-k) \times n\) parity check matrix and every product of an \( n \)-tuple codeword and the parity check matrix yields 0 using binary arithmetic.

\[ Hq = 0, \forall q \in C \]

In wireless transmission the encoder is the hardware implementation of the generator matrix, while the decoder is the hardware implementation of the parity check matrix which allows the decoding algorithm to iterate and check the validity of the received message.

As a simple example, (taken from Wikipedia) both matrices are shown in their canonical form on Figure 2. The generator matrix will form a \((5, 2)\) – code, each 5-tuple of which will give 0 when multiplied by \( H \).

\[
G = \begin{bmatrix} 10 & 101 \\ 01 & 110 \end{bmatrix} \quad H = \begin{bmatrix} 11 & 100 \\ 01 & 010 \\ 10 & 001 \end{bmatrix}
\]

Figure 2 Generator and Parity-Check Matrices in canonical form

2.1.4 Soft and Hard Decoding

The incoming message to the decoder from the detector at the receiver can take several forms.

Hard decoding is performed when the incoming message from the detector consists of only 1 single bit. The value is decided using a threshold at the receiver. The threshold is computed based on channel
characteristics. The values above the threshold will be treated as 1 and values below as 0. Hard decoding yields hard decisions on the variables at each cycle.

Soft decoding implies multi-bit resolution. In this case not only do we receive the value of the signal from the receiver but also its probability to be true via extra bits added to the message. This is called the reliability of transmission. In this case the message is presented in sign-magnitude format where the sign is the value of the message (1 or 0 like in hard decoding) and the magnitude is the probability of being correct. If the magnitude is low then the received value is considered unreliable during the assessment in the decoder which can influence its algorithm.

The number of magnitude bits increases the complexity of the decoder but also allows it to better assess the incoming message and therefore gives it a better chance of successful decoding. The mere presence of the reliability bits allows soft decoders to make better assumptions on data compared to the hard decoders which don’t have any probability values to work with and all incoming bits are treated equally.

It is therefore preferable to use soft-decoding algorithms whenever possible especially for high-throughput systems where the bit-error and consequently frame-error rates have to be. This will be discussed further into the work.
2.2 LDPC Codes

Low Density Parity Check (LDPC) codes were first invented by Gallager in 1963 [3] however they haven’t made it past the theory until the last 15 years because the hardware requirements for the implementation of the scheme were too high at the time, due to the excessive wiring overhead such designs require.

Since the technology to effectively implement the scheme became less costly due to the miniaturization of the digital architecture in the late 1990s the LDPC codes regained the attention of the researchers due to their efficiency and their performance close to the Shannon limit [14][15].

2.2.1 General Notions

The notions introduced in this section describe the decoder part of the LDPC code, i.e. its parity check matrix implementation. The encoder uses the LDPC generator matrix and is not a subject of this research.

The LDPC code is a linear block code defined by an $M \times N$ sparse parity check matrix $H$. The $N$ denotes the number of bits in the codeword (or a block) and $M$ the number of parity checks. One will note that this translates perfectly from the theoretical notion of the parity check matrix. For the codeword to satisfy the parity checks means that it’s multiplication by the matrix yields 0. It is worth noting that in order to achieve 0 in binary arithmetic the resulting product of the codeword and a row of the $H$-matrix must have a pair number of 1s, hence the parity-check.

By design, the matrix defining the LDPC code has to be sparse, which implies a low density of 1s. It also has to be large. The LDPC code is identified by its rate $R$ which is calculated as follows:

$$R = \frac{N - M}{N}$$

In the $(n, k)$ notation, we have $N = n$, and $M = n-k$, therefore the code rate $R = k/n$ which signifies the proportion of information bits in the block. The larger proportion of information bits can lead to greater throughput however the error-rate is higher due to lack of redundant (parity check) bits.

The example on Figure 5 will illustrate the principle for a simple LDPC matrix. The $M$ rows (here 4) signify the number of parity checks while the $N$ columns (here 6) stand for the 6-tuple to process through the checks. The 1 on the intersection signifies which bits will participate in a parity check while 0 signifies the bits that do not participate in the check. For parity Check 1 we can see that bits 1, 3 and 4 are processed therefore their addition under binary arithmetic has to yield zero in case of the correct codeword.
The bipartite graph on the right is the graphical representation of the LDPC parity check matrix and is called the Tanner Graph representation. The bottom vertices are assigned to each bit in the code block while the top vertices substitute parity checks. Each arrow on the graph is a visual representation of the ones in the parity check matrix showing which checks affect which bits.

In hardware, each bit in the code block in LDPC decoder is mapped to a Variable Node (VN) while the parity checks are mapped to the Check Node (CN).

<table>
<thead>
<tr>
<th></th>
<th>Bit1</th>
<th>Bit2</th>
<th>Bit3</th>
<th>Bit4</th>
<th>Bit5</th>
<th>Bit6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CN2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CN3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CN4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**2.2.2 Sum Product Decoding**

In general, the decoders can be one-shot – receive inputs, compute the hard results and quit, or iterative where the message is being processed and modified via the internal decoder algorithm for several cycles. In this case the decoder converges on a result and quits the iterative algorithm if the hard decision is correct (i.e. it passes through the H-matrix), or it quits after the maximum number of iterations has been completed and no satisfactory result has been computed.

The LDPC decoder uses a soft-decoding iterative algorithm called belief propagation to compute the output. This is a message passing algorithm which is most easily described as the Sum-Product Algorithm or SPA.

In the LDPC decoder the messages are being passed between the variable and check nodes and vice versa for iterative decoding. Soft decoding implies that the messages are not just single bit received values but actual probabilities of a received value being 1 or 0.

The message sent from a certain variable node \( v_i \) to a connected check node \( c_j \) contains information on the probability of a certain value given the initial signal from the channel as well as all the other checks but the one it’s sent to (all \( c_y \) connected to \( v_i \), \( y \neq j \)).
Similarly the message sent from the check node $c_j$ back to the node $v_i$ contains the probability that the variable node $v_i$ has a certain value after having compared the messages sent to this particular check node apart from the once from $v_i$ (all $v_x$ connected to $c_j$, $x \neq i$).

The following graph on Figure 6 visually shows the flow of the sum-product algorithm. The $q_{ij}$ and $r_{ij}$ messages correspond respectively to variable-to-check-node and check-to-variable-node messages. The messages are passed between the $i$th variable node and $j$th check node.

The notation also means that the underlying LDPC H-matrix consists of $i$ columns for each VN and $j$ rows for each CN.

The following iteration algorithm is discussed using the LLR notation and transformations. For the original algorithm using probabilities from which the following is derived, please consult [4]. The thorough study of Sum-Product algorithms is performed in [11] for deeper knowledge.
1. INITIALISATION

The inputs to the designed LDPC decoder are Log-Likelihood Ratios (LLR) from the received signals, defined as:

\[ L^{pr}(x_i) = \log \frac{Pr(x_i = 0 | y_i)}{Pr(x_i = 1 | y_i)} \]

Where \( x_i \) is the bit value of a sent signal, and \( y_i \) the actual signal value.

This equation maps the higher probability of 0 to a positive value and higher probability of the negative value to a negative number, down to infinity if certainty is absolute. Each Variable Node receives a value for the bit it processes at the beginning. The range of this value is defined by the number of bits in the received message according to the soft decoding theory presented earlier.

The value is stored within the variable node for the duration of the decoding and is called a prior value.

2. ASSEMBLE VARIABLE TO CHECK NODE MESSAGE

The variable-to-check-node message between the \( i \)th variable and \( j \)th check nodes is composed of all the messages returned to the VN from all the CNs but the one the message is sent to summed with the prior of that VN.

\[ L(q_{ij}) = \sum_{j \in \text{Col}(i) \setminus j} L(r_{ij'}) + L^{pr}(x_{i'}) \]

In the first cycle the message simply consists of the prior value itself, while further iterations imply the marginalizing of the summed message received from the Check Nodes.

For example is VN1 is connected to CN3, CN5 and CN7. In the first cycle it sends the prior value it received in step 1 to each of those check nodes. In subsequent iterations the message sent to CN3 will be a sum of the prior value and the answers received from CN5 and CN7 but not CN3. In this way the message sent to CN3 contains only the external influence of the checks performed in all the nodes connected to VN1 (CN5 and CN7) but itself and therefore it is not biased by its own calculation that might be faulty.

Marginalisation is a necessary part of the decoding algorithm.
3. FORM CHECK TO VARIABLE NODE MESSAGE

The goal of the check node is to process the messages received from the variable nodes and if the result is equal to 0, then the check is satisfactory. This is the equivalent of the codeword conforming to the parity check matrix $H$.

In binary arithmetic such comparison is done by multiplying the received sign values as a pair number of 1s in the message would yield 0. For soft decoding the check nodes also process the probability of each variable to be correct.

The hard decision can be made at the output of the check node for the conformity of the codeword. At the same time, the probability of the check can be also computed. In LDPC decoding, the probability of the check is determined by aliasing the incoming messages from variable nodes using the $\Phi$ function:

$$\Phi(x) = -\log\left(\tanh\left(\frac{1}{2x}\right)\right), x \geq 0$$

The full form of the check-to-variable-node message is then:

$$L(r_{ij}) = \Phi^{-1}(\sum_{i'\in\text{Row}[j]\setminus i} \Phi(|L(q_{ij})|)) \ast \left(\prod_{i'\in\text{Row}[j]\setminus i} \text{sgn}(L(q_{i'j}))\right)$$

The analysis of the $\Phi$ function shows that the output magnitude of the Check Node is dominated by a low probability input magnitude. This means that the probability of the correct message analysis in the Check Node is approximately equal to the reliability of the most dubious message it receives from connected Variable Nodes. We can then approximate the check-to-variable-node message and completely remove the $\Phi$ function and the complexity it entails:

$$L(r_{ij}) = \max \{\min_{i'\in\text{Row}[j]\setminus i} |L(q_{i'j})| - \beta, 0\} \ast \left(\prod_{i'\in\text{Row}[j]\setminus i} \text{sgn}(L(q_{i'j}))\right)$$

This formula equates the reliability of the correct check to the reliability of the least probable message minus the parameter $\beta$ which is empirically adjusted to approximate the effect of the $\Phi$ function. It is usually small or non-existent.

If the check node possesses 8 inputs with the received VN values as described in Figure 7 then the output would be the product of the signs and the lowest input magnitude which is 2. The product of the signs gives 0 because the number negative values, which from the LLR equations mean that the assumed received bit value is 1, is pair. Therefore the output of this CN is +2 and the parity check is considered passed.
Once again the message is marginalized for the particular variable node. It works in the same manner as the check node message marginalizing.

In the example on Figure 7 if the input 1 was received from VN1, then the actual message sent to that node must exclude its contribution to the evaluation. It will then receive the extrinsic information from all the other nodes it was processed with. In this case the sign will be marginalized and VN1 will receive \(-2\) as the answer while the computed value in the CN is positive. In case of Input 4 (assuming it comes from VN4) the magnitude has to be marginalized and the message sent from CN to VN4 will be 3 as the sign is preserved and second minima value is chosen according to the simplified formula.

4. UPDATE VARIABLE NODE MESSAGE

The message received from the check node is used to update the internal value stored into the variable node by summing all the incoming messages as well as the prior LLR.

\[
L^{ps}(x_i) = \sum_{j \in \text{col}[i]} L(r_{ij'}) + L^{pr}(x_i)
\]

In the previous example of VN1 connected to CN3, CN5 and CN7 the value at the end of the decoding cycle (after the full matrix is processed) \(L^{ps}\) will be the sum of prior LLR and all of the messages received from CN3, CN5 and CN7.
Note that due to the marginalization of the CN message, if the check node passes the parity-check (i.e. it receives a pair number of ones) the returned messages will reinforce the message already stored in the VN.

It can be seen from the example on Figure 7. VN1 sends -8 to the CN and, while the output of CN is +2, the marginalized message to VN1 will be -2, therefore the sum at the variable node will be -10 which reinforces the reliability of having 1 at this node. In the same way if the check node doesn’t pass the parity check it will make the joined variable nodes internal values less reliable and can flip some values if the prior reliability is too low.

If a hard decision is required from the variable node the sign of the $L^{ps}$ determines the hard decision from the node, according to the same principles that govern the prior LLR. The steps 2 to 4 are looped to perform the iterative decoding of the message.

### 2.2.3 Iterative Schedule Representation

In the iterative decoder we can rearrange the equations to show the connections between the iterations.

The updated variable to check node message is simply the stored message minus the message received from the Check Node at iteration $n-1$.

$$L_n(q_{ij}) = L_{n-1}^{ps}(x_i) - L_{n-1}(r_{ij})$$

The new variable node value is computed by simply updating it with the message from the connected check nodes after the new iteration.

$$L_n^{ps}(x_i) = L_{n-1}^{ps}(x_i) - L_{n-1}(r_{ij}) + L_n(r_{ij}), \quad j \in Col[i]$$

These equations better illustrate of marginalization in the variable nodes which will be discussed in detail further.
Chapter 3. Existing Architecture

3.1 LDPC Decoder Architecture

3.1.1 Overall Architecture

The LDPC Decoder architecture is derived directly from the Tanner Graph for the corresponding H-Matrix. Its design can vary from being fully parallel, in which case the hardware maps every Variable and Check Node directly to the hardware, to fully serial, in which case only one Variable and Check Node exist in hardware with large memory banks to store data to pass messages. Both mappings are shown on Figure 8.

The fully parallel decoder design benefits from faster processing time since the matrix is encoded directly into the design however it is also an inflexible solution. The decoder can only process the one matrix that was transcribed into the design which severely reduces the practicality of such approach since it can’t be used in any design where a slightest degree of flexibility is required. The fully parallel design can achieve the decoding in lesser number of clock cycled however this solution requires additional hardware and the bloated structure leads to complicated wiring. This causes a large wiring overhead for fully parallel implementations and the wiring congestion which increases the size of the chip. Moreover the wiring congestion leads to longer wiring path which induce longer critical path, therefore lowering the maximum clock frequency at which such decoder can operate.

![Figure 8 LDPC Decoder Fully Parallel and Fully Serial Structures mapped from the same H-Matrix. From [1]](image-url)
The fully serial design is the most flexible solution as the H-Matrix implementation is done through memory banks and control signals. The hardware only represents one Check Node and one Variable Node wired together with the memory array which stores all the passing messages. Depending on the decoding schedule. Due to the simplicity of the design, the clock frequency of such circuit is usually very high, however the throughput of a fully serial system is very slow as it needs to process one connected node pair at a time. Compared to the fully parallel decoder this design doesn’t suffer from wiring congestion and offers great flexibility. At the same time its throughput is so dismal that it’s of little use in a high-throughput application.

Any solution that falls between the fully serial and fully parallel ones, is called serial-parallel design. In this case a part of Variable and Check Nodes is implemented. The goal is to find a middle solution that would solve the decoding matrix and keeps as much flexibility as possible inherited from a fully serial design, while avoiding the wiring overhead of the fully parallel design. The process requires an appropriate scheduling to process an irregular number of nodes. In simplest terms if we compare a fully parallel design to the one where only a half of the Variable Nodes is implemented, it would require additional memory framework within the nodes themselves and two clock cycles to process the same amount of nodes in the parallel-serial design.

The parallel-serial design for a general non-structured decoding matrix suffers from a fatal flaw which is a complexity of scheduling. This manifests in excessive or sometimes irresolvable wiring, or its scheduling. In Figure 9 we have a variable number of Variable Nodes connected to one Check Node at each cycle. Should the hardware be designed for a random matrix, each Check Node would have to have enough inputs to accept simultaneous signals from each Variable Node in case the matrix possesses the row of 1s. This bloats the hardware and creates wiring congestions making the parallel-serial designs for random decoding matrices unrealistic.

![Figure 9 Variable wiring for parallel-serial design](image)
3.1.2 Structured LDPC Matrices

The introduction of structured LDPC Matrices allowed a much easier implementation of a parallel-serial design. These matrices are subject to a rigid set of rules by which they are created. The point of this thesis is not to discuss their elaboration and further information can be found in [1] and [4]. Nevertheless a short overview is necessary to understand the reason for the chose solution and the implications to the wiring.

A structured matrix is composed of smaller subset matrices of size $L \times L$ (square matrices). Those submatrices can be either an all 0-matrix or a shifted identity matrix. The example of such matrices is shown in Figure 11 and Figure 10.

![Figure 10 All-zero Matrix](image)

![Figure 11 1-shifted Identity Matrix](image)

The general LDPC matrix consists only of a combination of those two, and uses a notation where each block of known dimensions $L$ is either an all-0 submatrix block represented as empty, or a shifted identity matrix block represented by the number of shifts to the right. The following example on Figure 12 illustrates such matrix for a submatrix of size $4 \times 4$.

![Figure 12 Regular Decoding Matrix](image)

The conventional way to design a decoder using such matrices is to note that the Variable and Check Nodes in the decoder that is defined by such matrix can now be grouped to form Variable Node Groups and Check Node Groups respectively. The size of a group is identical to the size of the submatrix. The fact that each submatrix is very simple implies that the wiring between two groups is easy, as each Check Node from a group is connected exactly to one Variable Node of the group in the case of the non-zero matrix, due to the properties of the Identity matrix. The parallelism of the decoder is viewed in terms of how many groups of Variable or Check Nodes are actually implemented in hardware.
3.2 Existing Design

3.2.1 Decoding Matrices

The existing design is an improved version of the standard LDPC decoder designed specifically for the 802.11ad single carrier standard, which defines 4 regular LDPC matrices, designed specifically to simplify the hardware implementation. The matrices are presented on the Figure 13.

**Rate 13/16**

<table>
<thead>
<tr>
<th></th>
<th>29</th>
<th>30</th>
<th>0</th>
<th>8</th>
<th>33</th>
<th>22</th>
<th>17</th>
<th>4</th>
<th>27</th>
<th>28</th>
<th>20</th>
<th>27</th>
<th>24</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>37</td>
<td>31</td>
<td>18</td>
<td>23</td>
<td>11</td>
<td>21</td>
<td>6</td>
<td>20</td>
<td>32</td>
<td>9</td>
<td>12</td>
<td>29</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>22</td>
<td>4</td>
<td>34</td>
<td>31</td>
<td>3</td>
<td>14</td>
<td>15</td>
<td>4</td>
<td>2</td>
<td>14</td>
<td>18</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rate 3/4**

<table>
<thead>
<tr>
<th></th>
<th>35</th>
<th>19</th>
<th>41</th>
<th>22</th>
<th>40</th>
<th>41</th>
<th>39</th>
<th>6</th>
<th>28</th>
<th>18</th>
<th>17</th>
<th>3</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29</td>
<td>30</td>
<td>0</td>
<td>8</td>
<td>33</td>
<td>22</td>
<td>17</td>
<td>4</td>
<td>27</td>
<td>28</td>
<td>20</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>31</td>
<td>18</td>
<td>23</td>
<td>11</td>
<td>21</td>
<td>6</td>
<td>20</td>
<td>32</td>
<td>9</td>
<td>12</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>22</td>
<td>4</td>
<td>34</td>
<td>31</td>
<td>3</td>
<td>14</td>
<td>15</td>
<td>4</td>
<td>14</td>
<td>18</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rate 5/8**

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>36</th>
<th>34</th>
<th>31</th>
<th>20</th>
<th>7</th>
<th>41</th>
<th>34</th>
<th>10</th>
<th>41</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td>27</td>
<td>18</td>
<td>12</td>
<td>20</td>
<td>14</td>
<td>2</td>
<td>25</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>28</td>
<td>3</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>0</td>
<td>22</td>
<td>4</td>
<td>28</td>
<td>27</td>
<td>24</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>23</td>
<td>21</td>
<td>20</td>
<td>9</td>
<td>12</td>
<td></td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>34</td>
<td>31</td>
<td>14</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>

**Rate 1/2**

<table>
<thead>
<tr>
<th></th>
<th>40</th>
<th>38</th>
<th>13</th>
<th>5</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>34</td>
<td>35</td>
<td>27</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>31</td>
<td>7</td>
<td>34</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>18</td>
<td>12</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>0</td>
<td>22</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>23</td>
<td>21</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>34</td>
<td>31</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>

*Figure 13 802.11ad LDPC decoding matrices*
The submatrices have a dimension of 42x42. The matrices process 672 Variable Nodes in one decoding. The matrices have a variable row and column degrees \(d_v\) and \(d_c\), respectfully as rows represent check node groups and columns variable node groups) and their properties are summarized in Table 1.

The presented matrices are created specifically to allow the possibility of improving the design to increase the throughput and decrease power consumption.

We can note that the 13/16 and 3/4 rate matrices are very dense, i.e. they don’t feature many all-zero matrices, while the lower rate matrices have a lot of non-overlapping gaps. The all-zero matrix allows to collapse layers and process the matrix in fewer cycles. In rate 5/8 the top two layers are non-collapsible however layers 3 and 5, as well as layers 4 and 6 can be merged as seen on Figure 14.

Following the same logic and noticing that the bottom four rows of ½ and 5/8 rate matrices are identical it is easy to notice that in ½ rate matrix the following pairs of rows are collapsible: \((1,3)\) \((2,4)\) \((5,7)\) \((6,8)\). Therefore every presented matrix can be condensed to a 4-row matrix, which is an important property as it allows to process the matrix faster with proper hardware design.

### Table 1 802.11 Decoding Matrices Properties

<table>
<thead>
<tr>
<th>Matrix</th>
<th>(d_c)</th>
<th>(d_{v, min})</th>
<th>(d_{v, max})</th>
</tr>
</thead>
<tbody>
<tr>
<td>13/16</td>
<td>3</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>3/4</td>
<td>4</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>5/8</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>1/2</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 14 Merging of Rows for 802.11ad Rate 5/8 Matrix
3.2.2 Overall Design

The implemented LDPC decoder uses a parallel-serial design with fully parallel implementation of 672 variable nodes and serialized 42 check nodes. In accordance to the submatrices size the nodes are grouped in clusters of 42, therefore the design incorporates 16 variable node groups (VNG) and 1 check node group (CNG). A simplified view of the overall design can be seen on Figure 15.

We can now see that each row in code matrices can be viewed as a CNG and each column as a VNG.

The serialization of Check Nodes implies that their access is time-multiplexed. Each row of the matrix can be processed in one clock cycle. However due to the presence of collapsible layers the lower rate matrices can be processed in 4 cycles, just as quickly as the non-collapsible rate ¾ matrix.

The decoding cycle starts at the VNs which send out simultaneously their result to the respective CNs according to the processed layer of the matrix. Due to the matrix being regular there need not be more than 16 inputs on each check node to properly process the matrix. Due to the structure of the Identity matrix which doesn’t change during the shifting only 1 input from each VNG can go to a specific CN in one cycle. As the matrix is separated in 16 VNGs the result is directly inferred.

In comparison for an irregular matrix of this size (672 VNs) each CN would require 672 inputs in order to process the matrix.
The algorithm uses flooding scheduling, meaning that all messages are accumulated and updated in variable nodes before being sent to the check nodes instead of constantly updating itself (which would be layered scheduling). The differences between the scheduling types are not discussed in this work and can be viewed in [1]. Alternative scheduling methods exist in order to improve the algorithm however they are not subject of this research [10].

The barrel shifters are inserted before and after each node group. They are the hardware implementation of the Identity Matrix Shift. The forward shift is executed in front shifters, and the backwards shift in back shifters to assure that the messages from CNs go to proper VN nodes. The proper functioning of the shifters allow to simplify the analysis of the decoding matrix and view the overall design in terms of CNGs and VNGs and not separated nodes.

The length of the codeword is an important parameter as has been discussed in soft decoding theory. The original design runs at a 5-bit wordlength where the most significant bit (MSB) is the sign of the value from the LLR and the 4 remaining bits are its magnitude.

The magnitude can be split into fractional and integer bits. This step doesn’t influence the design of the decoder and is implemented before the input. The performance, however can be drastically different. The number of integer bits allows for a greater swing in magnitude value, however fractional bits add more precision to the calculations.

For example should all the magnitude bits be integer in the designed decoder, the maximum magnitude value would be 15 (4 bits). Therefore during the LLR assessment stage, every received value above 15 (very certain) is cropped down to that number while the values between 15 and -15 are mapped directly. The precision is 1 in such case, however the reliable bits carry more weight and cannot be easily flipped.

If the decoder uses 4 bits and splits them in 2 integer and 2 fractional bits, then the maximum magnitude value is only 4. Therefore all the stronger signals are cropped down to that value. The precision of calculations, however will be of 0.25 (2 fractional bits). In this case, the calculations are much more precise, however there is less difference between the certain bits and the dubious ones.

3.2.3 Variable Node

The sum-product algorithm equations directly influence the internal hardware of the variable node which can be seen on Figure 16. The current design allows the simultaneous processing of two frames, which doubles the rate of the decoding. It is discussed further in pipelining explanation.

During the initialization phase the prior LLR are stored in a register and its value is sent bypassing the accumulators to the output to CNs for the first iteration.
On successful iterations the prior value is added to the accumulator along with the results arriving from the previous CNs. The value in the accumulator is being updated for four cycles necessary to process all the time-multiplexed CNs after which it is being sent for the next cycle to the corresponding CNs.

Marginalization of check-to-variable-node (C2V) and variable-to-check-node (V2C) messages is also being performed in the VN. Marginalization is very important for proper functioning of the algorithm and is described in the sum-product equations.

Before the message is sent to the check node i for any iteration after the first one, according to the sum-product algorithm equation, the value received from that CN in previous cycle must be subtracted. The way the VN works is that it stores the message from all cycles summed up with the prior in the accumulator, and keeps in memory the messages received from the CNs during the 4 accumulation clock cycles. Then during the next 4 clock cycles when the V2C message is being output, the message is formed by taking the sum from the accumulator and subtracting the stored CN message from it. This process is called V2C marginalization.

C2V marginalization is performed because of the simplification of the check node processing algorithm. The simplified algorithm sends back the computed C2V message with the weight of the least reliable message received by the CN. However the algorithm dictates that during the processing of the C2V message to the specific VN the node must not take into account the message incoming from this particular VN. This would create a complicated hardware design in the check node and therefore it processes all the VNs and sends identical messages back with two minimum weights attached, however
in each VN the previously output V2C message is stored in memory and compared to the message sent back from the CN.

If those messages are identical for the lowest weight then the second lowest weight is chosen for that particular VN to be processed and added to the accumulated value. The sign of the message is also marginalized by multiplying it with the stored value.

The accumulated sum can be used to output the hard decision when requested which is the last function implemented in the VNs.

### 3.2.4 Check Node

Check Node design is very straightforward due to the simplification of the $\Phi$ function. The simplified design requires the computation of the sign which is the product of all the arriving hard values according to 2-bit arithmetic and can be implemented as a simple XOR tree as seen on Figure 17. All the 16 inputs are multiplied with each other.

![Figure 17 Check Node Sign Computation XOR tree from [1]](image)

The check node also needs to compute two minima which will be sent back to VNs for soft decoding as the reliability of the computed result. This is implemented in form of a compare-select block tree, where
inputs from each VN are being compared one to each other until only the two smallest values remain. As can be seen from the simplified sum-product algorithm equations these are the exact values to be sent back in C2V message considering that the marginalization is being done in VNs both for sign and magnitude.

The processing of collapsible rows requires additional enhancements of the basic design. The check node as presented on Figure 18 processes all the messages, therefore it takes one row at a time, however when the two matrix rows are merged their sign and magnitude values have to be compared separately for each merged row which complicated the design of the wiring and the check node.

The first point to infer from the matrix design is that the maximum number of non-zero matrices for every merged row combination does not exceed 8. That means that for any combination of two rows we can separate the check node in two identical smaller check nodes taking in 8 inputs each and process outputs separately.

Moreover such design does not impede the ability to process one complete 16-input row as an extra compare select block can be inserted to select the absolute two minima from the inputs of the internal 8-bit blocks.
The complete design of the check node magnitude tree compatible with row merging is shown on Figure 19. A control signal (TwoLayers on the diagram) is required to select an appropriate output into the pipeline stage that follows the CN depending on whether one row is being processed or two merged rows with separate calculations. In the latter case the wiring must take care of connecting the required messages into the top and the bottom circuit.

The CS blocks take 4 inputs and output two minimal weights.

3.2.5 Pipelining

To increase the decoder throughput the hardware can be modified in a way to process two independent 672-bit frames at the same time.

This is possible due to the collapsible structure of the regular 802.11ad LDPC matrices as well as clever scheduling and hardware design tweaks in nodes and wiring.

It is known from the LDPC matrices that after the layer collapsing it is possible to process the whole matrix in 4 clock cycles at best, because the check nodes are serialized and time multiplexed, if the clock cycle is long enough to clear the check nodes. The flooding scheduling requires that all the messages from the check nodes are to be summed before the new ones can be sent. This means that until the last message from the last matrix row is not processed and added in the VN accumulators it is impossible to send the new messages to iterate through the matrix once more starting from the top row.
This situation is illustrated in Figure 20. It is noticeable that while the messages are being accumulated the time and hardware is wasted in waiting.

![Figure 20 No-pipelining Decoding Schedule From [1] (altered)](image)

To maximize the effectiveness of the design, 4 pipeline stages have to be implemented into the wiring of the decoder. This ensures the synchronization between the 4 cycles it takes to accumulate the message in the VNs and 4 cycles it takes to process the other message through the wiring and the check node so there is no extra delay. Their placement is shown on Figure 21.

![Figure 21 Pipeline Register Placement (in blue)](image)

In this scenario as soon as all the messages are accumulated they can be output back into the wiring as shown on Figure 20. This eliminates the dead time between the cycles.
From the Figure 20 it can be deduced that the time between iterations of a single frame is sufficient to process another one. The extra registers to operate two frames are inserted into the design of the Variable Node and operate in alternate fashion. These extra registers include one for an extra prior as well as the extra accumulator for the second frame as seen on Figure 16.

Figure 23 shows the perfect pipelining for the LDPC matrices for 802.11ad standard. This result is achieved if exactly 4 pipeline registers are inserted and show that there are no idle cycles in the loop. The only exception is the rate 13/16 code which can be processed in 3 cycles and which pipeline is shown on Figure 22. Due to the generalized structure of the decoder the pipeline has an idle stage which is replaced in the design with dummy messages in order to simplify the controls. Dummy messages do not alienate the algorithm when processed.

### 3.2.6 Operating Results

The design was run through Design Compiler and IC Compiler and then tested at different clock frequencies yielding the results summed up in Table 2.

The original design was developed in Simulink and mapped to gates through Insecta tool. The design was elaborated at 200 MHz clock at 1.20V. The results were then scaled down to the operating values.
The throughput decoder scales linearly with the clock frequency as well as the power consumption. The design was synthesized using a modified version of ST 65nm toolkit. The analysis of the results can be viewed in (INSERT REFERENCE HERE)

### 3.2.7 Power Consumption

In order to effectively reduce the power consumption of the decoder one must first understand the parts that dissipate the most of it. The following results were obtained for a version of a pipelined decoder for the same 802.11ad standard with different memory cell technology and presented in [6].

<table>
<thead>
<tr>
<th>Technology</th>
<th>ST 65nm low power CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Area</td>
<td>1.3mm²</td>
</tr>
<tr>
<td>Operating Class</td>
<td>1</td>
</tr>
<tr>
<td>Coded Throughput</td>
<td>1.54Gb/s</td>
</tr>
<tr>
<td>Clock Frequency</td>
<td>75MHz</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>0.8V</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>42mW</td>
</tr>
</tbody>
</table>

*Table 2 Original Decoder Results from [1]*

The throughput decoder scales linearly with the clock frequency as well as the power consumption. The design was synthesized using a modified version of ST 65nm toolkit. The analysis of the results can be viewed in (INSERT REFERENCE HERE)
The graph on Figure 24 shows that more than half of the total power comes from Memory (i.e. pipeline registers) even after using a modern memory cell design. It is also shown that in memory power the largest amount of losses come from buffer cells for data alignment and extrinsic memory for the data exchanged between the nodes. Those result are logical considering the high level of switching activity in the pipeline registers compared to those storing prior and posterior results. The implemented decoder dissipates over 65% of its power in the pipeline registers due to their switching activity. The pipelining which allows to decode two frames at a time without wasting time also ensures that the majority of the pipeline registers are switching at every clock cycle. The Variable Nodes house the largest number of those registers and consume almost 60% percent of all the register power. The results are summarized in the Table 3 below.

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Number</th>
<th>Total Power (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Node</td>
<td>672</td>
<td>59.1</td>
</tr>
<tr>
<td>Pipeline Register</td>
<td>4</td>
<td>19.5</td>
</tr>
<tr>
<td>Back Shifter</td>
<td>16</td>
<td>6.2</td>
</tr>
<tr>
<td>Front Shifter</td>
<td>16</td>
<td>4.4</td>
</tr>
<tr>
<td>Check Node</td>
<td>42</td>
<td>4.4</td>
</tr>
<tr>
<td>Pre-router</td>
<td>672</td>
<td>2.5</td>
</tr>
<tr>
<td>Post-router</td>
<td>42</td>
<td>1.4</td>
</tr>
<tr>
<td>Other Logic</td>
<td>-</td>
<td>2.5</td>
</tr>
<tr>
<td><strong>Total Register Power</strong></td>
<td>-</td>
<td><strong>67.5</strong></td>
</tr>
</tbody>
</table>

*Table 3 802.11ad LDPC Decoder Register Power Consumption Breakdown from [1]*

The variable nodes house a large number of registers for marginalization and storage of data. These registers are refreshed at each clock cycle which leads to an increased power consumption. 20% of the power is also consumed by the pipeline registers which are inserted to assure the fastest possible processing of data. These registers just like the ones in the variable node switch their value at each clock cycle.

The other 10-15% of power consumption comes from the inevitable losses, as well as wiring multiplexing, clock tree and control logic.

It is therefore logical to concentrate on reducing the power dissipation in the pipeline registers in the decoder especially those housed within the variable nodes as together they are responsible for almost 80% of the total power consumption. This will be the main focus of the research into reducing the power consumption of the design.
Chapter 4. Simulated Improvements

4.1 General Notions

Shannon theorem can be used to derive a Shannon limit based on error-rate versus noise, expressed as Eb/No. Such derivation can be seen in [4]. This work will explain the important values necessary for the comprehension of the simulation results.

The graphs shown in the following sections show the bit error rate (BER), frame error rate (FER) and average iteration number curves over EB/No.

Eb/No is an important parameter which is a normalized measure of the signal-to-noise ratio. For a discrete channel the information rate can be expressed as:

\[ R = \rho W \text{ b/s}, \]

Where \( W \) is the bandwidth of the channel and \( \rho \) is its spectral efficiency in \((\text{b/s})/\text{Hz}\).

The signal power (average energy per second) is:

\[ P = E_s * W \]

The SNR is expressed as the ratio of the signal energy \( E_s \) to noise energy \( N_0 \):

\[ \text{SNR} = \frac{E_s}{N_0} \]

From here we can extract the Eb/No value which will be derived from SNR:

\[ \text{SNR} = \frac{Eb}{\rho * N_0} \]

\[ \frac{Eb}{N_0} = \text{SNR}/\rho \]

Eb/No is a measure of signal strength compared to noise and can be viewed as SNR per bit. There is a Shannon limit on Eb/No which defines the lowest possible ratio after which no decoding algorithm can reliably restore transmitted information.

An example is given on Figure 25. At low Eb/No the LDPC code with an infinite block length (n) cannot assure the acceptable error-rate. This area is called the Shannon limit on Eb/No.
BER is the rate of individual bits that were not properly decoded using the algorithm. At low Eb/No the messages received are virtually indistinguishable from noise and therefore their reliability is low and the results are almost random. The performance of the decoder is the severely limited as the data corruption is too high.

For high Eb/No when the signal is strong the errors arise from the internal decoding algorithm. For the LDPC decoder at a certain Eb/No the BER reaches its lowest point and saturates. This phenomenon is called the error floor and is due to a certain decoding patterns which cannot solve the errors.

FER is the rate of complete frames that were not properly decoded. This value is directly related to BER as any bit error will lead to the wrong codeword at the output and therefore to a frame error.

Average number of iterations measures the speed of the convergence of the decoder. The decoder is limited to a certain number of iterations per frame before it gives up. However if the codeword is decoded correctly before the limit is reached, the algorithm quits and the new frame is loaded. At higher Eb/No the signal is strong and therefore the algorithm decodes the errors much faster. Lesser amount if iterations per decoding leads to a higher throughput of the decoder.
4.2 Simulation Parameters

Simulation were performed using a model of the decoder written in C++. This is not the model of the implemented decoder rather than a golden model of a simple design. The code can take any decoding H-Matrix as an input and emulate the resulting decoder function.

The tested matrices included the high rate ¾ matrix as well as a low rate ½ matrix in order to test the changes in different settings. The code also allows to vary the wordlength of its operating signals. Most Configurations were run using the real existing design as a starting point and a comparison point, therefore the simulations were usually using a 5-bit wordlength although in order to reduce power this value was modulated in some cases.

4.3 Reduced Precision

The implemented LDPC decoder design works with 5-bit words therefore there are 32 precision levels in the signal. In sign-magnitude notation, the first bit is responsible for the sign and the 4 tail bits represent certainty ranging from 0 to 15.

To compare the raw performance of the decoder a simulation was performed where the length of the codeword has been reduced by 1 or 2 bits. Such an analysis was originally performed during the elaboration of the initial design to maximize the performance-to-power-consumption ratio.

Predictably, the BER of a design with a shorter wordlength (and therefore lesser amount of magnitude bits) is much higher which renders decoding at high frequency impossible. It is however worth noting on Figure 28 that the BER and FER do not diverge drastically until 4.2 Eb/No. From the Figure 29 we can misleadingly believe that removing a bit doesn’t yield any mosses for high Eb/No however such design exhibits a much earlier and higher bit-error-floor and is therefore inherently weaker.

At the same the difference in performance between a 5-bit and a 4-bit designs is not as drastic as the gap between the 4-bit and the 3-bit designs. Therefore in the next session an attempt to save power by reducing precision in the middle of the decoding cycle will be analysed. The potential energy gain is high as precision affects all the registers in the variable node and the pipeline as shown on Figure 26 and Figure 27.

For the simplicity of comparison, all the results are performed on a 5-bit wordlength decoder with 4 integer magnitude bits. Simulations for other cases with fractional bits were performed with comparable results. The decoder is also implemented with 4 integer magnitude bits in mind.
Figure 26 Pipeline stages (in red) are all affected by reducing precision.

Figure 27 Reduced Precision in Variable Node (circled registers are affected).
Figure 28 Matrix Rate 3/4 varying wordlength from 5 to 3 bits (top - BER, left - FER, right - Avg. Iterations)
Figure 29 Matrix Rate 1/2 varying wordlength from 5 to 3 bits (top - BER, left - FER, right - Av. Iterations)
4.4 Dynamically Reduced Precision

In order to reduce the power consumption while maintaining the BER floor relatively high the possible solution could be the dynamic reduction of the wordlength. The decoding begins with the 5-bit wordlength and after a certain amount of iterations one bit is removed from every pipeline register, prior value, accumulator etc. This simulation has been performed for rate ¾ matrix with 4 integer magnitude bits.

The hardware implementation of such solution requires extra scheduling and heavy modification of the control node. It is also problematic to decide which register is easier to turn off. The simulations show the result when the signal value is adjusted to a lesser amount of bits after a certain amount of iterations, which is the same as cropping the MSB magnitude bit.

As can be seen from Figure 30 the reduction of precision of the registers during the decoding heavily degrades the performance. The BER floor is present at high BER making this solution incompatible with higher bit-rates.

The implementation of such method in the real design is quite tricky because, while most of the information is passed in sign-magnitude format, the values stored in the Variable Nodes are converted to twos complement representation due to a heavy amount of arithmetic in the node (summation in the accumulators and marginalisations). Simple turning off the MSB in all the registers will yield erroneous results.

Empirical approach would be needed to assess the performance of such modification or a fundamental overwrite of the C++ code to better reflect the decoder hardware.
Figure 30 Matrix Rate 3/4 dynamically reduced wordlength (top - BER, left - FER, right - Avg. Iterations)
4.5 Dynamically Removed Marginalization

The key focus of the work is to find ways to reduce power consumption of the decoder while maintaining the BER at the approximately the same level.

As seen from the analysis of the power consumption of the LDPC Decoder the best way to drastically reduce the power consumption is to find ways to reduce the power consumption in the pipeline stages. The two ways to reduce the power which are independent on technology is to reduce the size of the pipeline stages or to reduce their switching activity.

The reduction of switching activity is complicated without breaking the decoding algorithm as in ideal case every stage should switch and change its value at each clock cycle, apart from several registers (e.g. prior registers, VN accumulators alternatively during output stages keep their value constant). This problem comes directly from the dense pipelining and the ability to process two frames at the same time.

Reduction in the size of the stages leads directly to the reduction of the wordlength which according to soft-decoding algorithm reduces the precision of the weight and raises the probability of error. It is however possible to change the precision (and register size) of certain elements of the decoder without sacrificing the overall precision.

The following figures show the effect that V2C and C2V marginalization has on the decoding algorithm. The decoding was performed using the normal algorithm, however after a certain amount of iterations the marginalisation was completely removed.

Figure 31 Reduced/Removed Marginalisation in Variable Node (red circle: C2V marginalisation affected, blue circle: V2C Marginalisation affected)
Figure 31 shows the registers that are affected by the reduction of marginalization. The saved values in those registers usually consist of 5 bits and switch at every clock cycle. Their elimination allows a considerable reduction in power consumed by the variable node. As seen from section 3.2.7 variable node consumes almost 60% of total power in the decoder, therefore it is very interesting to see whether removing or tweaking marginalization allows to keep the BER stable.

The results on Figure 32 and Figure 33 show that completely removing marginalisation for V2C or C2V message is ruinous for the algorithm. The designs in which either marginalisation is missing are completely non-functional. They prove that marginalisation is vital to the algorithm. The situation doesn’t improve by much if the marginalisation is removed after a certain amount of iterations. In fact, the decoder almost never reaches a good result unless it’s able to compute it before the marginalisation is turned off.

V2C marginalisation is shown to have a slightly lesser effect on the decoding accuracy, with the BER rate increasing by an order of magnitude with it being turned off. Without the C2V marginalisation the BER jumps by more than two orders of magnitude for the decoder.

At high Eb/No the average number of iterations per decoding is low therefore turning off marginalisation after several iterations does not influence the algorithm as much. In this case, if the decoder reaches a conclusive result before the marginalisation registers are powered off, there is no gain in power consumption.

It is therefore non-productive to simply ignore or switch off the marginalisation and a more subtle approach is required.
Figure 32 Matrix Rate 3/4 dynamically removed C2V marginalisation (top - BER, left - FER, right - Avg. Iterations)
Figure 33 Matrix Rate 3/4 dynamically removed V2C marginalisation (top - BER, left - FER, right - Avg. Iterations)
4.6 Reduced Marginalization

While removing marginalisation involves a drastic change in the algorithm it is also possible to reduce the size of the registers that are responsible for the marginalisation. The decoder uses a 5 bit codeword with 4 magnitude bits. The sign bit is necessary for both marginalisations to add the correct value. We will then discuss the effects of reducing the magnitude of marginalisations.

For C2V marginalisation, the CN sends two minima magnitude values and if the first minimum is identical to the one stored within the VN node memory, the second minimum magnitude is used instead. The question is then, how many bits of the minimum is it sufficient to compare in order to make a relatively informed guess.

Figure 35 shows the gradual removal of MSBs from the magnitude of the Variable-to Check Node (V2C) marginalisation. 5 MSB removed signifies that the marginalisation is completely turned off for the sake of comparison. It is clear that strong marginalisation values do not play an important part in determining the accuracy of the algorithm and can be removed without any loss in precision from the design. There is little noticeable difference in decoder performance even if 3 MSB bits are removed. The design shows a jittery behavior for 1 MSB removed at low Eb/No, which is an artifact of the random sample selection.

The logical explanation to this behavior is the fact that the subtracted message is the one arriving from the check node, which according to the simplified decoding algorithm keeps the lowest magnitude of the incoming signals. Therefore the probability of subtracting a message with a strong magnitude in V2C marginalization is extremely low as it would require all 16 inputs to the check node having a strong magnitude.

In these situations, however, it is unusual for the values to be incorrect, as their reliability is high therefore the algorithm doesn’t care for that particular marginalization.

The gradual removal of LSB bits from the V2C marginalisation is also performed and the results are shown on Figure 36. 5 LSB removed signifies that the marginalization is completely turned off. Once again it is shown that the removal of a single LSB doesn’t induce drastic changes in the BER and FER curves behavior compared to the unaltered design. However the removal of 2 or more LSBs leads to a jump in BER.

By the same reasoning if the probability of having a low magnitude in V2C marginalizing is very high, removing those LSB effectively equates to removing V2C marginalization entirely. By combining the results of the two simulations it is interesting to see that most of the performance of V2C marginalization is related to the middle bits. The removal of either MSB or LSB does not affect the decoding potency of the structure.
A different method was used to model the C2V marginalization which relies on comparison between the value stored in the VN and the incoming message from CN. In the simulation presented on Figure 37 compares magnitudes were both aliased using a bitwise AND function. This allows to selectively compare certain magnitude bits.

The example of such comparison is shown on Figure 34. The incoming 9-bit message from CN contains the sign value (1) and two minima (0001) and (0111). The signs are separated and the magnitudes are compared. In this case the stored magnitude (0101) is not identical to the lowest in the message and therefore the marginalized C2V value proceeds to summation in the accumulator with weight (0001).

In the simulation on Figure 37 both magnitudes are aliased by a certain AND condition. If the marginalization is aliased by AND 3, then the compared values are (0001 & 0011 = 0001) from CN and (0101 & 0011 = 0001) stored in VN. In this case an error is induced as not enough bits from both sides got compared. The marginalization then proceeds with the wrong weight and might affect the performance of the decoder.

This aliasing effectively emulates the fact that only several bits of the outgoing V2C message are stored and compared against the incoming message. It also allows to exactly select the bits that are going to be removed compared to simply switching off LSBs and MSBs.

In this decoder the magnitude is mapped over four integer bits and is therefore constrained between 0 and 15. The result of this simulation shows that if the comparison is reduced to just comparing the LSB (both signals aliased by 3 (4’b0011) or 7 (4’b0111) then the marginalization is ineffective and severely impacts the performance of the decoder.

At the same time if only the MSB are compared, the performance doesn’t suffer. It is therefore possible to remove several LSB from the C2V marginalizing registers within the variable node.
Figure 35 3/4 Matrix Removing MSB from V2C Marginalisation
Figure 36 3/4 Matrix Removing LSB from V2C Marginalisation
Figure 37 $1/2$ Matrix C2V Marginalisation Aliasing
Chapter 5. Implemented changes

5.1 Verilog

The original design was implemented via the Matlab plugin Simulink and employed custom blocks, written in Verilog as well as premade proprietary Xilinx blocks. The resulting design was then processed through the Insecta tool which derives the gate-level Verilog design from Simulink.

Due to the complexity of the representation as well as difficulty in iterating the modifications to the design the whole decoder was rewritten in Verilog making a completely fresh code, using the pre-existing Memory and Control Blocks. The functional implementation of the nodes remains identical to the original design while some nodes were optimized due to the changes in the wiring.

5.2 Wiring

The original design was not specifically optimized for the particular decoding scheme, the only limiting parameter being the total size of the parity-check matrix, therefore it featured an adaptable and versatile yet quite cumbersome wiring.

The original wiring can be seen on Figure 38, for the sake of comparison, and features a set of routers for every output, considering the possible matrix permutations. The limiting parameter for this design is for the whole matrix to be processed in 4 clock cycles or less and be compatible with the LDPC matrix construction mechanics. The wiring requires a 16-bit control signal which is generated at the same time as the matrix and all the values are stored during the initialization phase in the memory.

During the elaboration of the new design in Verilog the wiring was completely rewritten sacrificing the versatility for much lower wiring overhead and design simplicity. The original wiring is a better version for a random standard, however several optimizations were made specifically for 802.11ad matrices during the redesign that allows faster clocking and significant overhead reduction, which is one of the biggest problems with LDPC decoders.

The wiring design begins with the assessment of the Check Nodes. The simplification and the merging of layers in LDPC implementation is based on the fact that within each check node there are two identical compare-select blocks that process 8 top and 8 bottom inputs separately and then in case of processing two layers the check node produces two separate outputs while in case of processing one layer the additional compare-select stage is used and one single output is achieved. These properties can be used to greatly simplify the wiring.

The barrel shifters placed after the Variable node group assure that at the output we receive the Identity matrix. It simply means that the first output of the barrel shifter from each Variable node group will
always go to the first check node, the second output to the second check note etc. independent on the internal permutation of variable nodes within the group. From the check point perspective it means that the first check node will receive 16 signals, one from each topmost output of every barrel shifter. To assure the correct decoding we only need to properly assign the incoming signals to the top and bottom circuits depending on the processing matrix rate.

This situation can be seen on Figure 39 where the 1st output of the barrel shifter (BS) 1 and 2 go both to the first check node, 2nd outputs go to the second check node and so on.
The main concern is then how to attribute those check node inputs into the top and bottom circuit respectively.

If we process the full row (i.e. compare-select all 16 inputs), the location of those inputs on the Check Node is irrelevant as they will all be compared with each other. This means that for Rate 13/16, 3/4 and the first two checks of rate 5/8 we don’t need to regulate the V2C wiring as long as the barrel shifters assure the proper rotation. Any wiring permutation of the inputs would work in these cases. We only need to assure that the incoming wiring is properly wired for the cases when two rows are processed simultaneously in the check nodes, because in these cases the wiring must take care of arranging the inputs that are compared against each other in either top or bottom node.

---

**Figure 39 Barrel Shifter Function and Output Schematic**
Considering that the bottom rows of rate 5/8 matrix are identical to those of rate ½ matrix we only need to examine the wiring for rate ½ matrix to solve the overall wiring as it presents all the possible cases of having two rows analyzed at the same time. The following table shows the connection of input signals and the respective input on the check node. There are only four cases in which the rows are merged therefore we only need 4 wiring paths to assure the correct functioning of the decoder for the rate ½ matrix.

The results shown in Table 4 are a direct mapping of the rate ½ matrix onto the wiring pattern. It is important to note that we do not care about the check nodes for they are all identically wired, but the important part is their inputs. This table shows the identical wiring solutions for every check node out of 42.

The check node inputs highlighted in green all receive the same signal from the barrel shifters at each iteration and therefore do not require any multiplexors in the routing. The values in red are unassigned, however in order to satisfy the property of the identity matrix it is required that at each iteration there can be no two signals from the same Variable node group wired to the same check node. Therefore an optimization is required, taking into account the fact that it is preferable to limit the number of multiplexors in the design in order to simplify the wiring and reduce power during the switching.

The final solution for the wiring that maximizes the number of fixed connection is given in Table 5.

The resulting wiring contains 10 multiplexed paths and 6 directly wired connections which are identical for every check node. Due to the fact that the wiring is irrelevant for the case of processing a single matrix row, the four wiring paths suffice to process any of the decoding matrices. Therefore the control signal for the routing is simplified to a 2-bit signal (in case the permutations were to be random the control signal would have to be sent via the 16-bit bus).

The wiring can only process the 802.11ad matrices. A different wiring is required if a different set of matrices is to be processed.

The wiring simplification is estimated to reduce the area of the decoder and slightly influence its power consumption due to the fact that the number of multiplexors compared to the original design is an order of magnitude lower. The real result is hard to compare as the design has been completely rewritten with numerous smaller changes that could influence the data.
<table>
<thead>
<tr>
<th>Top Circuit</th>
<th>Bottom Circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN_x input</td>
<td>VNG Connected</td>
</tr>
<tr>
<td>1</td>
<td>0 (00)</td>
</tr>
<tr>
<td>1</td>
<td>1 (01)</td>
</tr>
<tr>
<td>1</td>
<td>2 (10)</td>
</tr>
<tr>
<td>1</td>
<td>3 (11)</td>
</tr>
<tr>
<td>2</td>
<td>0 (00)</td>
</tr>
<tr>
<td>2</td>
<td>1 (01)</td>
</tr>
<tr>
<td>2</td>
<td>2 (10)</td>
</tr>
<tr>
<td>2</td>
<td>3 (11)</td>
</tr>
<tr>
<td>3</td>
<td>0 (00)</td>
</tr>
<tr>
<td>3</td>
<td>1 (01)</td>
</tr>
<tr>
<td>3</td>
<td>2 (10)</td>
</tr>
<tr>
<td>3</td>
<td>3 (11)</td>
</tr>
<tr>
<td>4</td>
<td>0 (00)</td>
</tr>
<tr>
<td>4</td>
<td>1 (01)</td>
</tr>
<tr>
<td>4</td>
<td>2 (10)</td>
</tr>
<tr>
<td>4</td>
<td>3 (11)</td>
</tr>
<tr>
<td>5</td>
<td>0 (00)</td>
</tr>
<tr>
<td>5</td>
<td>1 (01)</td>
</tr>
<tr>
<td>5</td>
<td>2 (10)</td>
</tr>
<tr>
<td>5</td>
<td>3 (11)</td>
</tr>
<tr>
<td>6</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>1 (01)</td>
</tr>
<tr>
<td>6</td>
<td>2 (10)</td>
</tr>
<tr>
<td>6</td>
<td>3 (11)</td>
</tr>
<tr>
<td>7</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>1 (01)</td>
</tr>
<tr>
<td>7</td>
<td>2 (10)</td>
</tr>
<tr>
<td>7</td>
<td>3 (11)</td>
</tr>
<tr>
<td>8</td>
<td>NULL</td>
</tr>
<tr>
<td>8</td>
<td>1 (01)</td>
</tr>
<tr>
<td>8</td>
<td>2 (10)</td>
</tr>
<tr>
<td>8</td>
<td>3 (11)</td>
</tr>
</tbody>
</table>

*Table 4 Variable-to-Check Node Wiring as Inferred from Rate 1/2 Matrix*
<table>
<thead>
<tr>
<th>CN_x input</th>
<th>VNG Connected</th>
<th>Layer (sel)</th>
<th>CN_x input</th>
<th>VNG Connected</th>
<th>Layer (sel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 (00)</td>
<td></td>
<td>9</td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 0 (01)</td>
<td></td>
<td>10</td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 0 (10)</td>
<td></td>
<td>11</td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3 0 (11)</td>
<td></td>
<td>12</td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 0 (00)</td>
<td></td>
<td>13</td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 0 (01)</td>
<td></td>
<td>14</td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 0 (10)</td>
<td></td>
<td>15</td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 0 (00)</td>
<td></td>
<td>16</td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0 0 (00)</td>
<td></td>
<td></td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 0 (00)</td>
<td></td>
<td></td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 0 (00)</td>
<td></td>
<td></td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 0 (00)</td>
<td></td>
<td></td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0 0 (00)</td>
<td></td>
<td></td>
<td>0 0 (00)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 0 (01)</td>
<td></td>
<td></td>
<td>1 0 (01)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2 0 (10)</td>
<td></td>
<td></td>
<td>2 0 (10)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3 0 (11)</td>
<td></td>
<td></td>
<td>3 0 (11)</td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Variable to Check Node Optimised Wiring
5.3 Control and Memory

Due to the changes in wiring numerous modifications were made to the control and memory nodes, simplifying their structure. Two most important improvements are summarized below, while numerous smaller improvements in particular cases are not interesting from the pure performance point of view.

- Reduction of static memory size: the memory stores values relevant to the decoding matrix. The decoder can process the matrix of any rate but cannot switch the rate in mid-process. The new wiring structure allows to remove most of the information, and only the shift values for barrel shifters have to be stored as they are very different for each matrix and do not follow a particular pattern.

- Simplification of control signals for wiring: due to the simplicity of the matrices there are only four possible wiring routes that can be used to process any code rate. This allows a 2-bit signal to control all the wiring for this format.

5.4 Reduced Marginalisation

The simulation results from section 4.6 were taken into consideration as several solutions were found where a large amount of registers could be removed without impacting the BER.

The simulations on Figure 40 and Figure 41 show the comparison of the original unaltered decoder with the one where one or both types of marginalisations were altered. In this case the C2V marginalization lost 2 LSB and therefore the registers only carry 3 bits: 1 bit for sign marginalization and 2 MSB for comparison the attached magnitude. The V2C marginalization also lost 2 bits from its magnitude correction: 1 LSB and 1MSB as previous results showed little deviation from the ideal curve with those bits missing.

In total 4 bits were removed. Considering that V2C and C2V marginalization pipeline consist of 4 registers of 5 bits (down to 3 bits each), every variable node has lost 16 registers (see Figure 31). For 672 parallel VN that are included in the design this constitutes a big part of registers removed, considering that these registers are switching their values at each clock cycle.
Figure 40 Matrix Rate 1/2 reducing marginalisations (top - BER, left - FER, right - Avg. Iterations)
Figure 41 Matrix Rate 3/4 reducing marginalisations (top - BER, left - FER, right - Avg. Iterations)
Chapter 6. Results and Discussion

6.1 Resulting Tables

The old design refers to the original decoder, the new design is the rewrite made in Verilog and the improved version has a reduced number of marginalisation registers.

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>New</th>
<th>Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Matt Weiner</td>
<td>Sergey Skotnikov</td>
<td>Sergey Skotnikov</td>
</tr>
<tr>
<td>Technology</td>
<td>ST065</td>
<td>ST065</td>
<td>ST065</td>
</tr>
<tr>
<td>Voltage (scaled)</td>
<td>0.8V</td>
<td>0.8V</td>
<td>0.8V</td>
</tr>
<tr>
<td>Clock (scaled)</td>
<td>150 Mhz</td>
<td>150 MHz</td>
<td>500 MHz</td>
</tr>
<tr>
<td>Area</td>
<td>1.31 mm²</td>
<td>1.25 mm²</td>
<td>1.29 mm²</td>
</tr>
<tr>
<td>Power Measured</td>
<td>213 mW</td>
<td>130.5 mW</td>
<td>287 mW</td>
</tr>
</tbody>
</table>

Table 6 LDPC Decoder comparison at synthetized frequencies and voltages

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>New</th>
<th>Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Matt Weiner</td>
<td>Sergey Skotnikov</td>
<td>Sergey Skotnikov</td>
</tr>
<tr>
<td>Technology</td>
<td>ST065</td>
<td>ST065</td>
<td>ST065</td>
</tr>
<tr>
<td>Voltage (scaled)</td>
<td>0.8V</td>
<td>0.8V</td>
<td>0.8V</td>
</tr>
<tr>
<td>Clock (scaled)</td>
<td>150 Mhz</td>
<td>150 MHz</td>
<td>150 MHz</td>
</tr>
<tr>
<td>Power Measured</td>
<td>84 mW</td>
<td>81 mW</td>
<td>71 mW</td>
</tr>
</tbody>
</table>

Table 7 LDPC Decoder comparison at 0.8V and 150 MHz

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>New</th>
<th>Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Matt Weiner</td>
<td>Sergey Skotnikov</td>
<td>Sergey Skotnikov</td>
</tr>
<tr>
<td>Technology</td>
<td>ST065</td>
<td>ST065</td>
<td>ST065</td>
</tr>
<tr>
<td>Voltage (scaled)</td>
<td>0.8V</td>
<td>0.8V</td>
<td>0.8V</td>
</tr>
<tr>
<td>Clock (scaled)</td>
<td>75 Mhz</td>
<td>75 MHz</td>
<td>75 MHz</td>
</tr>
<tr>
<td>Power Measured</td>
<td>42 mW</td>
<td>41 mW</td>
<td>35 mW</td>
</tr>
</tbody>
</table>

Table 8 LDPC Decoder comparison at 0.8V and 75 MHz
6.2 Verilog remake Comparison

The original design had to be rewritten in Verilog in order to make it compatible with newer processes. It also allowed to modify select internal components. This comparison is interesting as different tools were used to achieve approximately the same result.

The original design was developed in Simulink and then transcribed to gates using the proprietary Xilinx Insecta tool. The new design is written in pure Verilog. The wiring has been changed and several minor alterations within the blocks were made. Therefore the new design is expected to have comparable power consumption with less area.

Table 6 shows the raw data taken at the values using which the design was synthetized. The original design was synthetized in IC Compiler at 200 MHz at 1.20V and uses a mix of LP and GP blocks. The new Verilog design was pushed at 200 MHz and 0.90V using LP gates only.

The results have to be compared using the same frequency and voltage parameters. As seen from the Table 7 and Table 8 the differences between the two implementations are minor. The new design seems to consume slightly less power when scaled down to similar dimensions. The scaling is linear with frequency and to the power of 1.6 (empirically estimated) for voltage.

The area of the new decoder is also comparable to the older design if only slightly smaller. Evidently the designs being almost identical, the sudden change in performance is not expected.

The virtue of the new Verilog design hides mostly in its universal compatibility with various processes and tools. The code is easily modified and it’s an easier solution when implementing changes into the design as was the case with the reduced marginalization.

6.3 Reduced Marginalisation Comparison

Reducing the number of registers within the variable node brings definite improvements in terms of area and power consumption. The decoder was processed with a high-frequency imposed clock, which took its toll on the total area as the design is not much smaller than the original and larger than the rewritten Verilog synthetised with lesser constraints.
At the same time the device exhibits a 15% drop in power consumption when scaled down to working parameters. The device consumption fell from 84 mW down to 71 mW at 150 Hz. The solution to trim the marginalization has thus proven to be effective considering that it entails a minor loss in overall performance.

6.4 Conclusion and Future Work

The work on the LDPC decoder design optimized the original structure for the 802.11ad decoding scheme. Multiple methods of reducing power consumption were tested to a various degree of success. While some research focuses on technological advances to reduce the power [6], [13] and other research focuses on fundamental algorithm changes to achieve the same goals, the point of this thesis was to push the existing design to its limits and test the robustness of the algorithm.

The main focus was paid to the influence of marginalization. The reduction in the precision of marginalization allowed to save more than 15% power compared to the original design. Turning either marginalisation off proved to be a wrong way to go however combining initial cropping and then further reducing the precision of marginalisation after several iterations might save even more power if tuned correctly.

For the devices using the 802.11ad standard, this design offers a streamlined solution with adapted wiring and a lot of optimizations, some inherited from the original (pipelining, dual frame processing) and some new additions. However this design is not yet ideal.

It is of great interest to see whether the precision of certain other blocks can be reduced in the same way without impacting the BER. A deeper research needs to be done on the influence of precision on the algorithm separately at certain pipeline nodes. This research would require an extensive amount of additional scenarios and simulations and warrants a separate project.
References


[4]. Digital Communications Course II, MIT Chapters 1-10


[6]. Youn Sung Park, David Blaauw, Dennis Sylvester, Zhengya Zhang, “A 1.6-mm 38-mW 1.5-Gb/s LDPC Decoder Enabled by Refresh-Free Embedded DRAM”, 2012 Symposium on VLSI Circuits Digest of Technical Papers, p114 -115

[7]. Weiner, M., “LDPC decoder architecture for high-data rate personal-area networks”


[12]. Engling Yeo, Borivoje Nikolic, and Venkat Anantharam, “Architectures and Implementations of Low-Density Parity Check Decoding Algorithms “
[13]. Alessandro Cevrero, Yusuf Leblebici, Paolo Ienne, Andreas Burg, “A 5.35 mm2 10GBASE-T Ethernet LDPC Decoder Chip in 90 nm CMOS”, IEEE Asian Solid-State Circuits Conference November 8-10, 2010 / Beijing, China
