Lehrstuhl für
Eingebettete Syteme
der Informationstechnik

Design and Implementation of an LDPC-based FEC encoder/decoder suitable for Storage devices

Schriftliche Prüfungsarbeit für die Masterprüfung der Fakultät für Elektrotechnik und Informationstechnik an der Ruhr-Universität Bochum (Master-Prüfungsordnung für den Studiengang „Elektrotechnik und Informationstechnik“ an der Ruhr-Universität Bochum vom 12. August 2013)

vorgelegt von:

Henry Bathke

Datum: October 13, 2018

erster Betreuer:	Prof. Dr.-Ing Michael Hübner
zweiter Betreuer:	M. Sc. Keyvan Shahin

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Official Declaration

Hereby I declare, that I have not submitted this thesis in this or similar form to any other examination at the Ruhr-Universität Bochum or any other Institution of University.

I officially ensure, that this paper has been written solely on my own. I herewith officially ensure, that I have not used any other sources but those stated by me. Any and every parts of the text which constitute quotes in original wording or in its essence have been explicitly referred by me by using official marking and proper quotation. This is also valid for used drafts, pictures and similar formats.

I also officially ensure, that the printed version as submitted by me fully confirms with my digital version. I agree that the digital version will be used to subject the paper to plagiarism examination.

Not this English translation, but only the official version in German is legally binding.

Datum / Date	Unterschrift / Signature

1. Abstract

Contents

List of Figures

List of Tables

2. Motivation

3. Error Correcting Codes

For modern communications systems reliable data transmission and storage ist required. To achieve this goal usually error correcting codes are used. There are different possible codes available for error correction, but I will restrain myself to LDPC[4] codes in this thesis. As these codes can archive good performance and can be used at large block lengths[8]. This is especially useful for use with NAND based solid state drives.

When describing a block code there are important parameters as the message length k. The message is what is given into the encoder and the result from the decoder. The block length n, and the rate $R = k/n$.

In this case error correction code are used to add additional information to data to allow errors. The errors are then corrected with that information. The addition of additional information is also called redundancy. With this redundancy it is possible to lose information while data is transmitted over a channel and decode it after the channel. After decoding the original data is recovered by using the additional information. figure 3.1 shows such a system where information is transmitted, the channel can be of different type. It can for example be a wireless transmission or memory where information is first stored and later read back.

3.1. Low-Density Parity-Check (LDPC) Codes

The following section will describe LDPC codes invented by Robert Gallager[4]. Starting with a graph representation I will describe the LDPC code and then continue with a matrix representation. LDPC codes can be shown as a bipartite graph also called Tanner graph[9] based on their inventor. figure 3.2 shows an example of one, where the check and parity

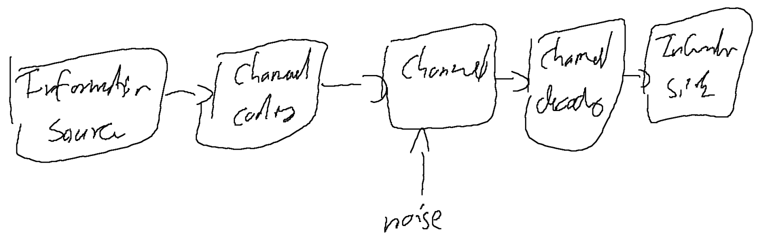

Figure 3.1.: A basic channel where information is transmitted

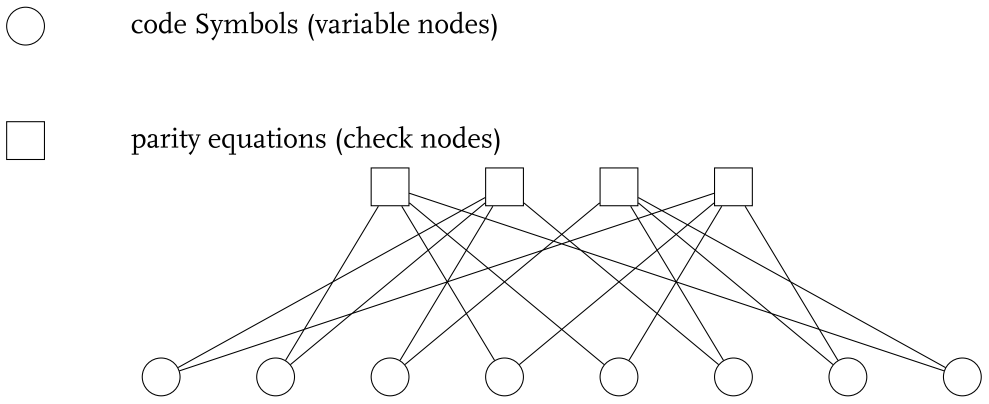

Figure 3.2.: An example Tanner graph.

nodes are connected by edges. This is an effective representation, moreover it will also help understanding the decoding algorithm later.

Instead of using the Tanner graph one can also use a matrix representation. In this matrix the ones represent the edges of the graph. Usually for a LDPC code the matrix is sparse or low density as the name implies. In equation (3.1) a matrix representing the same code as in the graph in figure 3.2 is shown. The $\boldsymbol{H}$ matrix is of size $(n-k)\times n$. And the possible code words are given by the null space of $\boldsymbol{H}$, so in other words c is a code word if and only if $c\boldsymbol{H}^T = \mathbf{0}$[7].

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.1}$$

3.1.1. Quasicyclic LDPC (QC-LDPC) codes

LDPC codes can be difficult to implement, especially randomly generated ones. On the other hand structured codes can be devised to be more easily implemented. One class of these structured codes are quasicyclic LDPC codes. In these codes the parity check matrix is only built from circulant matrices and zero matrices[3]. The circulant matrices are defined by their first row as the following rows are the first one shifted. The base matrix specifies where zero matrices and where rotated circulant matrices are placed. Each circulant matrix has size $p\times$. The parity check matrix has size $(n-k)\times n = p(N-K)\times pN$. Thus the base matrix $\boldsymbol{B}$ has $(N-K)\times N$ entries. Often times the circulant matrix is the identity matrix and is rotated by the corresponding amount in the base matrix. In this base matrix the elements can be either a shift factor smaller that the circulant size $0 \leq b_{ij} < p$ or -1 representing a zero matrix.

In the following example the circulant matrix is a 5×5 identity matrix.

$$\boldsymbol{B} = \begin{bmatrix} -1 & 0 & 2 & -1 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 0 & 2 \end{bmatrix} \tag{3.2}$$

After the expansion the matrix looks like this:

$$\boldsymbol{H} = \begin{bmatrix} 0&0&0&1&0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&1&0&0&0&0&1 \\ 0&0&0&0&0&1&0&1&0&1&0&0 \\ 0&0&0&1&0&0&0&0&1&0&1&0 \end{bmatrix} \tag{3.3}$$

more qc specifics?

3.1.2. Encoding

Genrator Matrix

For encoding the probably simplest algorithm is transforming the parity check matrix into systematic form $\boldsymbol{H} = \begin{bmatrix} -\boldsymbol{A}^T & \boldsymbol{I}_{n-k} \end{bmatrix}$. Where $\boldsymbol{I}_{n-k}$ is a $n - k \times n - k$ identity matrix and $\boldsymbol{A}$ has $k \times n - k$ elements. To archive this form one could for example use gaussian elimination. With $\boldsymbol{A}$ known we can construct the generator matrix $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{I}_k & \boldsymbol{A} \end{bmatrix}$. Now encoding can be done with a simple matrix multiplication. With u the information word and v the code word is given by $v = \boldsymbol{G}u$.

Take for example the matrix from equation (3.1). If we use gaussian elimination to bring the right side to identity we are left with equation (3.4). Now we take the left part of the matrix and transpose it to get $\boldsymbol{A}$. With we build $\boldsymbol{G}$ in equation (3.5).

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The main disadvantage of this strategy is the high computational complexity. When transforming the parity check matrix into systematic form we have a complexity of $\mathcal{O}(n^3)$. This is not to bad as it will mostly be done offline and only the $\boldsymbol{G}$ matrix stored in the encoder, but the bigger problem is that due to the gaussian elimination the matrix is no longer sparse. Thus the matrix multiplication will result in a complexity of $\mathcal{O}(n^2)$[6].

Approximate Lower Triangular Form

Richardson and Urbanke[7] describe a way to reorder the parity check matrix to reduce the encoding complexity. They bring the matrix into a so called approximate lower triangular form. This is done by only doing row and column permutation, so the low density of the matrix is kept. The resulting structure of the matrix is shown in figure 3.3. Especially advantageous is the reduced complexity for encoding, here the encoding complexity is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n + g^2)$, where g is the gap. This gap is the number of rows that cannot be brought into triangular form, as seen in figure 3.3. We can also write

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{E} \end{bmatrix} \tag{3.6}$$

the submatrices all have the dimensions given in figure 3.3. By multiplying equation (3.6) with

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{T}^{-1} & \boldsymbol{I} \end{bmatrix} \tag{3.7}$$

the resulting matrix is

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ -\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A} + \boldsymbol{C} & -\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B} + \boldsymbol{D} & \boldsymbol{0} \end{bmatrix} \tag{3.8}$$

. By splitting the codeword into three parts $c = \begin{bmatrix} s & p_1 & p_2 \end{bmatrix}$ and applying the definition for valid code words $\boldsymbol{H}^T = \boldsymbol{0}$. It splits into

$$\boldsymbol{A}s^T + \boldsymbol{B}p_1^T + \boldsymbol{T}p_2^T = 0 \tag{3.9}$$

$$\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A} + \boldsymbol{C}\right) s^T + \left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B} + \boldsymbol{D}\right) p^T = 0 \tag{3.10}$$

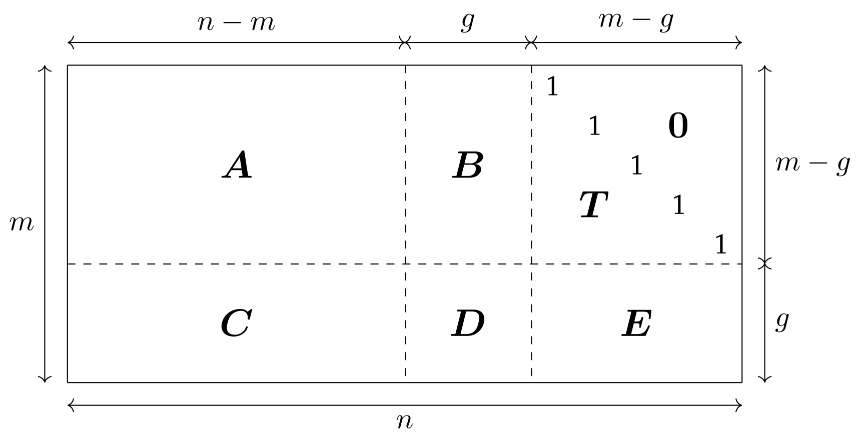

Figure 3.3.: Structure of a matrix in approximate lower triangular form.

Operation	Type
$\boldsymbol{A}s^T$	sparse multiplication
$\boldsymbol{T}^{-1}\boldsymbol{A}s^T$	sparse back substitution
$-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T$	sparse multiplication
$\boldsymbol{C}s^T$	sparse multiplication
$\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T\right)+\left(\boldsymbol{C}s^T\right)$	vector addition
$\boldsymbol{\phi}^{-1}\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}s^T+\boldsymbol{C}s^T\right)$	dense $g\times g$ multiplication

Table 3.1.: Calculations for $p_1^T=\boldsymbol{\phi}^{-1}\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}+\boldsymbol{C}\right)s^T$

. The resulting equations for p_1 and p_2 are

$$p_1^T = -\boldsymbol{\phi}^{-1}\left(-\boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}+\boldsymbol{C}\right)s^T \tag{3.11}$$
$$p_2^T = -\boldsymbol{T}^{-1}\left(\boldsymbol{A}s^T+\boldsymbol{B}p_1^T\right) \tag{3.12}$$

. The complexity of the computations can be reduced by computing $\boldsymbol{\phi}^{-1}$ offline. Offline meaning that it is precomputed and when encoding multiplying by the matrix. All the other matrix multiplications from equations (3.11) and (3.12) are done separately. Multiplications by $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{E}$ are sparse and the resulting complexity for these is $\mathcal{O}(n)$. The multiplication with $\boldsymbol{T}^{-1}$ is replaced by the system $x^T=\boldsymbol{T}y^T$. As $\boldsymbol{T}$ is a sparse lower triangular matrix the system can be solved by back substitution in $\mathcal{O}(n)$. The only part with higher complexity is the multiplication with the dense $g\times g$ matrix $\boldsymbol{\phi}$ where the complexity is $\mathcal{O}(g^2)$.

do i want do describe the algorithm to get into alt form? yes!

Operation	Type
$\boldsymbol{A}s^T$	sparse multiplication
$\boldsymbol{B}p_1^T$	sparse multiplication
$\left(\boldsymbol{A}s^T\right) + \left(\boldsymbol{B}p_1^T\right)$	vector addition
$-\boldsymbol{T}^{-1}\left(\boldsymbol{A}s^T + \boldsymbol{B}p_1^T\right)$	sparse back substitution

Table 3.2.: Calculations for $p_2^T = -\boldsymbol{T}^{-1}\left(\boldsymbol{A}s^T + \boldsymbol{B}p_1^T\right)$

3.1.3. Decoding

Decoding LDPC codes is a nontrivial problem, in fact maximum likelihood decoding is computationally infeasible. Therefore other decoding methods were developed. There are different decoding algorithms which differ in performance and complexity. I will start with a simple algorithm to introduce the required concepts and then go on to more performant ones.

Hard Descision Decoding

This algorithm works on binary input data and all the messages are also binary.

decode dat

4. Channel Models

As the encoded message is passed through a channel I will introduce some basic channel models in this chapter. I will only work with binary codes throughout this thesis. For binary codes it is convenient to use $\{+1, -1\}$ as the alphabet. This simplifies the calculations of LLRs likewise it makes the bit energy E_b simple. Also the channel is memoryless, this may sound confusing at first when dealing with storage devices, but it says that each symbol is independently mangled by the channel. The input to the channel is in the input alphabet $\mathcal{X} = \{1, -1\}$. Whereas the the output alphabet $\mathcal{Y}$ will change depending on the channel. Now when transmitting a codeword $c \in \mathcal{X}^n$ the channel outputs $y \in \mathcal{Y}^n$ and it is the receivers task to compute the original codeword c from y. Ideally this is done with few errors and close to the channel capacity.

better word

4.1. Binary Erasure Channel (BEC)

The binary erasure channel is characterized with a single parameter $0 \leq \alpha < 1$ the erasure probability. The symbol for an erasure is $?$, therefore the output alphabet is $\mathcal{Y} = \{1, -1, ?\}$. The channel outputs x with probability $1 - \alpha$ and $?$ with probability α. figure 4.1 shows the transitions for a BEC. So for a codeword with large length n there will be $(1 - \alpha)n$ correct symbols, this suggests that the maximum rate is $1 - \alpha$. Elias[1] shows that this rate can be archived and also proves that this is the capacity.

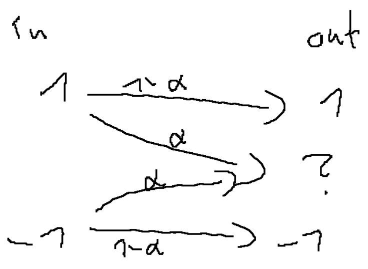

Figure 4.1.: Symmetric binary erasure channel

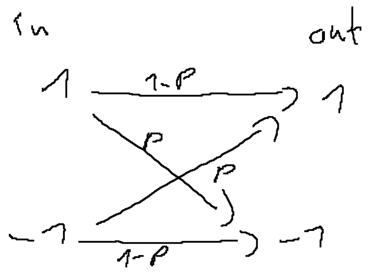

Figure 4.2.: Symmetric binary symmetric channel

4.2. BSC

A binary symmetric channel has the output alphabet $\mathcal{Y} = \{1, -1\}$. An incoming symbol has the probability p to create a crossover. With probability $1 - p$ the symbol is correctly transmitted and with probability p it is flipped. In figure 4.2 a graph shows these probabilities. A binary symmetric channel with crossover probability p has the capacity $1 - H(p)$ with $H(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$ being the binary entropy function.

4.3. Additive White Gaussian Noise Channel

The additive white gaussian noise (AWGN) channel is the most important noise model for me as it characterizes flash memory well. It has a continuos output alphabet $\mathcal{Y}$. The model for the channel is $y = x+z$ where the input is $x \in \mathcal{X}$. z is a variable with normal distribution with 0 mean and variance σ^2. It has the distribution $f(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{z^2}{2\sigma^2})$. The capacity for this channel is $\frac{1}{2} \log_2(1 + \frac{1}{\sigma^2})$ as the Shannon limit shows. Often it is preferable to allow scaling of the input, so we use a signal to noise ratio E_b/σ^2. This allows the inputs to be arbitrary values then E_b is the energy of the transmission of a single bit. The σ^2 is frequently called N_0, the energy of the noise that is added in a single bit transmission. When using E_b/N_0 the capacity is $\frac{1}{2} \log_2(1 + \frac{E_b}{N_0})$. For example when having a rate of $\frac{1}{2}$ we get a minimum signal to noise ratio required of 1.

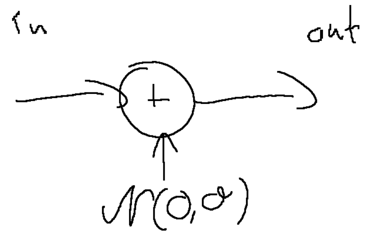

Figure 4.3.: Additive White Gaussian Noise Channel

5. Field Programmable Gate Array (FPGA)

In modern hardware design it is advantageous to have reprogrammable hardware elements. To create reconfigurable logic circuits basic elements consisting of lookup tables (LUT) and registers is built into an array and connected by programmable interconnects.

write some basics about FPGA

6. Flash Memory

Flash memory often in the form of solid state drives (SSD) are becoming more and more of primary storage for computers. The cost per storage has been decreasing steadily therefore driving adoption. Especially the low access latency and the high possible throughput compared so spinning disk hard drives are an advantage. In the application predominantly NAND type flash is used due to its higher density.

6.1. Flash Basics

NAND flash which will be discussed is of most interest for SSDs due to its high density and therefore lower cost per bit. figure 6.1 shows a floating gate transistor which is used to store data. By inserting charge into the floating gate it is possible to change the threshold voltage of the transistor.

6.2. Error Types

[10]

make image of floating gate flash transistor

Figure 6.1.: A floating gate field effect transistor

make image of nand string

Figure 6.2.: Multiple transistors are arranged to make a NAND string

7. Approach

My architecture is somewhat based on Yanhuan Liu, Chun Zhang, Pengcheng Song, and Hanjun Jiang[5] paper. Although I change the stored values. For a message parsing LDPC decoder the straightforward implementation is to store the messages sent between the check and parity nodes. This results in needing to store multiple values per row of the parity check matrix. For example the LDPC code used for 802.11n with block length 1926 and rate 0.5 has row weights of 7 and 8. This requires to store 8 messages per row of the parity check matrix. The approach I chose is only suited to the min-sum algorithm and will result in a reduction of storage requirements. Instead of splitting the iteration at the message step it is in this case preferable to split at the minimum and the sum for the variable node calculation.

find nice source for 802.11n codes

For decoding the are two message types. The message from the variable nodes to the check nodes q_{nm} and the message going the other way r_{nm}. Decoding is done in different steps[2]:

1. Initialization
   The values from the channel are converted to LLRs y_n. These initial LLR values used as q_{nm} the input to the first check node.

2. Check Node Step
   Each check node m receives the messages from the variable nodes and calculates its response message.

$$r_{mn} = \left(\prod_{n' \in M(m)n} \operatorname{sign}(q_{n'm}) \right) \min_{n' \in vn\{m\}n} (|q_{n'm}|) \tag{7.1}$$

3. Variable node step
   Each variable node n receives the messages from the check nodes and calculates its response message.

$$q_{nm} = y_n + \sum_{m' \in N(n)m} r_{m'n} \tag{7.2}$$

4. Output Descision
   The result LLR values are updated.

$$L_n = y_n + \sum_{m \in N(n)} r_{m'n} \tag{7.3}$$

Instead of storing all the messages it is also possible to store the sign, the minimum, and the sum over all messages directed to a column of variable nodes. When storing the minimum it is not enough to just store the minimum. This arises due to the fact that each minimum calculation excludes the current check node. Therefore I store the smallest, the second smallest, and the position of the smallest number. The notation $\min^2$ is for the smallest argument but not including the element min returns. If I call the minimum s, the second smallest t, the product of all signs v, and the id k, I get the check node step split into two:

$$s(m) = \min_{n' \in M(m)} (|q_{n'm}|) \tag{7.4}$$

$$t(m) = \overset{2}{\min_{n' \in M(m)}} (|q_{n'm}|) \tag{7.5}$$

$$k(m) = \arg\min(|q_{n'm}|) \tag{7.6}$$

$$v(m) = \prod_{n' \in M(m)} \operatorname{sign}(q_{n'm}) \tag{7.7}$$

And for the check node calculation I can use the results from equations (7.4) to (7.7) to simplify the calculations for each check node.

$$r_{mn} = v(m)\operatorname{sign}(q_{nm})c_{mn} \tag{7.8}$$

with

$$c_{mn} = \begin{cases} t(m), & \text{for } n = k(m) \\ s(m), & \text{else} \end{cases} \tag{7.9}$$

In the variable node step I instead calculate the sum over all messages:

$$S(n) = y_n + \sum_{m \in N(n)} r_{m'n} \tag{7.10}$$

8. Implementation

9. Results

A. Appendix

Bibliography

[1] P. Elias. In: (1955).

[2] A. A. Emran and M. Elsabrouty. “Simplified variable-scaled min sum LDPC decoder for irregular LDPC codes”. In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. 2014, pp. 518–523. DOI: `10.1109/CCNC.2014.6940497`.

[3] M. P. C. Fossorier. “Quasicyclic low-density parity-check codes from circulant permutation matrices”. In: 50.8 (2004), pp. 1788–1793. DOI: `10.1109/TIT.2004.831841`.

[4] Robert R. Gallager. “Low-Density Parity-Check Codes”. In: (1963).

[5] Y. Liu et al. “A high-performance FPGA-based LDPC decoder for solid-state drives”. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2017, pp. 1232–1235. DOI: `10.1109/MWSCAS.2017.8053152`.

[6] Hanghang Qi and Norbert Goertz. “Low-Complexity Encoding of LDPC Codes: A New Algorithm and its Performance”. In: ().

[7] Thomas J. Richardson and Rüdiger L. Urbanke. *Efficient Encoding of Low-Density Parity-Check Codes*. 2001. DOI: `10.1109/18.910579`.

[8] Bashar Tahir, Stefan Schwarz, and Markus Rupp. “BER comparison between Convolutional, Turbo, LDPC, and Polar codes”. In: (2017). DOI: `10.1109/ICT.2017.7998249`.

[9] M. Tanner. “A recursive approach to low complexity codes”. In: (1981). DOI: `10.1109/TIT.1981.1056404`.

[10] S. A. A. Zaidi et al. “FPGA Accelerator of Algebraic Quasi Cyclic LDPC Codes for lt;sc gt;nand lt;/sc gt; Flash Memories”. In: *IEEE Design Test* 33.6 (2016), pp. 77–84. DOI: `10.1109/MDAT.2015.2497322`.