Nemanja Stamenić, BSc

Efficient Implementation of a DVB-S2X-Compatible LDPC Codec Using the AVX Instruction Set

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Electrical Engineering

Submitted to

Graz University of Technology

Supervisor

Assoc. Prof. Dipl.-Ing. Dr. techn. Wilfried Gappmair
Institute of Communication Networks and Satellite Communications
Graz University of Technology, Austria

Dipl.-Ing. Dr. techn. Johannes Ebert
Joanneum Research, Graz, Austria

Graz, September 2015

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Acknowledgment

I would like to express my gratitude to my supervisor Assoc. Prof. Dipl.-Ing. Dr. techn. Wilfried Gappmair for the useful comments, remarks and engagement through the learning process of this master thesis.

I am deeply indebted to my second supervisor Dipl.-Ing. Dr. techn. Johannes Ebert who had the idea for the topic of this thesis and whose stimulating motivation and valuable suggestions were crucial for the completion of this work.

Thanks to my parents Gordana and Milan and brother Stefan for their love and unconditional support in hard times. Thanks to my father Milan for inspiring me to pursue a career in engineering.

Special thanks to Ivana, who was always standing by me, for her precious love.

Abstract

LDPC codes were discovered by Gallager in 1963. They were neglected and forgotten due to the inability of the processing units at that time to satisfactorily perform calculations needed for decoding this sort of channel codes. LDPC codes were rediscovered by MacKay in 1999, who showed that LDPC codes are able to compete with turbo codes. In 2001, Richardson and Urbanke demonstrated that LDPC codes perform better than turbo codes in the case of long frame lengths. A very significant advantage of LDPC codes over turbo codes is the fact that they are computationally less demanding.

Excellent performance of LDPC codes for long frame lengths makes them suitable for television broadcasting. As such they form part of the DVB-S2, DVB-S2X, DVB-T2 and DVB-C2 standards. Time efficient implementation of DVB-S2 and DVB-S2X compatible LDPC codec for PC simulation purposes is the main topic of the current thesis.

The data rates of software-based LDPC codecs can be significantly increased through the use of single-instruction-multiple-data (SIMD) operations, because they enable parallel decoding of multiple frames. Encoder and encoder presented in this thesis use the Intel® Advanced Vector Extensions instruction set, a SIMD extension to the x86 instruction set. They are able to encode and decode 32 DVB frames in parallel. The decoder implementing 20 iterations of the min-sum algorithm achieves up to 50 Mbit/s data rate on the fourth generation Intel i-7 processors using single CPU core. The encoder uses approximate linear triangulation and achieves approximately 300 Mbit/s using the same hardware. If the simulation framework is run in parallel on all four CPU cores, data rates of up to 150 Mbit/s (decoder) and 900 Mbit/s (encoder) are achieved.

Zusammenfassung

LDPC-Codes wurden 1963 von Gallager entdeckt. Aufgrund der Tatsache, dass Prozessoren zu diesem Zeitpunkt die notwendigen Berechnungen nicht effizient genug durchführen konnten, gerieten diese Codes wieder in Vergessenheit. Erst 1999 wurden sie von MacKay wieder entdeckt, der zeigen konnte, dass sie eine echte Alternative zu den sogenannten Turbo-Codes darstellen. Im Jahr 2001 gelang Richardson und Urbanke dann der Nachweis, dass LDPC-Codes bei großen Blocklängen eindeutig besser sind als die Turbo-Codes. Ein entscheidender Vorteil gegenüber letzteren ist, dass sie weniger Rechenleistung benötigen.

Die Leistungsfähigkeit von LDPC-Codes bei großen Blocklängen macht sie besonders geeignet für alle Anwendungen, wo digitales Fernsehen eine Rolle spielt. Mittlerweile sind sie Bestandteil verschiedener Standards, so etwa bei DVB-S2, DVB-S2X, DVB-T2 oder DVB-C2. Dementsprechend ist das Hauptthema der vorliegenden Masterarbeit die effiziente Implementation von LDPC-Codecs, welche im DVB-S2- bzw. DVB-S2X-Standard empfohlen werden.

Die Datenrate von Software-basierenden LDPC-Codes kann erheblich gesteigert werden, wenn man auf Single-Instruction-Multiple-Data (SIMD) Operationen zurückgreifen kann, weil damit eine parallele Verarbeitung von Daten möglich ist. Zu diesem Zweck wird der Intel® Advanced Vector Extensions Befehlssatz verwendet, der eine SIMD-Erweiterung des x86 Befehlssatzes darstellt. Damit ist die parallele Verarbeitung von 32 DVB-Rahmen möglich. Wird zur Decodierung der Min-Sum-Algorithmus mit 20 Iterationen verwendet, dann erreicht man auf dieses Weise einen Durchsatz von bis zu 50 Mbit/s, wenn die Software auf einem Intel i-7 Prozessor der 4. Generation mit Single-CPU-Core läuft. Der Encoder mit näherungsweise linearer Triangulation schafft einen Durchsatz von etwa 300 Mbit/s auf derselben Hardware. Falls die Simulation parallel auf allen vier CPU-Cores gestartet wird, dann ist eine Rate von 150 Mbit/s (Decoder) bzw. 900 Mbit/s (Encoder) denkbar.

Table of Contents

References

[RD1] B. Sklar, “Digital Communications – Fundamentals and Applications”, Prentice Hall, 2nd Edition, 2001

[RD2] R. H. Morelos-Zaragoza, “The Art of Error Correcting Coding”, Wiley, 2nd Edition, 2006

[RD3] C. Marchand, “Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards”, Master Thesis, Université de Bretagne Sud, France, 2010

[RD4] L. Hanzo, T. H. Liew, and B. L. Yeap, “Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels”, Wiley, 2002

[RD5] N. Stamenic, “Efficient Turbo Code Implementation and Simulation for an AWGN Channel”, Bachelor Thesis, Graz University of Technology, Austria, 2014

[RD6] E. Voges, “Hochfrequenztechnik: Bauelemente, Schaltungen, Anwendungen”, 3. Auflage, Hüthig, 2004

[RD7] G. L. Stüber, “Principles of Mobile Communication”, Kluwer Academic Publishers, 2002

[RD8] A. F. Molisch, “Wireless Communications”, Wiley, 2nd Edition, 2011

[RD9] B. Friedrichs, “Kanalcodierung: Grundlagen und Anwendungen in modernen Kommunikationssystemen”, Springer, 1995

[RD10] C. E. Shannon, “Mathematical Theory of Communication”, BJST, 1948

[RD11] W. Riedler, G. Petter, and W. Weselak, “Informationstheorie und Codierung”, Vorlesungsskriptum, Graz University of Technology, Austria, 2000

[RD12] M. C. Valenti, “Iterative detection and decoding for wireless communications,” Ph.D. Thesis, Bradley Dept. Elect. & Comp. Eng., Virginia Tech, 1999

[RD13] R. Gallager, “Low Density Parity Check Codes”, Cambridge, 1963

[RD14] D. MacKay, ‘’Good Error-Correcting Codes Based on Very Sparse Matrices”, IEEE Transactions on Information Theory, vol. 45, no. 2, 1999

[RD15] T. Richardson and R. Urbanke, “The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding”, IEEE Transactions on

Information Theory, vol. 47, no. 2, 2001

[RD16] H. Qi and N. Goertz, “Low-Complexity Encoding of LDPC Codes: A New Algorithm and its Performance”, The University of Edinburgh, available at: http://publik.tuwien.ac.at/files/PubDat_166941.pdf

[RD17] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, “Practical erasure resilient codes”, in Proc. 29th Annu. ACM Symp. Theory of Computing (STOC, pp. 150-159), 1997

[RD18] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, “LDPC block and convolutional codes based on circulant matrices”, IEEE Transactions on Information Theory, vol. 50, no. 12, pp. 2966-2984, Dec. 2004.

[RD19] P. H. Siegel, “An Introduction to Low-Density Parity-Check Codes”, University of California, San Diego, available at: cmrrstar.ucsd.edu/psiegel/pubs/07/ldpc_tutorial.ppt

[RD20] R. Shedsale, “A Review of Construction Methods for Regular LDPC Codes”, Indian Journal of Computer Science and Engineering, vol. 3, 2012

[RD21] ETSI EN 302 307, European Standard (Telecommunications Series)

[RD22] ETSI EN 302 307-2, Draft Version

[RD23] J. Fan, Y. Xiao, and K. Kim, “Design of LDPC Codes without Cycles of Length 4 and 6“, Journal of Electrical and Computer Engineering - Research Letters in Communications, Article ID 354137, 2008

[RD24] A. Balatsoukas-Stimming, ”Construction of LDPC codes“, Telecommunications Laboratory, Technical University of Crete, Greece, 2009

[RD25] T. J. Richardson and R. L. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes”, Transactions on Information Theory, vol. 47, no. 2, February 2001

[RD26] F. A. Newagy, Y. A. Fahmy, and M. M. S. El-Soudani, “Novel construction of short length LDPC codes for simple decoding”, Journal of Theoretical and Applied Information Technology, 2007

[RD27] Introduction to Intel Advanced Vector Extensions, available at: https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

[RD28] X. Wu, “Adaptive-Normalized/Offset Min-Sum Algorithm”, IEEE Communications Letters, vol. 14, no. 7, 2010

[RD29] Y. Jung, Y. Jung, S. Lee and J. Kim, “New Min-Sum LDPC Decoding Algorithm Using SNR-Considered Adaptive Scaling Factors“, ETRI Journal, 2014

[RD30] D. J. C. MacKay, ”Information Theory, Inference and Learning Algorithms”, Cambridge University Press, 2003

[RD31] G. Falcao, M. Gomes, V. Silva, L. Sousa, and J. Cacheira, “Configurable M-factor VLSI DVB-S2 LDPC decoder architecture with optimized memory tiling design”, EURASIP Journal on Wireless Communications and Networking, pp. 1-16, 2012

[RD32] X. Zhang and P. H. Siegel, “Quantized Min-Sun Decoders with Low Error floor for LDPC Codes”, In Proc. IEEE International Symposium on Information Theory , 2012

Acronyms

ACM	Adaptive coding and modulation
AGC	Automatic gain control
ALT	Approximate lower triangular form
ARQ	Automatic repeat request
AVX	Advanced Vector Extensions
BPSK	Binary phase shift keying
CCM	Constant coding and modulation
CENELEC	European Committee for Electrotechnical Standardization
CN	Check node
DVB-C	Digital Video Broadcasting – Cable
DVB-S	Digital Video Broadcasting – Satellite
DVB-T	Digital Video Broadcasting – Terrestrial
EBU	European Broadcasting Union
ETSI	European Telecommunications Standards Institute
FEC	Forward error correction
FER	Frame error rate
FSM	Finite-state machine
FSPL	Free space loss
LDPC	Low density parity check
LLR	Log-likelihood ratio
LR	Likelihood ratio
MAP	Maximum a posteriori
ML	Maximum likelihood
QEF	Quasi error free
RF	Radio frequency
RSC code	Recursive systematic convolutional code
SALT	Systematic approximately triangular form
SIMD	Single Instruction Multiple Data
SISO	Soft-input soft-output
SNR	Signal-to-noise-ratio
VA	Viterbi algorithm
VCM	Variable coding and modulation
VN	Variable node

1. Introduction

1.1. Digital Communication

Fast and reliable communication systems have become indispensable in today's world. The demand for mobile wireless and/or satellite communication links, which use free space as a propagation medium, is steadily increasing. Most of the systems providing such services are digital. In [RD1] a brief comparison of digital and analog systems is presented. The most important advantage of digital systems is that signals are much easier regenerated. On its path from the transmitter to the receiver the signal is affected by a series of impairments, which can and typically do change its waveform. The ability to restore the original signal is crucial for a successful communication. Digital signals are two-state signals, unlike analog signals which can take an infinite number of states. As a consequence, digital circuits are less sensitive to distortion and interference and can prevent the accumulation of disturbances. Another important advantage of digital systems is the possibility to significantly reduce error rates by use of forward error detection and correction techniques (FEC). In terms of transmission, different types of digital signals can be treated identically as they are all represented by bits, which is not the case with analog signals. A large number of terminals nowadays are digital (computers, mobile phones, measurement devices etc.) and digital communication links are best suited for the communication between them.

But there are also some disadvantages of digital systems. From the computational point of view, they can be very demanding. The synchronization is more complex and requires much more resources than it is the case with analog systems. Non-graceful degradation [RD1] is another drawback of digital systems compared to analog ones. Quality of service in digital systems tends to drastically change when the signal-to-noise-ratio (SNR) falls under a certain threshold, whereas analog systems have more graceful degradation.

Major goal of a digital communication system is to transfer data from source to destination by minimizing bit error rate and transmitting power, whereas the useful information throughput is maximized. The following figure shows the simplified functional diagram of a digital communication system.

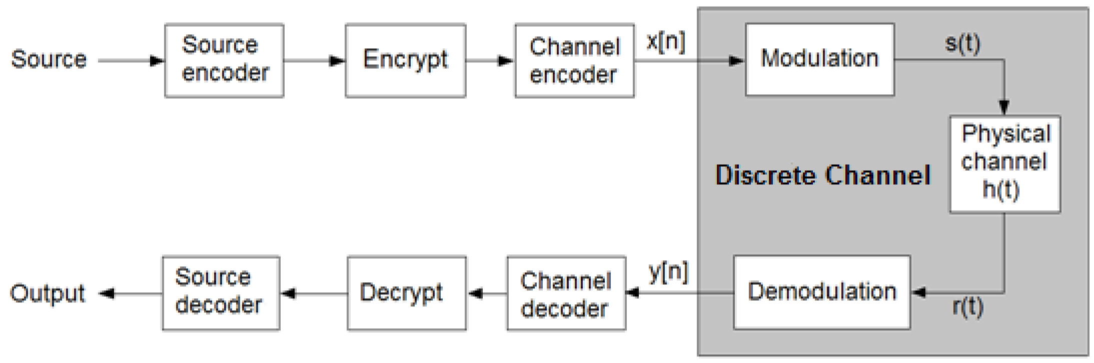


Figure 1: Basic blocks of a digital communication system

1.2. Physical Channel

In wireless communication systems, the signal is determined by a variety of impairments. Weather conditions, obstacles on the communication path, relative motion between the receiver and the transmitter are some of the reasons for such impairments [RD4].

1.2.1. Free Space Loss

Even in case of no atmospheric losses, the signal degrades. This phenomenon is called *free-space path loss* (FSPL) [RD1]. The degree of the signal degradation depends on the carrier frequency and the distance between the transmitter and the receiver [RD1]. In the equation bellow *f*, *d* and *c* stand for carrier frequency, distance and the speed of light, respectively:

$$\mathrm{FSPL} = \left(\frac{4\pi f d}{c}\right)^2 \tag{1}$$

1.2.2. Thermal Noise

Random impairments, which can be described only statistically, are called *noise*. There are many different sources of noise. Some of them are natural and some of them are manmade. Much of the noise effects can be eliminated by good system design. However, the noise generated by random thermal motion of the charge carriers cannot be eliminated [RD1]. It is white, meaning that is has approximately constant power spectral density, and the amplitude is normally distributed with zero mean. The common model for this kind of impairment *is additive white Gaussian noise* (AWGN) and it is the only source of noise that will be considered for the channel model in this thesis. Equation (2) shows the probability density function (PDF) for AWGN:

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2 \right] \tag{2}$$

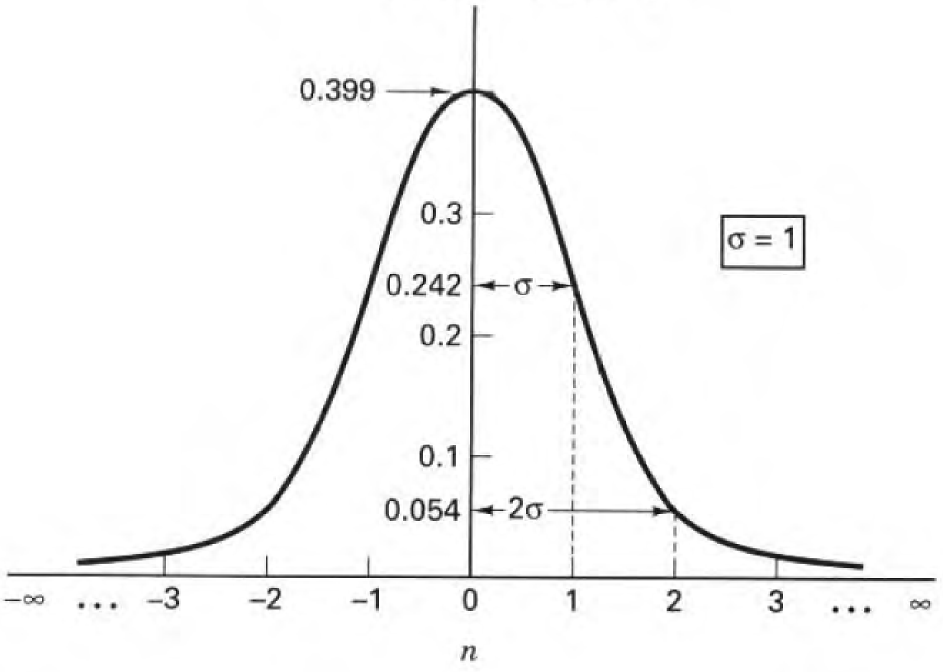


Figure 2: PDF for Gaussian noise with variance $\sigma^2 = 1$

The two following figures depict an AWGN signal in time and frequency domain, respectively.

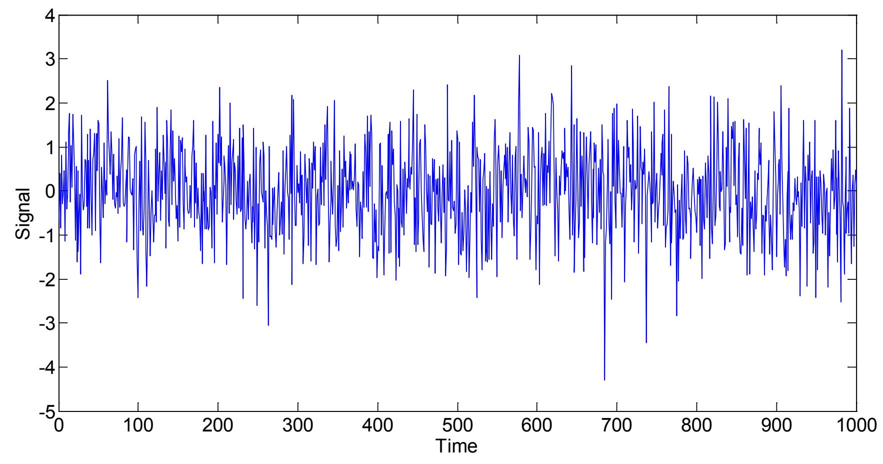


Figure 3: Gaussian noise with variance $\sigma^2 = 1$

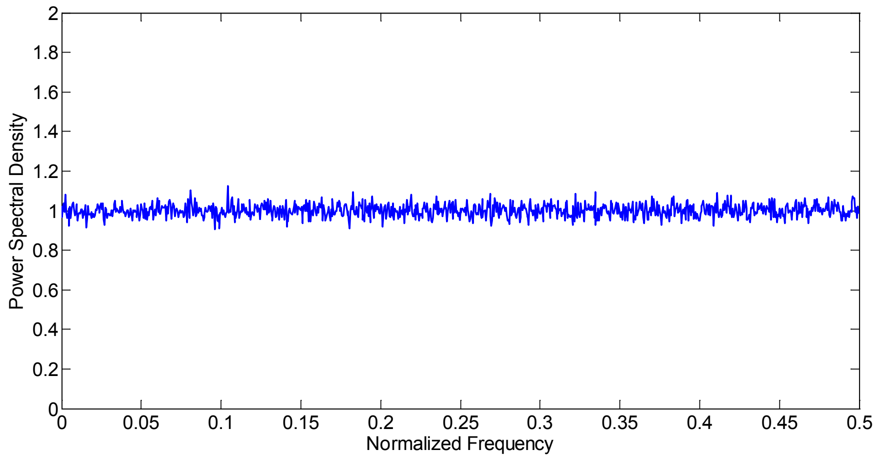


Figure 4: One-sided power spectral density of AWGN

1.2.3. Multipath Propagation and Fading

On their way from the transmitter to the receiver, electromagnetic waves can get reflected, diffracted and scattered [RD4].

Reflection occurs at the boundary between two dielectric media, i.e. when the waves hit the obstacle with smooth surface and dimensions much larger than their wavelength. One part of the wave is transmitted and the other one is reflected, except in the case of total reflection in which no transmission occurs [RD1].

Diffraction occurs when the line of sight (LOS) between transmitter and receiver is obstructed by an impenetrable object larger than the wavelength. Secondary waves are created behind the object and they can still reach the receiver even though it is shadowed by an obstacle, which is why this phenomenon is also called shadowing [RD1].

The causes of scattering are non-uniformities in propagation medium much smaller than the wavelength. Large objects with rough surfaces or small objects relative to the wavelength are typical examples. When an electromagnetic wave gets scattered, it means that it is reflected in all directions.

Reflection, diffraction and scattering are the reasons why electromagnetic waves typically do not travel from the transmitter to the receiver over a single path, but via multiple paths. This phenomenon is called multipath propagation.

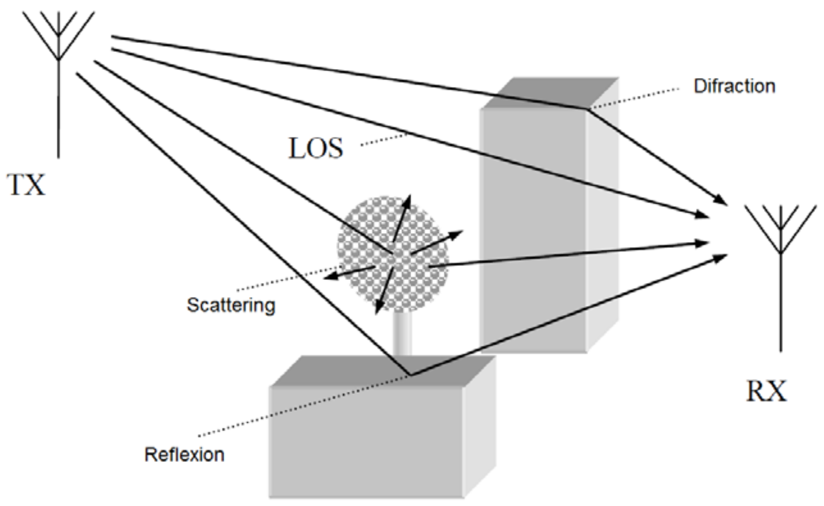


Figure 5: Multipath propagation

The decrease of the received signal power that is not caused by the increase of the transmission distance is called *fading*. There are two types of fading: *large-scale fading* and *small-scale fading* [RD1]. Large-scale fading is the average signal power attenuation due to the motion over large areas [RD1]. It is affected by the weather and by the terrain contours between the transmitter and the receiver, e.g. the receiver is shadowed by a building or a hill. The effects of the large-scale-fading can be compensated with the use of automatic gain control (AGC) [RD6].

Small-scale fading refers to the significant changes of the signal power, phase or the angle of arrival caused by small changes (order of the wavelength) in spatial positioning between the transmitter and the receiver [RD1]. Multipath propagation is one of the causes of the small-scale fading, because of the interference between the signal components which reached the receiver through different propagation paths. This interference can be constructive and results in the amplification of the signal or destructive, which causes signal attenuation. Another cause of the small scale fading is the relative motion between the transmitter and the receiver resulting in Doppler effects [RD7].

We can also classify fading depending on how fast the impulse response of the channel changes. If it significantly changes during one symbol period, we have *fast fading*. If the changes don't occur that fast, we speak of *slow fading*. In some cases there are identical fading effects over the whole bandwidth (*flat fading)*, otherwise we talk about *frequency selective fading* [RD8].

Fading channels are typically modeled by using a *Rice distribution* in the case where there is a LOS component or by using a *Rayleigh distribution*, if there is no LOS [RD8].

1.3. Modulation

Time-discrete signals cannot be transmitted over the channel described in the previous chapter, because the channel itself is analog. Instead, the modulator in Figure 1 converts digital data x[n] to analog waveforms s(t). This process is called baseband or digital modulation. The simplest digital modulation is binary modulation in which binary values are mapped onto two different waveforms, e.g. binary phase shift keying (BPSK). BPSK changes the phase of a sinusoidal signal depending on the bit value (π for binary '0' and 0 for binary '1', or vice versa). More advanced modulation schemes map more than one bit to a single analog waveform. A group of *K* bits can be grouped into a symbol, which is mapped to one out of $M = 2^K$ waveforms. This technique is called *M-ary modulation* [RD8].

Additionally, baseband signal undergoes carrier modulation, which places the signal in the optimal frequency range for transmission via radio frequency (RF). Demodulator does the opposite by converting the analog signal r(t) into digital data y[n].

1.4. Channel Model

In this thesis, modulator and demodulator will be considered to be part of the channel. Multipath propagation, Doppler shift as well as all other fading effects are considered to be totally compensated by suitably selected synchronization and estimation techniques, which are also considered to be the part of the channel. These very significant simplifications leave us with the channel (discrete channel from Figure 1) whose input x[n] and output y[n] are both digital. The only impairment that will be considered is AWGN. BPSK modulation scheme is assumed. If a binary sequence $\boldsymbol{x}$ is transmitted, the signal at the receiver's end is given by

$$y_i = s_i + n_i \tag{3}$$

where $s_i = 2x_i - 1$. The Gaussian noise is denoted by n_i, with two-sided power spectral density $N_0/2$ equal to its variance σ^2.

1.5. Source Coding

The goal of source coding is to compress the data provided by the source to the smallest number of bits possible, without any loss of information. Statistical redundancy is identified and eliminated [RD9]. The point is to transmit the same information with fewer bits thus increasing the throughput.

1.6. Encryption

Encryption is coding data in a way so that only authorized parties can access them. It is also used to prevent counterfeiting and forgery of messages. Encrypted messages can be read only, if the matching key is available [RD4].

1.7. Shannon Limit

In [RD10] Shannon showed that there is a theoretical upper limit for the information rate of any communication channel, called Shannon capacity or channel capacity, below which an arbitrarily small error probability is achieved. Using the Shannon-Hartley theorem, the channel capacity *C* can be computed as

$$C = W \log_2(1 + \frac{S}{N}) \tag{4}$$

where S and N denote the average signal and noise power. The latter is proportional to bandwidth *W:*

$$N = N_0 W \tag{5}$$

Combining Equations (4) and (5) yields:

$$C = W \log_2(1 + \frac{S}{N_0 W}) \tag{6}$$

If we assume that the transmission occurs at a rate equal to the channel capacity, we have that [RD1]

$$\frac{S}{N_0 C} = \frac{E_b}{N_0} \tag{7}$$

After a couple of straightforward steps, we obtain

$$\frac{C}{W} = \log_2\left(1 + \frac{E_b}{N_0} \cdot \frac{C}{W}\right) \tag{8}$$

which is equivalent to

$$2^{C/W} = 1 + \frac{E_b}{N_0} \cdot \frac{C}{W} \tag{9}$$

and

$$\frac{E_b}{N_0} = \frac{W}{C}(2^{C/W} - 1) \tag{10}$$

The following figure shows the normalized bandwidth W/C versus E_b/N_0 according to Equation (10).

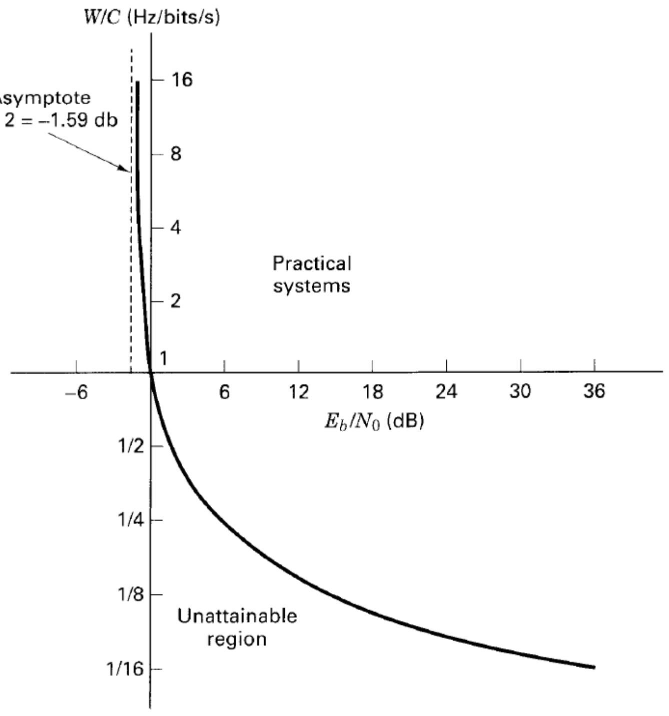


Figure 6: Normalized channel bandwidth as a function of E_b/N_0 [RD1]

The curve from the Figure 6 shows the asymptotical behavior for $W/C \to \infty$. This means that error-free transmission is not possible if E_b/N_0 falls below a certain value. This value can be calculated from Equation (10):

$$\frac{E_b}{N_0} = -1.59 \text{ dB} \tag{11}$$

is called the *Shannon limit* [RD1]. This limit cannot be reached, because the bandwidth requirements would increase without bound.

1.7.1. Channel Coding

Binary modulation schemes alone operate far from the Shannon limit. The goal of channel coding is to improve the performance of the communication system, i.e. to facilitate the system operation as close to the Shannon limit as possible by adding redundancy at the transmitter side, which is then used at the receiver to detect and possibly correct errors.

There are two different groups of techniques used for channel coding. Automatic repeat request (ARQ) is a strategy that solely detects errors without being able to correct them at the receiver side. If a transmission error is detected, the retransmission request is sent by the receiver and the same message is transmitted again. ARQ techniques are suitable for data applications, but they might be problematic for delay-critical communication, like voice applications, because the information flow is influenced by the channel conditions [RD7]. Also, the repeated transmission of the same messages increases the energy consumption. Forward error correction (FEC) techniques are able to correct the detected errors, which means that the state of the communication channel has an impact on the error rate, but it does not influence the information flow. Using FEC, larger number of errors can be tolerated. This means that FEC-coded systems can operate with lower transmit power, transmit over longer distances, use smaller antennas and/or transmit at higher data rates [RD4]. ARQ is not the topic of this thesis and will not be considered further.

As mentioned, a FEC encoder adds redundancy to the input sequence. An input of an FEC encoder is k bits long and the output is n bits long, where $n > k$. The output of an FEC encoder is also called a code word. Although there are 2^n possible sequences using n bits, there are only 2^k valid code words [RD11]. The amount of added redundancy is typically described by the code rate:

$$CR = \frac{k}{n} \tag{12}$$

The choice of the code rate influences both energy and bandwidth efficiency of the system. A trade-off between the two has to be made. Higher code rates result in the smaller number of errors, thus improving the energy efficiency of the system. However, a higher code rate decreases the bandwidth efficiency of the system, because it implies larger number of parity bits for the same number of information bits [RD4].

There are linear and non-linear FEC codes. If any linear combination of code words is also a code word, we speak of linear codes. Channel errors can transform one valid code word into another valid code word. In that case, error detection and correction is impossible. This means that the maximum number of errors that can be corrected depends on the number of positions at which the binary value is different for any two code words of a code [RD12]. This number is called *minimal Hamming distance* and it is a measure for the minimum number of channel errors that could have transformed one valid code word into another valid code word. The number *t* of errors whose correction can be guaranteed*,* depends on the minimal distance. However, it is possible to correct more than *t* errors in the case of code words with weights higher than the minimal distance [RD30], which is why the *distance distribution A(w)* is often used to characterize a linear code. It is the number of code words with weights equal to *w* [RD30]. The following figure shows the distance distribution of the (7, 4) Hamming code [RD30].

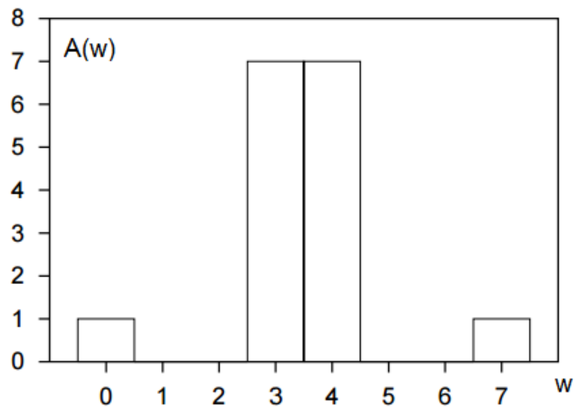


Figure 7: Distance distribution of the (7, 4) Hamming code

Linear codes are divided into convolutional codes and block codes. The following figure shows the typical classification of forward error correcting codes.

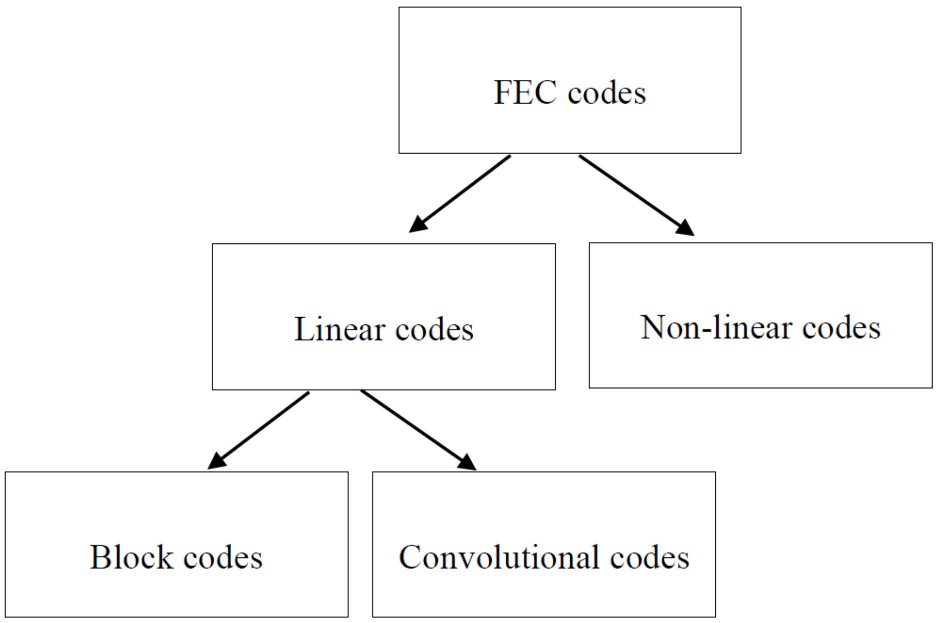


Figure 8: Classification of FEC codes

1.7.1.1. Block Codes

Block codes operate on distinct data frames or blocks. They do not process data continuously. An input of a binary block encoder is a k-bit long vector $\mathbf{u} = (u_1, u_2, \dots, u_k)$ and its output is a length-n codeword $\mathbf{c} = (c_1, c_2, \dots, c_n)$, where $u_i \in \{0,1\}$ and $c_i = \{0,1\}$. Code words are generated through the linear mapping [RD7]:

$$\mathbf{c} = \mathbf{u}\,\mathbf{G} \tag{13}$$

with **G** denoting a $k \times n$ matrix with full row rank *k*, called the *generator matrix*. The design of the generator matrix is crucial for the performance and decoding complexity of block codes.

For every generator matrix **G**, there is a $(n-k) \times n$ parity check matrix **H** with full row rank $n-k$ such that [RD7]

$$\mathbf{G}\,\mathbf{H}^{\mathrm{T}} = \mathbf{0}_{k\times(n-k)} \tag{14}$$

For any code word ***c***, we have that

$$\mathbf{c}\,\mathbf{H}^{\mathrm{T}} = \mathbf{0}_{(n-k)} \tag{15}$$

A block code with the generator matrix of the form

$$\mathbf{G} = [\mathbf{I}_{k\times k}|\mathbf{P}] \tag{16}$$

where **I** is the identity matrix and **P** denotes a $k \times (n-k)$ matrix, is called a systematic block code [RD7]. From equations (13) and (16) it can be seen that the first *k* bits of a code word **c** of a systematic block code are equal to the input vector **u**, whereas the other n – k bits are parity check bits.

The parity check matrix of a systematic block code has the following form:

$$\mathbf{H} = \left[\mathbf{I}_{(n-k)\times(n-k)}\middle|\mathbf{P}_{k\times(n-k)}\right] \tag{17}$$

If a code word ***c*** is transmitted, then we have a vector $\mathbf{y} = \mathbf{c} + \mathbf{n}$ at the receiver, where **n** represents the AWGN component. The *syndrome* of **y** is defined as follows [RD7]:

$$\mathbf{s} = \mathbf{y}\,\mathbf{H}^{\mathrm{T}} \tag{18}$$

If the syndrome has a non-zero value, an error must have occurred. However, $\boldsymbol{s} = 0$ does not guarantee that no errors have occurred, because there is also a possibility that due to the large number of channel errors, a valid code word has been transformed into another valid code word. This shows that there is a maximum number of channel errors that a block code can detect which given by $\mathrm{d_{min}} - 1$, where $\mathrm{d_{min}}$ denotes the minimal Hamming distance [RD7]. Also, there is an upper limit on the error correction capability [RD4]:

$$t = \left\lfloor\frac{\mathrm{d_{min}}-1}{2}\right\rfloor \tag{19}$$

1.7.1.2. Convolutional Codes

Unlike block codes, convolutional codes process data continuously. They use binary convolution to encode the whole information sequence, without dividing it into individual information words or blocks. There are two basic types of convolutional codes: non-recursive and recursive convolutional codes [RD7].

Convolutional encoders are typically represented as finite-state-machines (FSM). The way they encode data is determined by the *generator polynomials*. The number of generator polynomials is defined by the desired code rate, as each generator polynomial defines the computation of one parity bit per one information bit. Two generator polynomials will result in the code rate of 1/2, three in the code rate of 1/3 and so on [RD4]. The figure below shows a non-recursive convolutional encoder with a code rate of *CR* = 1/2 and the shift register length *K* = 3 [RD5].

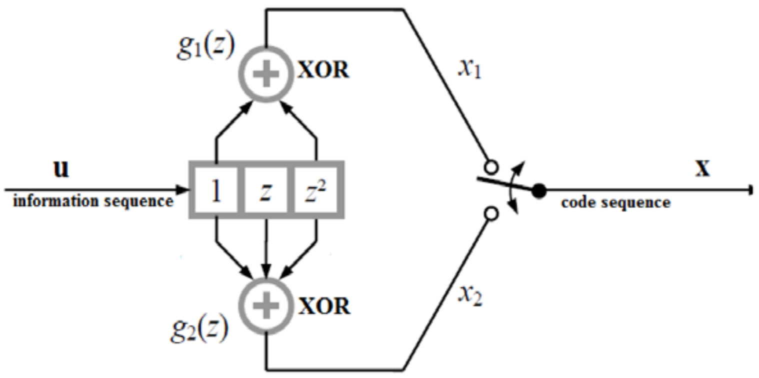


Figure 9: Non-recursive convolutional encoder with *K* = 3*, CR* = 1/2

The generator polynomials that define the encoder from the Figure 9 are represented as follows:

$$g_1(z) = 1 + z^2,\ g_2(z) = 1 + z + z^2 \tag{20}$$

Recursive convolutional encoders implement a feedback loop, which is the reason why they are infinite impulse response filters (IIR), unlike non-recursive encoders that have finite impulse response. The following figure depicts a recursive systematic convolutional (RSC) encoder with the generator polynomials [RD5]:

$$g_1(z) = 1 + z^2,\ g_2(z) = 1 + z + z^2 \tag{21}$$

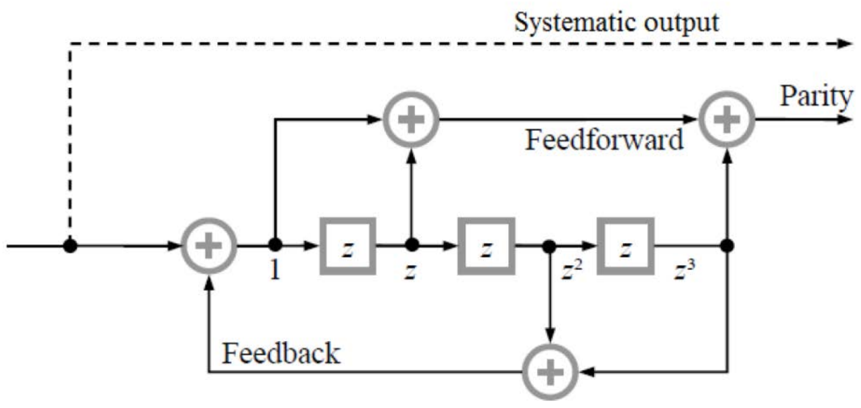


Figure 10: RSC encoder for UMTS

The performance of the RSC codes is better at low SNR and worse at high SNR, compared to non-recursive, non-systematic convolutional codes [RD4].

The common way to describe a convolutional encoder is the *trellis diagram*. It enables us to represent discrete time, the state of the shift registers, the input and the output of the encoder, all on the same graph. Figure 11 shows the trellis diagram for the encoder depicted in the Figure 9.

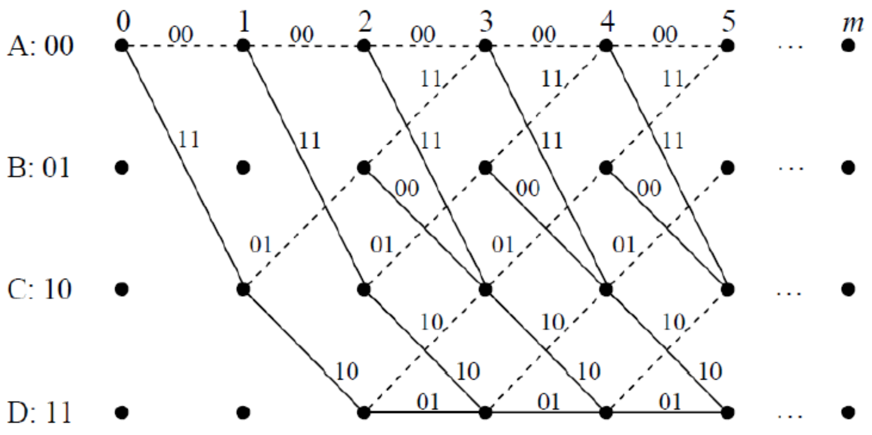


Figure 11: Trellis diagram related to the encoder in Figure 9

Algorithms based on the Viterbi algorithm (VA) or the maximum a-priori (MAP) based algorithms are typically used for decoding of convolutional codes. Both VA and MAP use the trellis structure of the code and are able to significantly reduce the computational complexity of the decoding process, compared to the brute force decoding, in which the conditional probabilities for each possible information sequence are computed before the most probable one is chosen [RD4].

1.7.1.3. Turbo Codes

A turbo code is a parallel combination of two or more constituent convolutional codes. The inputs of the constituent encoders are made statistically independent through *interleaving*. This reduces significantly the probability of both constituent encoders producing low weight outputs [RD12], which is of great importance at small SNRs. However, turbo codes possess relatively low minimal Hamming distance and are thus not able to perform as well at high SNR [RD12]. The following figure depicts a turbo encoder with RSC encoders as a constituent components.

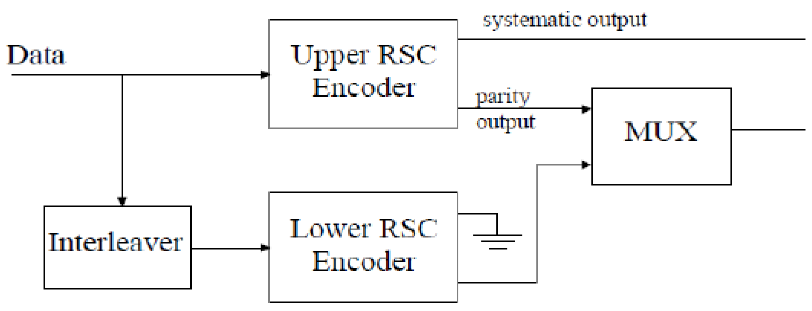


Figure 12: Example for a turbo encoder

Due to its high computational complexity, ML decoding of turbo codes is not used in practical applications. This complex decoding problem is divided into smaller tasks through the introduction of two constituent decoders and iterative decoding. The constituent decoders exchange information after each half-iteration, which greatly improves the performance of turbo codes [RD4]. In order to make the exchange of a posteriori information possible, soft-input soft-output (SISO) constituent decoders have to be used.

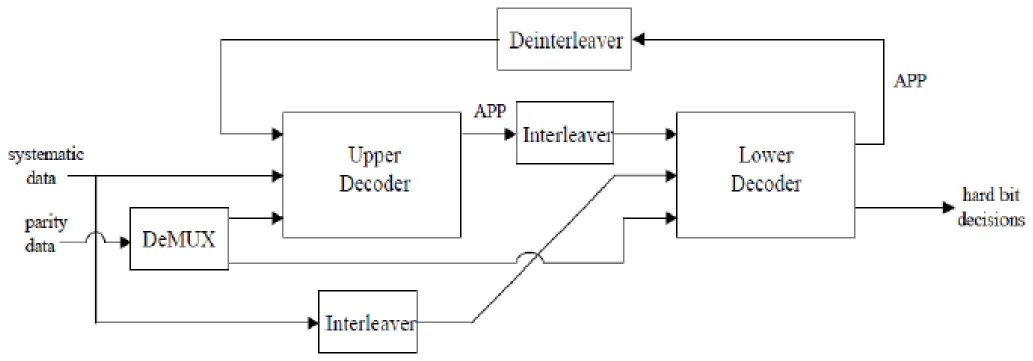


Figure 13: Example for a turbo decoder

The decoding algorithm used by the two constituent decoders has a great influence on the performance and computational complexity of turbo codes. As already mentioned, there are two main groups of decoding algorithms for convolutional codes. One group is based on the Viterbi algorithm, which itself is not used for turbo decoding due to its inability to produce soft-outputs, and the other is based on the MAP algorithm. The decoding algorithms based on the VA minimize the sequence error probability; whereas the MAP based algorithms minimize the bit error probability [RD4]. MAP-based algorithms are in general computationally more complex, but they are able to achieve better performance [RD12].

2. LDPC Codes

2.1. Brief History

Prior to the introduction of turbo codes and their iterative decoding in 1993 by C. Berrou, A. Glavieux, and P. Thitimajshima, the most popular coding scheme was a concatenation of a convolutional code with Viterbi decoding and a Reed-Solomon code. Turbo codes are able to perform significantly better than this combination. They can perform as close as 0.7 dB to the Shannon limit [RD12]. This substantial progress explains the great success and popularity of turbo codes. The application of iterative decoding was so important that it lead to a small revolution in digital communications.

However, there is a coding algorithm based on iterative decoding, called low-density parity check (LDPC), discovered by Gallager [RD13] in 1963, thirty years before the discovery of turbo codes. LDPC codes were neglected and forgotten due to the inability of the processing units at that time to satisfactorily perform calculations needed for decoding this sort of codes. LDPC codes were rediscovered by MacKay [RD14] in 1999, who showed that LDPC codes are able to compete with turbo codes. In 2001, Richardson and Urbanke [RD15] demonstrated that LDPC codes perform better than turbo codes in the case of long frame lengths. A very significant advantage of LDPC codes over turbo codes is the fact that they are computationally less demanding [RD3].

The excellent performance of LDPC codes for long frame lengths makes them suitable for digital video broadcasting (DVB). The first standard recommending LDPC codes for its FEC strategy was the Second Generation of Digital Video Broadcasting via Satellite (DVB-S2) in 2003. LDPC is also used in the two other second generation digital television broadcasting standards: Terrestrial DVB (DVB-T2) and Digital Video Broadcasting via Cable (DVB-C2). The same decoder can be used for all three DVB standards. Since 2008, LDPC codes are used in the home network technology family of standards developed under the International Telecommunication Union (ITU-T

G.hn). Other standards that suggest LDPC codes are the 10GBase-T Ethernet and the Wi-Fi 802.11 standard (optional part of 802.11n and 802.11ac).

In DVB standards, a BCH code is used as an outer code in order to correct few errors that are left by LDPC decoding.

2.2. Representation of LDPC Codes

LDPC codes are a class of linear block codes. They got their name because of the fact that they use sparse or low-density parity check matrices. An $M \times N$ matrix is low density or sparse, if each column and each row is specified by

$$w_c \ll N,\ w_r \ll M \tag{22}$$

where column weight w_c and row weight w_r denote the number of non-zero entries in a column and the number of non-zero entries in a row, respectively. For the parity check matrix of an LDPC code, all non-zero entries are binary 1's. An LDPC code is called regular, if its parity check matrix meets the following condition:

$$w_c = const.,\ w_r = const. \tag{23}$$

It is not practical to give an example of a low-density matrix on a single sheet of paper, as the matrix dimensions have to be very large so that the condition from (22) can be met. In that sense, the following matrix is not sparse, but it can be used as an example for the representation of LDPC codes.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{24}$$

Like with other block codes, every valid code word ***c*** is determined by

$$\mathbf{c}\,\mathbf{H}^{\mathrm{T}} = \mathbf{0} \tag{25}$$

As already mentioned, LDPC codes are able to perform near the Shannon limit, if the block lengths are very high. Very long frames mean very large parity check matrices and a multiplication of a code word with a large matrix can be a very complex task. The computational complexity is proportional to the number of 1's in the parity check matrix, which is why low-density parity check matrices are used.

Each column of the parity check matrix **H** corresponds to one code word bit. Parity check equations are defined by the rows of the parity check matrix **H**, i.e. by their non-zero elements. Each row corresponds to one parity check equation and a non-zero element at the position *(i, j)* means that *j-th* bit contributes to the *i-th* parity check equation. Parity check equations associated with the parity check matrix **H** from (24) have the following form:

$$\begin{aligned} z_1 &= y_1 + y_3 + y_4 + y_8 \\ z_2 &= y_1 + y_2 + y_5 + y_7 \\ z_3 &= y_2 + y_3 + y_6 + y_8 \\ z_4 &= y_4 + y_5 + y_6 + y_7 \end{aligned}$$

A typical way to represent LDPC codes is using a bipartite graph, called *Tanner graph*, whose incident matrix is equal to the parity check matrix of the code. A bipartite graph is a graph whose nodes are divided into two disjoint sets such that there is no branch that connects two nodes belonging to the same set. In the Tanner graph, the nodes are divided into *check nodes* (CN), which represent parity check equations and *variable nodes* (VN), which represent code bits. The following figure depicts the Tanner graph associated with the parity check matrix from (24).

Analog to (23), all variable nodes in the Tanner graph of a regular LDPC code are connected to the same number of check nodes and vice versa. The Tanner graph depicted in the Figure 14 represents a regular code with $w_c = 2$ and $w_r = 4$.

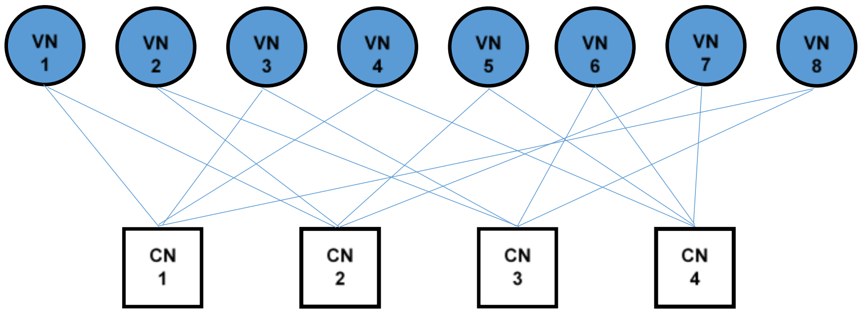


Figure 14: Tanner graph corresponding to the parity check matrix (24)

2.3. Construction of LDPC codes

Besides the obvious sparseness requirement, there is one more aspect of great importance which should be considered when constructing an LDPC code – the *girth* of the Tanner graph. The girth of a graph is the length of the shortest cycle in the graph. A cycle in a Tanner graph is defined as a finite set of connected edges, the cycle starts and ends at the same node and it satisfies the condition that no node (except the initial and final node) appears more than once [RD23]. The cycles in the Tanner graph of a LDPC code, especially short ones, significantly influence the performance of the code. They can even prevent the decoding algorithm from converging. This happens because the independence of the extrinsic information exchanged during the iterative decoding of LDPC codes is affected by the cycles in the Tanner graph [RD23]. Because of this, a well-constructed LDPC codes will have Tanner graphs with large girths. The following figure shows a Tanner graph with a cycle of length 4, which is indicated by bold lines.

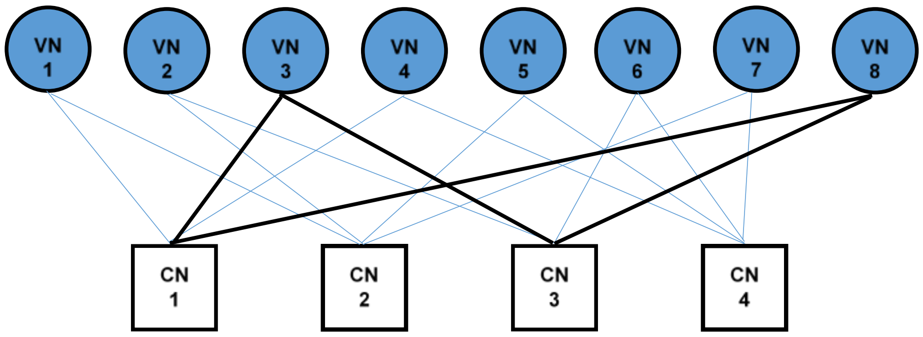


Figure 15: An example of a short cycle

2.3.1. Construction of Regular LDPC Codes

In this chapter, a brief overview of several construction methods for regular LDPC codes ($w_c = const, w_r = const$) is given.

2.3.1.1. Gallager Codes

This is the original construction presented by Gallager in [RD13]. The parity check matrix is obtained by concatenating w_c sub matrices:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_1} \\ \mathbf{H_2} \\ \vdots \\ \mathbf{H}_{w_c} \end{bmatrix} \tag{26}$$

The sub-matrix $\mathbf{H_1}$ contains a single binary 1 in each column and w_r binary 1's in each row. The i-th row contains 1's in columns $(i-1)w_r + 1$ to w_r.

The sub-matrices $\mathbf{H_2} \dots \mathbf{H}_{w_c}$ are random permutations of $\mathbf{H_1}$. The block length of a Gallager code is equal to $n \cdot w_r$, there are $n \cdot w_c$ parity checks and the design rate is equal to

$$CR = \frac{n \cdot w_r - n \cdot w_c}{n \cdot w_r} = 1 - \frac{w_c}{w_r} \tag{27}$$

The actual code rate may defer from the design rate because **H** is not guaranteed to be full rank. Additional disadvantages of Gallager codes are the possibility of short cycles and their non-suitability for quick encoding. On the other hand, they are easy to construct and show good performance.

2.3.1.2. Quasi-Cyclic LDPC Codes

The parity check matrix of a quasi-cyclic LDPC code consists of square sub-matrices which are either full-rank circulants, or zero matrices. The circulant matrix has the following form [RD24]:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & & 0 \end{bmatrix}_{L\,x\,L} \tag{28}$$

The parity check matrix **H** is constructed in the following way:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^{a_{11}} & \mathbf{P}^{a_{12}} & \cdots & \mathbf{P}^{a_{1n}} \\ \mathbf{P}^{a_{21}} & \mathbf{P}^{a_{22}} & \cdots & \mathbf{P}^{a_{2n}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{P}^{a_{m1}} & \mathbf{P}^{a_{m2}} & \cdots & \mathbf{P}^{a_{mn}} \end{bmatrix} \tag{29}$$

where $a_{ij} = \{1,2,\dots,L-1,\infty\}$ and $\mathbf{P}^{\infty}$ is a zero matrix. The block size of a quasi-cyclic LDPC code is equal to nL, there are mL parity checks and the design rate is equal to

$$CR = \frac{nL-mL}{nL} = \frac{n-m}{n} \tag{30}$$

In order to produce a regular LDPC code, all sub matrices have to be non-zero matrices. As well as Gallager codes, quasi-cyclic LDPC codes neither guarantee the absence of short cycles, nor the generation of the full rank matrix **H** (actual code rate may defer from the design code rate). However, these codes are suitable for low hardware cost encoding (feedback shift registers with spatial complexity proportional to mL) and they have low memory requirements.

2.3.1.3. Array Codes

The same circulant matrix **P** as for quasi-cyclic LDPC codes is used. For L = q, where q is a prime number and integers $j \le k \le q$, the parity check matrix is generated in the following way:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{I} & \mathbf{P}^{1} & \cdots & \mathbf{P}^{\mathrm{j}-1} & \cdots & \mathbf{P}^{(\mathrm{k}-1)} \\ \mathbf{I} & \mathbf{P}^{2} & \cdots & \mathbf{P}^{2(\mathrm{j}-1)} & \cdots & \mathbf{P}^{2(\mathrm{k}-1)} \\ \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{I} & \mathbf{P}^{(\mathrm{j}-1)} & \cdots & \mathbf{P}^{(\mathrm{j}-1)(\mathrm{j}-1)} & \cdots & \mathbf{P}^{(\mathrm{j}-1)(\mathrm{k}-1)} \end{bmatrix} \tag{31}$$

The block length of array codes is equal to kq, there are jq parity checks and the design rate is equal to

$$CR = \frac{kq - jq}{kL} = \frac{k-j}{k} \tag{32}$$

Array codes guarantee non-existence of cycles with length 4, when $j > 3$ [RD24]. Unlike Galager and quasi-cyclic codes, the actual rate is always equal to the design rate, because the generated parity check matrix **H** is guaranteed to be full rank. Array codes show worse performance than Gallager codes and do not have structure suitable for quick encoding.

2.3.1.4. Random Regular LDPC Codes

Starting from the zero matrix $\mathbf{H_0}$, a regular LDPC code with CR = (n – m)/n can be constructed in the following three steps, which are repeated until n columns (matrix $\mathbf{H_n}$) have been generated [RD24]:

> Step 1: At step i, choose a random $m \times 1$ column of weight w_c, which is not already being used in $\mathbf{H_{i-1}}$ or rejected in previous steps, and add it to $\mathbf{H_{i-1}}$.
>
> Step 2: Check whether the added column has more than one 1-component in common with any column in $\mathbf{H_{i-1}}$. If not, go to next step. Else reject the column and go back to step 1.
>
> Step 3: If all rows have weight less than w_r , save $\mathbf{H_i}$ and continue to the next round. Else, reject the column and go back to step 1.

Random LDPC codes are proven to have good performance [RD20]. One of the reasons for their good performance is the non-existence of cycles with length 4 due to step 2 of the generation algorithm. The parity check matrix may not be full rank, which is why the actual rate is perhaps not equal to the design rate. The structure of random LDPC codes is not well suited for quick encoding and the construction itself can be computationally very demanding for large frame lengths and column weights.

2.3.2. Construction of Irregular LDPC codes

Several different methods for construction of irregular LDPC codes are briefly explained in this section.

2.3.2.1. Modified Array Codes

Sub-matrix **P** and integers L, j, k and q are defined as done with array codes. The way the parity check matrix **H** is created is modified in the following way:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{P}^{1} & \cdots & \mathbf{P}^{j-2} & \cdots & \mathbf{P}^{(k-2)} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \cdots & \mathbf{P}^{2(j-3)} & \cdots & \mathbf{P}^{2(k-3)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I} & \cdots & \mathbf{P}^{(j-1)(k-j)} \end{bmatrix} \tag{33}$$

Same as in array codes, the block length of modified array codes is kq, there are jq parity checks and the design rate is equal to

$$CR = \frac{kq - jq}{kL} = \frac{k-j}{k} \tag{34}$$

H is always full rank which means that the actual code rate equals the design rate. For $j > 3$, no length-4 cycles exist. Modified array codes can be encoded using the encoding algorithm proposed in [RD25], whose time complexity is linearly proportional to the block length.

2.3.2.2. Random Construction of Irregular LDPC Codes

One can represent the variable and check node degree distribution for irregular LDPC codes in polynomial form. For variable node degree distribution we have [RD26]

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1} \tag{35}$$

where λ_i is the fraction of edges emanating from variable nodes of degree *i* and d_v is the maximum variable node degree of the irregular LDPC codes. The check node degree distribution can defined in similar manner:

$$\rho(x) = \sum_{i=2}^{d_c} \rho_i x^{i-1} \tag{36}$$

where ρ_i is the fraction of edges emanating from check nodes of degree i and d_c is the maximum check node degree of the irregular LDPC codes. The random construction technique for an irregular LDPC code from a profile given by $\lambda(x)$ and $\rho(x)$ as well as block length *N* consists of the following steps:

Step 1: Start with an empty matrix with size $M \times N$, where $M = N(1-R)$. For rate R we have:

$$R = 1 - \frac{\sum_i \rho_i / i}{\sum_j \lambda_j / j} \tag{37}$$

Step 2: According to a given degree distribution, calculate the number of ones in each row (from 1 to M) and in each column (from 1 to N).

Step 3: For each column, select a random row and assign ones to this matrix entry (row, column).

Step 4: Check if the degree constraint for the corresponding row is violated and if any cycles of length 4 will be formed. If any of the two constraints is violated, select another random row and check again.

This technique guarantees that the girth is higher than four and that the column weights are exactly as desired. However, for short block lengths, row weights cannot be controlled.

2.3.2.3. Accurate Random Construction of Irregular LDPC Codes

In [RD26], an improved version of the random construction technique, called accurate random construction, is presented. In order to precisely control the row weights, step 3 from the random construction algorithm is changed. The improvement is achieved by the generation of a vector **u** with length equal to the total number of 1s in the parity check matrix. Each element of **u** contains a row number in which a non-zero entry can be placed. Instead of the direct random selection of rows, they are randomly selected from the vector **u.** After placing the non-zero entry in the parity check matrix, the corresponding element of **u** is deleted. At the end of the process, the vector **u** will be empty.

2.3.2.4. Speed-up Accurate Random Construction Irregular LDPC Codes

According to [RD26], this technique provides better performance of LDPC codes than the previous two. The following example describes the technique (CR = 0.5, i.e. N = 2M):

Step 1: Start with an empty matrix **H** with size $M \times M$

Step 2: Split **H** into two sub-matrices [$M \times M$ | $M \times M$]

Step 3: Set both sub-matrices to eye matrices (identity matrices in our case).

Step 4: According to a given degree distribution (λ, ρ), calculate the number of ones in each row and column.

Step 5: Subtract one from the number of ones of each column and two from the number of ones of each row.

Step 6: For each column, randomly choose rows from **u** and assign ones to that position.

Step 7: Check if the degree constraint for the corresponding row is violated and if any cycles of length 4 will be formed. If any of the two constraints is violated, select another random row and check again. When a row which satisfies the two constraints has been found, delete the entry from the vector **u**.

2.4. Encoding

Like with any linear block code, LDPC encoding can be performed through linear mapping of an information sequence **u** into the corresponding code word **c** using generator matrix **G**:

$$\mathbf{c} = \mathbf{u}\,\mathbf{G} \tag{38}$$

2.4.1. Conventional Encoders

The conventional LDPC encoding algorithm is systematic with the generator matrix **G** derived from the parity check matrix **H** by Gaussian elimination (row permutations, sums of rows, column permutations). The goal is to transform the parity check matrix into the systematic form:

$$\mathbf{H}' = [\mathbf{P} \mid \mathbf{I}] \tag{39}$$

Generator matrix $\mathbf{G} = [\mathbf{I}_{k\times k}|\mathbf{P}]$ is then easily obtained, according to Equation (14). The computational complexity of transforming the parity check matrix into systematic form is $O(n^3)$ [RD16], where n stands for the frame length. The low-density property is lost in the Gaussian elimination process, which is why the computational complexity of the encoding is approximately equal to the complexity of the matrix multiplication [RD16], namely $O(n^2)$. LDPC codes are very often used with large block sizes, in which case the encoding complexity can be significant. Because of this, the focus is on algorithms for encoding of LDPC codes which would, unlike the conventional encoding of block codes, take advantage of the sparseness of LDPC codes and result in lower complexity.

2.4.2. Message-Passing Encoders

In [RD17] and [RD18] a message-passing encoder is proposed. The main idea is to look at the code word at the transmitter side like it has been transmitted over a binary erasure channel, with the missing parity bits being erased by the channel and then to perform message-passing decoding algorithm (discussed in the next chapter) in order to recover the erased bits, i.e. missing parity bits. This algorithm is not completely reliable, because it fails in the case when missing parity bits form a stopping set. A stopping set is a subset *S* of the variable nodes such that every check node connected to *S* is connected to *S* at least twice. This happens due to the inability of the message-passing algorithm to proceed with decoding, if there is a check node connected to more than one erasure [RD19].

2.4.3. Encoding by Approximate Linear Triangulation

Another encoding algorithm, without any stopping set constraints is presented in [RD16]. It transforms the parity check matrix into an approximate lower triangular form (ALT) by performing row and column permutations. Additions of rows are not performed. The main advantage of this algorithm is the fact that the sparseness of the parity check matrix is preserved in its ALT form. This results in almost linear encoding computational complexity O(n+g^2), where n denotes the block length and g << n is called the "gap" of linear encoding and represents the number of rows that cannot be transformed into triangular through simple permutations.

The first step is to transform the parity check matrix into its ALT form. The result is the matrix $\mathbf{H}_{ALT}$ of the following form:

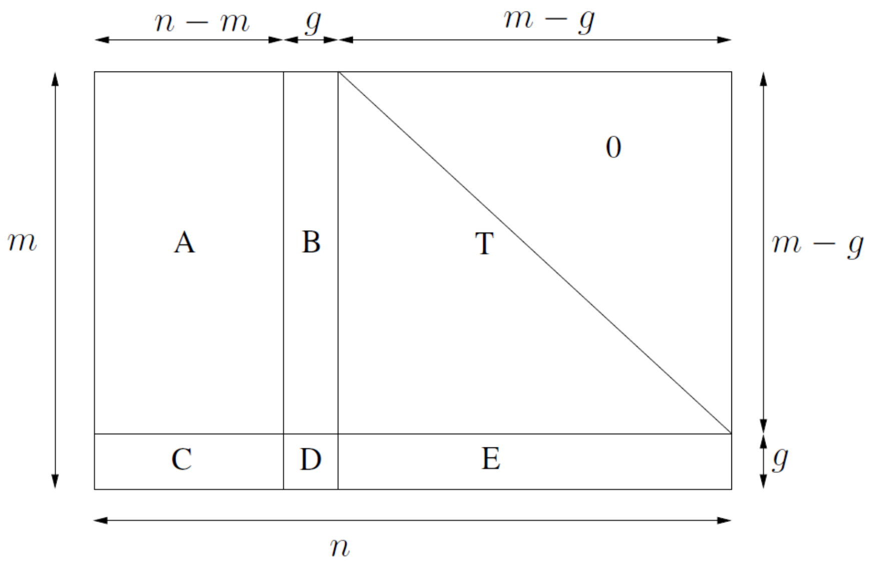


Figure 16: Approximate lower triangular form of the parity check matrix

It is important to note that all sub-matrices of the matrix $\mathbf{H}_{ALT}$ are sparse. The next step is to construct the systematic approximately lower triangular form (SALT) of the parity check-matrix. Using Gaussian transformation, the matrix **E** is transformed into an all-zero matrix and matrix **D** into an identity matrix.

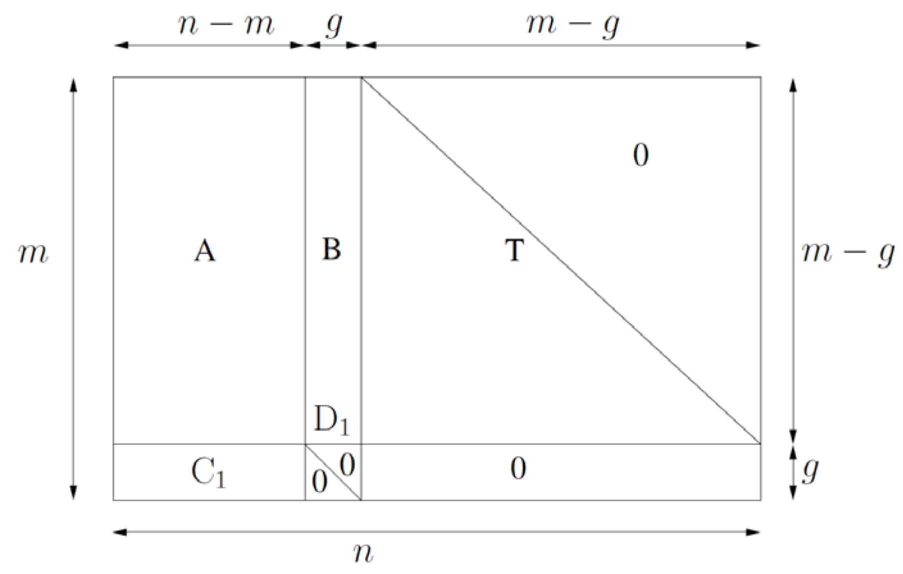


Figure 17: Systematic approximate lower triangular form of the parity check matrix

The first *n-m* bit positions of the code word are occupied by data bits **u**. Parity bits are divided into two segments: *g* bits long segment $\mathbf{p}_1$ and *m-g* bits long segment $\mathbf{p}_2$. The first segment of parity check bits is obtained in the following way:

$$\mathbf{p_1} = \mathbf{u} \cdot \mathbf{C}_1^{\mathrm{T}} \tag{40}$$

From the parity check condition in Equation (15) we obtain

$$\mathbf{A} \cdot \mathbf{u}^{\mathrm{T}} + \mathbf{B} \cdot \mathbf{p}_1^{\mathrm{T}} + \mathbf{T} \cdot \mathbf{p}_2^{\mathrm{T}} = \mathbf{0}_{m\times 1} \tag{41}$$

And for the second segment $\boldsymbol{p_2}$ we have that

$$p_2(l) = \sum_{j=1}^{n-m} A_{l,j} \cdot u_j + \sum_{j=1}^{g} B_{l,j} \cdot p_1(j) + \sum_{j=1}^{l-1} T_{l,j} \cdot p_2(j) \tag{42}$$

for $l = 1,2,\dots,m-g$.

All matrices used to calculate parity bits are sparse, except for $\mathbf{C_1}$, which is used to calculate g parity bits. As total computational complexity of the algorithm equals $O(n+g^2)$, it is of utmost importance to find a SALT form of the parity check matrix in a way that minimizes *g.* In [RD16], an algorithm called “greedy permutation algorithm” is proposed for generating SALT form of a sparse matrix. The complexity of this algorithm is $O(n^3)$, i.e. it is the same as the complexity of the Gaussian permutation.

2.5. Decoding

As already mentioned, decoding of LDPC codes is performed in an iterative manner. Iterative decoding is a technique that employs a soft-output decoding algorithm that is iterated several times. The goal is to approach the performance of ML decoding while minimizing the computational complexity [RD2]. Iterative decoding algorithms are sometimes referred to as probabilistic decoding, because of their property to maximize the a posteriori probability that a certain symbol has been sent given the noisy observation [RD2]. They typically represent a posteriori information in the form of *likelihood ratios (*LR) or, if decoding is performed in logarithmic domain, in the form of *log-likelihood ratios* (LLRs):

$$LR(u_i|\mathbf{y}) = \frac{P(u_i=1|\boldsymbol{y})}{P(u_i=-1|\boldsymbol{y})} \tag{43}$$

$$LR(y_i|x_i) = \frac{P(y_i|x_i=+1)}{P(y_i|x_i=-1)} \tag{44}$$

$$LLR(y_i|x_i) = ln\frac{P(y_i|x_i=+1)}{P(y_i|x_i=-1)} \tag{45}$$

where u_i, y_i, x_i denote output, noisy observation (received signal) and the original message bit, respectively. The hard decision on a bit is determined by the sign of its LLR and the magnitude indicates the probability that the right decision has been made [RD3]. In case of BPSK, for the output of an ideal demodulator we have [RD4]:

$$P(y_i|x_i = \pm 1) = \frac{1}{\sqrt{\pi \cdot N_0}} \exp\left(-\frac{E_s}{N_0}(y_i \mp a)^2\right) \tag{46}$$

where E_s stands for the energy per symbol and a is the fading amplitude. By combining Equations (45) and (46) we get:

$$LLR(y_i|x_i) = 4a\frac{E_s}{N_0} \cdot y_i = L_c \cdot y_i \tag{47}$$

For an AWGN channel, $a = 1$. The value L_c reflects the state of the channel and it might be considered as a measure for channel reliability [RD4].

2.5.1. Belief Propagation

The belief propagation algorithm was proposed by Gallager in his original paper. It has been re-discovered several times and it is also known under other names such as Pearl's algorithm or Sum-Product algorithm [RD2].

The algorithm is based on the Tanner graph representation in which each received bit is associated with one variable node. The nodes of the graph exchange messages through the edges in an iterative manner. The messages are in LR form and represent soft information about the state of a bit corresponding to a variable node. The node update schedule used by the belief propagation algorithm is called flooding schedule and it consists of four steps [RD3], which are presented in detail in the sequel.

Step 1: **Initialization**

A priori LRs are calculated for each VN:

$$LR_{in} = \frac{P(x_i=+1|y_i)}{P(x_i=-1|y_i)} = e^{2y_i/\sigma^2} \tag{48}$$

where σ^2 is the variance of the received signal. Message passing starts in that each VN passes its LR_{in} value to each CN it is connected to. At this point we have:

$$M_{v\to c} = LR_{in} \tag{49}$$

Step 2: **Check node update**

In this step, a response of each check to each variable node it is connected to is calculated:

$$\delta = \prod_{v' \in v_c \setminus v} \frac{1 - M_{v' \to c}}{1 + M_{v' \to c}} \tag{50}$$

In the above equation, v is the variable node that the response is being calculated for, v_c represents the set of all variable nodes the check node is connected to and $v_c \setminus v$ is the same set, but without the variable node v. For the outgoing message we have:

$$M_{c \to v} = \frac{1 - \delta}{1 + \delta} \tag{51}$$

Step 3: **Variable node update**

In this step, soft outputs are calculated along with the response to the check nodes:

$$SO_v = LR_{in} \prod_{c \in c_v} M_{c \to v} \tag{52}$$

From the soft-output, outgoing messages are finally calculated:

$$M_{v \to c} = \frac{SO_v}{M_{c \to v}} \tag{53}$$

Step 4: **Stopping criteria**

At this point, soft outputs are available and hard decisions are made. If parity check equations are fulfilled or the maximum number of iterations has been reached, the decoding process is terminated. Otherwise, the process is repeated starting with step 2 of the procedure.

2.5.2. Belief Propagation in the Logarithmic Domain

Belief propagation algorithm demands a large number of multiplications which are expensive to implement compared to additions. Because of this, the BP algorithm is often implemented in the logarithmic domain. The schedule stays the same:

Step 1: **Initialization**

Instead of a priori LRs, a priori LLRs are calculated for each VN:

$$LR_{in} = ln\frac{P(x_i=+1|y_i)}{P(x_i=-1|y_i)} = \frac{2y_i}{\sigma^2} \tag{54}$$

which are then passed to the check nodes:

$$M_{v\to c} = LLR_{in} \tag{55}$$

Step 2: **Check node update**

Transforming the equation into logarithmic domain we get:

$$\delta = \prod_{v'\in v_c\backslash v}\frac{1-\exp(M_{v'\to c})}{1+\exp(M_{v'\to c})} = \prod_{v'\in v_c\backslash v} -\tanh\frac{M_{v'\to c}}{2} \tag{56}$$

$$M_{c\to v} = -2\ \text{atanh}(\delta) \tag{57}$$

Step 3: **Variable node update**

$$SO_v = LR_{in} + \sum_{c\in c_v} M_{c\to v} \tag{58}$$

$$M_{v\to c} = SO_v - M_{c\to v} \tag{59}$$

Step 4: **Stopping criteria**

If parity check equations are fulfilled or the maximum number of iterations has been reached, the decoding process is terminated. Otherwise, the process is repeated starting with the step 2.

Because of the high computational complexity of the tangent hyperbolic and arc-tangent hyperbolic functions, sub-optimal versions of the belief propagation algorithm in the logarithmic domain, which do not use these functions, are used. All these algorithms differ from the standard belief propagation algorithm only in the way they perform check node updates, because the variable node update of the belief propagation algorithm are not critical.

2.5.3. Min-Sum Algorithm

The sign and the absolute value of the $M_{c\to v}$ message are calculated separately. For the sign we have that

$$sign(\, M_{c\to v}) = \prod_{v'\in v_c\setminus v} sign(M_{v'\to c}) \tag{60}$$

Magnitude is calculated using following equation:

$$|M_{c\to v}| \approx \min_{v'\in v_c\setminus v} |M_{v'\to c}| \tag{61}$$

This means that all incoming messages, except the one with the smallest magnitude, are ignored. The min-sum algorithm significantly reduces the computational complexity of the BP algorithm, however it performs poorly because it always overestimates the magnitude $|M_{c\to v}|$.

2.5.4. Normalized Min-Sum Algorithm

The idea behind the normalized min-sum algorithm is quite simple. The overestimation of the outgoing messages is compensated through multiplication by a normalizing factor α, with $0 < \alpha \leq 1$:

$$|M_{c\to v}| \approx \alpha \cdot \min_{v'\in v_c\setminus v} |M_{v'\to c}| \tag{62}$$

2.5.5. Offset Min-Sum Algorithm

Another way to compensate the over-estimation of the outgoing messages is to introduce negative offset:

$$|M_{c\to v}| \approx \max\ (\min_{v'\in v_c\setminus v} |M_{v'\to c}| - \beta, 0) \tag{63}$$

3. DVB-S2 and DVB-S2X Standards

Standards for digital video broadcasting (DVB) have been issued by a Joint Technical Committee (JTC) of the European Telecommunications Standards Institute (ETSI), European Committee for Electrotechnical Standardization (CENELEC) and European Broadcasting Union (EBU). DVB standards define physical and data distribution layers depending on the data distribution method:

- DVB-S, DVB-S2, DVB-S2X (draft) and DVB-SH for television via satellite
- DVB-C and DVB-C2 for television via cable
- DVB-T and DVB-T2 for terrestrial television

DVB-S2X (Draft ETSI EN 302 307-2) is an extension to the DVB-S2 standard (ETSI EN 302 307). DVB-S2X systems are optimized for the following broadband satellite applications [RD21]:

- Broadcast Services (BS) - Digital Multi-Program Television (TV)/High Definition Television (HDTV)
- Interactive Services (IS) - Interactive data services including Internet access
- Digital TV Contribution and Satellite News Gathering (DTVC/DSNG)
- Data content distribution/trunking and other professional applications (PS)

3.1. System Architecture

The following figure shows the system architecture of the DVB-S2 System. The architecture is not changed in the DVB-S2X extension.

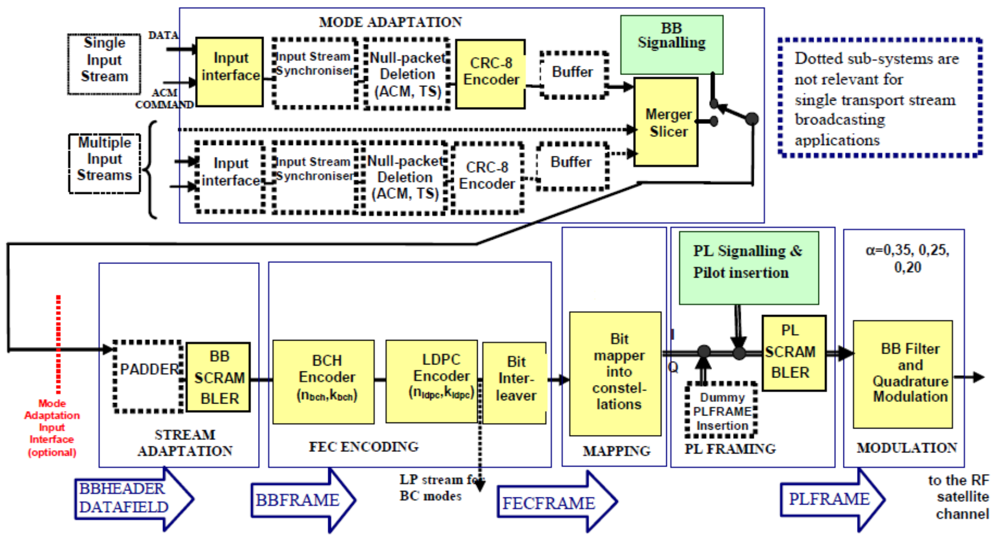

MODE ADAPTATION
Single Input Stream
DATA
ACM COMMAND
Input interface
Input Stream Synchroniser
Null-packet Deletion (ACM, TS)
CRC-8 Encoder
Buffer
BB Signalling
Merger Slicer
Multiple Input Streams
Dotted sub-systems are not relevant for single transport stream broadcasting applications
Mode Adaptation Input Interface (optional)
PADDER
BB SCRAM BLER
STREAM ADAPTATION
BCH Encoder (n_{bch},k_{bch})
LDPC Encoder (n_{ldpc},k_{ldpc})
Bit Inter-leaver
FEC ENCODING
Bit mapper into constel-lations
MAPPING
PL Signalling & Pilot insertion
PL SCRAM BLER
Dummy PLFRAME Insertion
PL FRAMING
α=0,35, 0,25, 0,20
BB Filter and Quadrature Modulation
MODULATION
BBHEADER DATAFIELD
BBFRAME
LP stream for BC modes
FECFRAME
PLFRAME
to the RF satellite channel

Figure 18: DVB-S2 and DVB-S2X system architecture

3.1.1. Mode Adaptation

Mode adaptation is application-dependent. It provides input stream interfacing, input stream synchronization (optional), null-packet deletion, CRC-8 coding for error detection at packet level in the receiver (for packetized input streams only), merging of input streams (for multiple input stream modes only) and slicing into data fields [RD21].

3.1.2. Stream Adaptation

Stream adaptation is applied, to provide padding to complete a baseband frame and baseband scrambling.

3.1.3. FEC Codec

Forward error correction (FEC) coding is carried out by the concatenation of BCH outer codes and LDPC inner codes. All allowed code rates and digital modulation schemes for normal and short frames are listed in the Table 1 and in Table 2, respectively. Original DVB-S2 combinations are marked as S2-MODCODs. The DVB-S2 standard defines two FEC block lengths: $n_{ldpc} = 64\,800$ bits or $n_{ldpc} = 16\,200$ bits depending on the application area. DVB-S2X introduces an additional block length of $n_{ldpc} = 32\,400$ bits. An outer BCH codec provides the correction of additional 12, 10 or 8 errors per frame, depending on the MODCOD, which are left over after LDPC decoding. The system can provide Constant Coding and Modulation (CCM), Variable Coding and Modulation (VCM) or Adaptive Coding and Modulation (ACM). In this latter case, a single satellite receiving station typically controls the protection mode of the full TDM multiplex, or multiple receiving stations control the protection mode of the traffic addressed to each one [RD21]. In the case of VCM and ACM, FEC and modulation scheme may differ for different frames, but they remain constant within a single frame.

FEC coding should enable “Quasi Error Free” (QEF) transmission using the highest code rate possible. The definition of QEF adopted for DVB-S2 is "less than one uncorrected error-event per transmission hour at the level of a 5 Mbit/s single TV service decoder", approximately corresponding to a transport stream packet error ratio PER < 10^{-7} before de-multiplexer [RD21].

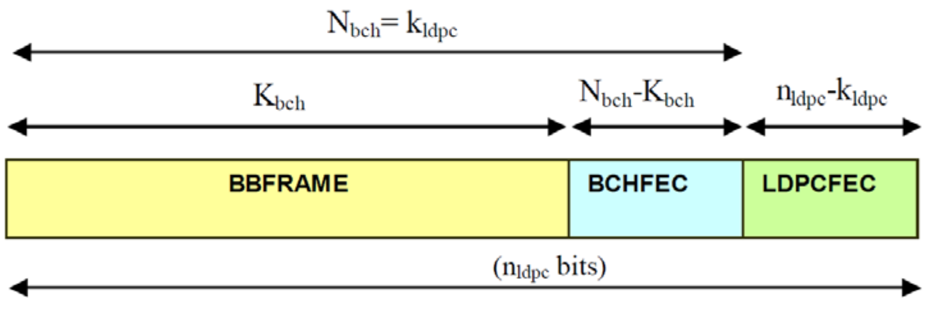


Figure 19: DVB-S2 format of data before bit interleaving

3.1.4. Bit Mapping into Constellations

In DVB-S2 four digital modulation schemes are defined: QPSK, 8PSK, 16APSK and 32APSK. Due to theirs quasi-constant envelope, QPSK and 8PSK are well suited for the applications in which the high power amplifier (HPA) in a transponder is driven near saturation. Non-constant envelope modulations are much more sensitive to non-linear distortions, e.g. at the output of a saturated high power amplifier there would be only one magnitude level even though the input signal is modulated with two magnitude levels using 16APSK. Non-linear compensation techniques, which can improve the power efficiency of non-constant envelope modulations, exist. The non-constant modulus modulations used in DVB-S2 have higher spectral efficiency than the quasi constant envelope ones, due to the higher number of bits in one symbol. DVB-S2X introduces additional non-constant modulus modulation schemes.

FECFRAME (normal) (see MODCODs below)	64 800 bits
QPSK	1/4, 1/3, 2/5 (S2-MODCODs)
	1/2, 3/5, 2/3, 3/4, 4/5, 5/6 8/9, 9/10 (S2-MODCODs)
	13/45
	9/20; 11/20
8PSK	3/5, 2/3, 3/4, 5/6, 8/9, 9/10 (S2-MODCODs)
	23/36; 25/36; 13/18
8APSK-L	5/9; 26/45
16APSK	2/3, 3/4, 4/5, 5/6, 8/9, 9/10 (S2-MODCODs)
	26/45; 3/5; 28/45; 23/36; 25/36; 13/18; 7/9; 77/90
16APSK-L	5/9; 8/15; 1/2; 3/5; 2/3
32APSK	3/4, 4/5, 5/6, 8/9, 9/10 (S2-MODCODs)
	32/45; 11/15; 7/9
32APSK-L	2/3
64APSK	11/15; 7/9; 4/5; 5/6
64APSK-L	32/45
128APSK	3/4; 7/9
256APSK	32/45; 3/4
256APSK-L	29/45; 2/3; 31/45; 11/15

Table 1: Code rates and modulation schemes for DVB-S2X – normal frame [RD21]

FECFRAME (short) (see MODCODs below)	16 200 bits
QPSK	1/4, 1/3, 2/5 (S2-MODCODs)
	1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9 (S2-MODCODs)
	11/45; 4/15; 14/45; 7/15; 8/15
	32/45
8PSK	3/5, 2/3, 3/4, 5/6, 8/9 (S2-MODCODs)
	7/15; 8/15; 26/45; 32/45
16APSK	2/3, 3/4, 4/5, 5/6, 8/9 (S2-MODCODs)
	7/15; 8/15; 26/45; 3/5; 32/45
32APSK	3/4, 4/5, 5/6, 8/9 (S2-MODCODs)
	2/3; 32/45

Table 2: Code rates and modulation schemes for DVB-S2X - short frame[RD21]

3.1.5. Physical Layer Framing

Physical layer framing shall be applied in a synchronous manner to the FEC frames, to provide Dummy PLFRAME insertion, Physical Layer Signaling, pilot symbols insertion (optional) and Physical Layer Scrambling for energy dispersal. Dummy PLFRAMEs are transmitted when no useful data are ready to be sent on the channel. The system provides a regular physical layer framing structure, based on SLOTs of M = 90 modulated symbols, allowing reliable receiver synchronization on the FEC block structure [RD21].

3.1.6. Baseband Filtering and Quadrature Modulation

Baseband filtering with squared-root raised cosine filters is used in order to shape the spectrum of the signal. DVB-S2 allows for roll-off factors: 0.35, 0.25 and 0.2. DVB-S2X introduces additional roll-off factors: 0.15, 0.1 and 0.05. Quadrature modulation is applied to produce the RF signal.

4. Implementation

The goal of the current thesis is to produce an efficient implementation of a DVB-S2X compatible LDPC codec for simulation purposes. In order to achieve this, Intel® Advanced Vector Extensions (AVX) are used. The programming language of choice is C++.

4.1. Intel® AVX

Intel® AVX is an instruction set enabling Single Instruction Multiple Data (SIMD) operations on Intel® CPUs. The basic idea behind SIMD is the processing of multiple data in a single step, which can significantly speed-up the throughput. For example, an addition of two length-8 vectors using AVX instruction set has the same throughput as the addition of two scalars.

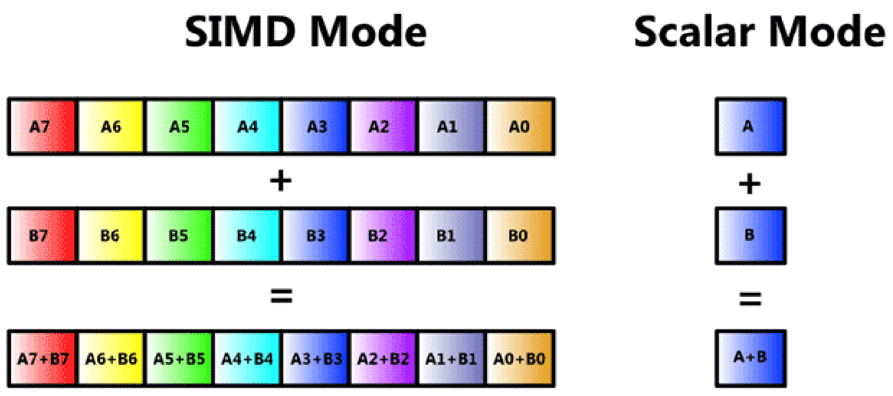


Figure 20: SIMD versus scalar instructions [RD27]

The hardware supporting Intel® AVX consists of the 16 256-bit YMM registers and a 32-bit control/status register called MXCSR [RD27]. The 256-bit register length enables the use of length-8 floating-point vectors (32 bit long elements) or length-4 double precision vectors (64 bit long elements).

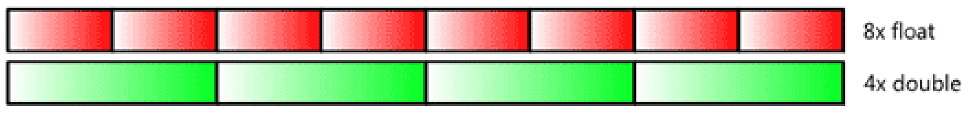


Figure 21: AVX-256 types [RD27]

The IEEE 754-2008 floating-point format offers satisfactory precision for the messages which are exchanged between check and variable nodes of an LDPC decoder. This means that it is possible to decode eight DVB-S2X frames in parallel using the AVX instruction set. The original goal of this thesis was to implement highly efficient LDPC codec for simulation purposes using the AVX instruction set, i.e. to encode and decode eight frames in parallel in order to maximize the data throughput. This approach increases latency (e.g. the decoding of the first received frame doesn't start before seven more frames are received) and is thus not well suited for real time operations.

4.2. AVX Implementation Problems

The main problem is the lack of an instruction in the AVX set, which can calculate the sign of a floating-point number efficiently. This represents a significant drawback because of the large number of sign computations which have to be performed in the scope of the check node updates. The sign of the outgoing messages (from the standpoint of a check node) is computed using the following relationship:

$$sign(\, M_{c\to v}) = \prod_{v'\in v_c \backslash v} sign(M_{v'\to c}) \tag{64}$$

Direct application of this equation for each outgoing message would require $w_r - 2$ multiplications per outgoing message, i.e. $w_r(w_r - 2)$ multiplications per check node update. Instead, we can calculate the product of signs of all incoming messages, P_{signs}. In order to calculate the sign of the message $M_{c\to v'}$, we just need to calculate the product of the sign of the message $M_{v'\to c}$ and P_{signs}:

$$P_{signs} = \prod_{v'\in v_c} sign(M_{v'\to c}) \tag{65}$$

$$sign(\, M_{c\to v}) = P_{signs} \cdot sign(M_{v'\to c}) \tag{66}$$

This requires a total of $2w_r - 1$ multiplications. The needed functionality can be defined as follows:

$$sign(M_{v\to c}) = \begin{cases} -1, & M_{v\to c} < 0 \\ 1, & M_{v\to c} \geq 0 \end{cases}$$

As already mentioned, the AVX instruction set doesn't offer such an instruction, so a function using multiple SIMD instructions has to be programmed. An "if - else" branch cannot be used because it is not a SIMD operation. Since $|M_{v\to c}|$ has to be computed anyway, a function which uses the following equation has been considered:

$$sign(M_{c\to v}) = \frac{M_{v\to c}}{|M_{v\to c}|} \tag{67}$$

The AVX division instruction performed on 16 floating-point numbers (8 dividends and 8 divisors) has a latency of 21 CPU cycles and throughput of 13, 14 or even 28 cycles depending on the CPU architecture. Because of the poor performance of division operation, an approach using multiple, computationally less demanding, operations has been considered. The result is the following function:

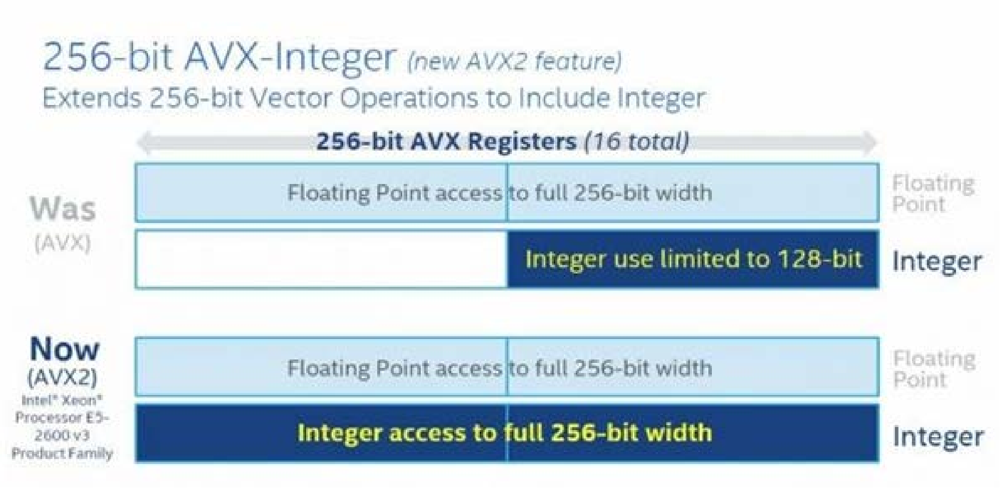
```
inline __m256 LDPC_codec::copy_sign_avx(__m256 a){

       __m256 c;

       c = _mm256_cmp_ps(a, zeros, 2);
        c = _mm256_castps_si256(c);
       c = _mm256_cvtepi32_ps(c);
       c = _mm256_mul_ps(c, twos);
       c = _mm256_add_ps(c, ones);

       return c;
```

Comparing SIMD operation and addition, both have a latency of 3 and a throughput of 1 CPU cycle, multiplication has a latency of 5 and a throughput of 1 CPU cycles. Data type conversion adds 3 additional cycles of latency and 1 of throughput. However, each instruction requires the result of the previous one. This results in very large latencies which in this case also reduces throughput. Latency could be decreased by insertion of additional instructions which have to be executed anyway, in times when CPU is idle.

However, processors offering the AVX2 instruction set came to market in the course of this thesis. The AVX2 instruction is a significantly improved version of AVX and it was decided to start implementing from the beginning using AVX2 instead of improving the AVX implementation which did not result in a satisfactory throughput.

4.3. Intel® AVX2

The most significant advantage of the AVX2 over the AVX instruction set is the existence of 256-bits integer SIMD instructions. The vector elements can be 64-bits, 32-bits, 16-bits or 8-bits long.



256-bit AVX-Integer (new AVX2 feature)
Extends 256-bit Vector Operations to Include Integer
256-bit AVX Registers (16 total)
Was
(AVX)
Floating Point access to full 256-bit width
Floating Point
Integer use limited to 128-bit
Integer
Now
(AVX2)
Intel® Xeon® Processor E5-2600 v3 Product Family
Floating Point access to full 256-bit width
Floating Point
Integer access to full 256-bit width
Integer

Figure 22: New Features (AVX2 vs. AVX)

The decoding of LDPC codes can be performed with a precision of 3 or more bits without performance loss [RD32]. In order to exploit this fact, 8-bits long elements are chosen which enables parallel processing of 32 frames. As it is presented in the following table, operations on 8-bit integers have lower latency and throughput than operations on floating-point numbers. Another very important improvement is the existence of sign instruction with the latency of 1 and the throughput of 0.5 CPU cycles. This feature alone saves $10.5w_r - 3.5$ CPU cycles per check node update compared to the implementation using AVX instruction set, because it enables efficient computation of the sign of the outgoing message. The AVX2 instruction set does not support multiplication for 8-bits longs integers. This is not a major issue when implementing the min-sum algorithm, because at least one operand in all multiplications performed has unit absolute value. This enables the replacement of the multiplication instruction by the sign instruction. For example, the following instruction call multiplies 32 absolute values (any positive 7-bits long value is allowed) stored in *abs_buf* vector with the corresponding 32 sign values stored in *sign_buf* (all elements must be equal to ±1).

```
Hcn[i][0].CV = _mm256_sign_epi8(abs_buf, sign_buf);
```

The `_mm256_sign_epi8` instruction copies the absolute value of the first operand and changes its sign if the sign of the second operand is negative. This makes it suitable for multiplications of two values with unit absolute values but whose signs have to be considered too; unlike in the previous example where it is guaranteed that the first operand is positive.

Another advantage of AVX2 is the existence of the instruction which computes absolute values of AVX2 array elements. This enables the additional increase of the data rate compared to the implementation using the AVX instruction set, because this set does not include such an instruction on floating-point numbers. Instead, in order to calculate the absolute value of an incoming message, one binary maximum operation and one multiplication are performed:

$$|M_{v\to c}| = \max(M_{v\to c} , -1 * M_{v\to c}) \tag{68}$$

The following table compares AVX and AVX2 in terms of latency and throughput of their instructions. In the case of the AVX instruction set, the values for absolute value and sign computations correspond to multiple instructions used to implement these functionalities.

Instruction	AVX (latency, through-put)	AVX2 (latency, through-put)
Minimum	3, 1	1, -
Maximum	3, 1	1, -
Addition	3, 1	1, 0.5
Sign	14, 4	1, 0.5
Subtraction	3, 1	1, -
Absolute value	6, 2	1, -

Table 3: AVX vs. AVX2 instruction set

4.4. Min-Sum Algorithm Family

Belief propagation algorithm has been implemented without use of SIMD operations, because of the lack of vector instructions that could perform complicated computations needed for the check node update as it is defined in this algorithm. The BP algorithm is used solely as a reference for the Min-Sum algorithm family in terms of frame error rate (FER). Its extremely bad performance in terms of computational complexity makes it unsuitable for time-efficient simulations. This is the reason why this thesis focuses on the efficient implementation of min-sum, normalized min-sum and offset min-sum algorithms.

The min-sum algorithm is the only decoding algorithm implemented using the AVX2 instruction set. The normalized min-sum and the offset min-sum algorithm are implemented using the AVX instruction set. The quantization of the offset and normalizing factors to 8 bits as well as the way to implement the multiplications needed for the normalized min-sum algorithm are some of the problems that arise. The data rate, latency and throughput values mentioned in the following chapters refer to the implementation of the min-sum algorithm using AVX2 instruction set.

4.4.1. Check Node Update

The sign and the absolute value of the outgoing messages are computed separately. The following table shows the rules used for the computation of absolute value for the three variations of the min-sum algorithm.

Min-sum	$\|M_{c\to v}\| \approx \min_{v' \in v_c/v} \|M_{v'\to c}\|$
Normalized min-sum	$\|M_{c\to v}\| \approx \alpha \cdot \min_{v' \in v_c/v} \|M_{v'\to c}\|$
Offset min-sum	$\|M_{c\to v}\| \approx \max\left(\min_{v' \in \frac{v_c}{v}} \|M_{v'\to c}\| - \beta, 0\right)$

Table 4: Check node updates used by the min-sum algorithm family

4.4.1.1. The Computation of the Absolute Value of the Outgoing Messages

It is clear that the calculation of the minimum value is the computationally most demanding aspect of the absolute value computation from the above table. This holds for each of the three algorithms. It is important to notice that the new minimum value has to be computed for each outgoing message, because the incoming message from a particular variable node has to be excluded from the set whose minimum is calculated when calculating the answer to that particular node. This means that each check node calculates w_r minima from w_r different sets of $w_r - 1$ positive numbers, where w_r stands for weight row of the sparse parity check matrix.

The straight forward approach to this problem would be to exclude the message $M_{v\to c}$ from the set when computing $M_{c\to v}$ and then to compute the minimum using a soft coded function able to compute the minimum of a set of any given length. The procedure is repeated w times for each check node. This approach requires $w_r(w_r - 2)$ binary minimum operations per check node per iteration. The operations needed to be performed in order to exclude one value from the set are not counted in. Independently of the way the w_r minima are calculated, all messages sent from a particular check node have the same absolute value except one. Let's use a check node connected to five variable nodes ($w_r = 5$) with incoming messages $M_{v1\to c} = 1, M_{v2\to c} = 2,\ M_{v3\to c} = 3,\ M_{v4\to c} = 4,\ M_{v5\to c} = 5$ as an example. Using the min-sum algorithm, the absolute values of outgoing messages would be as follows:

$$|M_{c\to v1}| = 2$$
$$|M_{c\to v2}| = 1$$
$$|M_{c\to v3}| = 1$$
$$|M_{c\to v4}| = 1$$
$$|M_{c\to v5}| = 1$$

This means that the calculation of the minimum for each sub-set of incoming messages doesn't have to be performed. Instead, we can calculate the first and the second minimum of the complete set of incoming messages. However, it is necessary to know which variable node has sent the message with the minimal absolute value to store this information. We can then set the absolute value of all outgoing messages to the first minimum of the complete set of incoming messages, with the exception of the message which is sent to the variable node from which the minimal absolute value has been received. The absolute value of this message should have the value of the second minimum.

This approach significantly reduces the computational complexity, especially for large weight rows. However, it cannot be used for parallel decoding of multiple frames using SIMD operations, because there is no guarantee that the messages with minimal and second minimal absolute value are sent from the same variable nodes for different frames. On the contrary, this will rarely be the case. An approach, which enables parallel minima calculation for multiple frames using SIMD operations, has to be used. Let's consider the straightforward method.

Below is the C++ source code which computes the absolute value of an outgoing message using this approach. The incoming messages are stored in vector VCs. It can be seen that for each outgoing value, we have a different set of positive numbers whose minimum is calculated. However, these sets are very similar; all elements of any two sets are the same but one. This means that a method that calls the above function for each outgoing message is calculating the binary minimum of same numbers multiple times.

```
for (branch = mod2sparse_first_in_row(H, i);
      !mod2sparse_at_end(branch);
    branch =mod2sparse_next_in_row(branch))
                                                              {
                                 if (c != 0)
                                       min_1 = VCs[0];
                                 else
                                       min_1 = VCs[1];

                                 for (j = 0; j < k; j++) {

                                       if (c != j)

       min_1=_mm256_min_ps(VCs[j], min_1);
       }
                          }
```

A significant reduction of the computational complexity can be achieved by storing these values instead of calculating them repeatedly. In order to achieve this, an algorithm which produces w_r outputs from a set of w_r positive floating-point numbers is used. The graphical representation in the following diagram illustrates this algorithm in the case when $w_r = 6$.

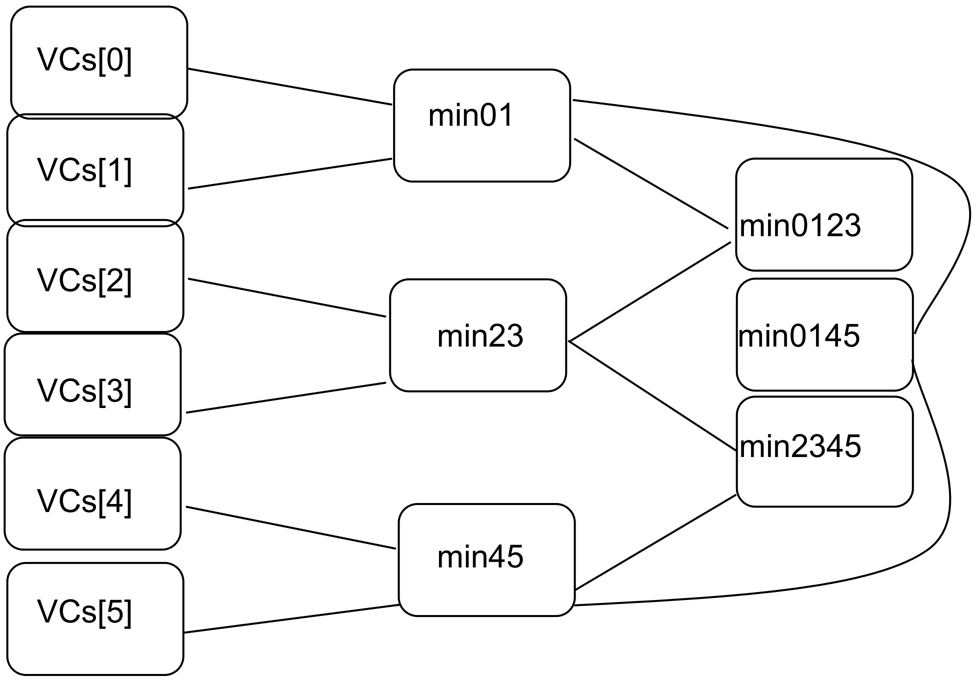


Figure 23: Minimum calculation

Each node labeled with “min” calculates the minimum from its two inputs. The nodes labeled with “VCs” represent the incoming messages. AVX2 data types and SIMD minimum operations are used, which means that the algorithm actually calculates $32w_r$ outputs from $32w_r$ inputs. The graph shows the example when $w_r = 6$. Using the values min0123, min2345 and min0145 we can calculate the absolute value of the outgoing messages (CVs) using one minimum operation per message. In our example we have that

$$|CVs[0]| = \min(VCs[1], min2345)$$
$$|CVs[1]| = \min(VCs[0], min2345)$$
$$|CVs[2]| = \min(VCs[3], min0145)$$
$$|CVs[3]| = \min(VCs[2], min0145)$$
$$|CVs[4]| = \min(VCs[5], min0123)$$
$$|CVs[5]| = \min(VCs[4], min0123)$$

Twelve minimum operations have to be performed using this algorithm in order to calculate the absolute values of all outputs when $w_r = 6$. It is clear that this approach delivers better performance in terms of computational complexity than the algorithm that repeatedly calculates the minimum of sets of $w_r - 1$ numbers which uses $w_r(w_r - 2) = 24$ minimum operation for the same functionality.

Figure 23 will be different for different weight rows of the parity check matrix. The parity check matrices are defined in the DVB-S2 and DVB-S2X standards which are different for different code rates and frame lengths. The algorithm is hard-coded and the adequate variant is chosen depending on the weight row. For some code rates, w_r differs throughout the matrix; this is why the choice of the algorithm variant has to be made independently for each check node. This can be done in real time by simply counting the number of incoming messages. An alternative approach would be to store weight rows of all check nodes for all code rates in a look-up table. The weights of parity check matrices from DVB-S2 standard are listed in the following table.

It is clear that the knowledge of all possible row weights is crucial for efficient implementation of min-sum algorithm. For this purpose, a simple program which investigates the parity check matrices for all MODCODS was written. The following table shows the properties of parity check matrices used for decoding of LDPC codes as they are defined in the DVB-S2 and DVB-S2X.

Frame Length	Code Rate	Row Weights	Column Weights	Standard
long	1/2	6, 7	8, 3, 2, 1	DVB-S2
long	1/3	4, 5	12, 3, 2, 1	DVB-S2
long	1/4	3, 4	12, 3, 2, 1	DVB-S2
long	2/3	9, 10	13, 3, 2, 1	DVB-S2
long	2/5	5, 6	12, 3, 2, 1	DVB-S2
long	2/9	3, 4	11, 3, 2, 1	DVB-S2X
long	3/4	13, 14	12, 3, 2, 1	DVB-S2
long	3/5	10, 11	12, 3, 2, 1	DVB-S2
long	4/5	17, 18	11, 3, 2, 1	DVB-S2
long	5/6	21, 22	13, 3, 2, 1	DVB-S2
long	7/9	17, 18	12, 5, 3, 2, 1	DVB-S2X
long	8/9	26, 27	4, 3, 2, 1	DVB-S2
long	9/10	29, 30	4, 3, 2, 1	DVB-S2
long	9/20	6, 7	12, 4, 3, 2, 1	DVB-S2X
long	11/20	8, 9	13, 3, 2, 1	DVB-S2X
long	13/18	13, 14	10, 3, 2, 1	DVB-S2X
long	13/45	4, 5	12, 3, 2, 1	DVB-S2X
long	18/30	10, 11	19, 4, 3, 2, 1	DVB-S2X
long	20/30	12, 14, 13	16, 4, 3, 2, 1	DVB-S2X
long	22/30	15, 16, 14, 13, 17	14, 4, 3, 2, 1	DVB-S2X
long	23/36	9, 10	11, 6, 3, 2, 1	DVB-S2X
long	25/36	12, 13	11, 9, 3, 2, 1	DVB-S2X
long	26/45	9, 10	13, 12, 2, 3, 1	DVB-S2X
long	28/45	9, 10	11, 7, 3, 2, 1	DVB-S2X
long	90/180	6, 8, 7	18, 3, 16, 9, 6, 2, 1	DVB-S2X
long	96/180	7, 9, 8	20, 14, 12, 3, 4, 2, 1	DVB-S2X
long	100/180	7, 9, 8	16, 3, 15, 10, 8, 2, 1	DVB-S2X
long	104/180	8, 10, 9	18, 14, 3, 7, 4, 2, 1	DVB-S2X

long	116/180	10, 12, 11	18, 12, 10, 3, 4, 2, 1	DVB-S2X
long	124/180	12, 14, 13	16, 3, 13, 12, 2, 1	DVB-S2X
long	128/180	13, 15, 14, 12	14, 12, 3, 4, 2, 1	DVB-S2X
long	132/180	14, 15, 16	14, 12, 3, 4, 2 ,1	DVB-S2X
long	135/180	15, 17, 16	14, 11, 3, 4, 2, 1	DVB-S2X
long	140/180	18, 19, 20	14, 12, 4, 3, 2, 1	DVB-S2X
long	154/180	28, 30, 29	13, 12, 3, 5, 2, 1	DVB-S2X
short	1/3	4, 5	12, 3, 2, 1	DVB-S2
short	2/3	9, 10	13, 3, 2, 1	DVB-S2
short	2/5	5, 6	12, 3, 2, 1	DVB-S2
short	3/5	10, 11	12, 3, 2, 1	DVB-S2
short	8/9	26, 27	4, 3, 2, 1	DVB-S2
short	1/2	3, 8	6	DVB-S2
short	1/4	3, 12	4	DVB-S2
short	3/4	3, 12	12	DVB-S2
short	4/5	3	14	DVB-S2
short	5/6	3,13	19	DVB-S2
short	9/10	3,4	30	DVB-S2
short	4/15	4, 5	21, 4, 3, 2, 1	DVB-S2X
short	7/15	8, 9	24, 4, 3, 2, 1	DVB-S2X
short	8/15	9, 10	21, 4, 3, 2, 1	DVB-S2X
short	11/45	3, 4	10, 3, 2, 1	DVB-S2X
short	14/45	4, 5	12, 9, 3, 2, 1	DVB-S2X
short	32/45	12, 13	12, 5, 3, 2, 1	DVB-S2X
short	26/45	8, 9, 10	13, 12, 2, 3, 1	DVB-S2X

Table 5: Properties of DVB-S2 and DVB-S2X LDPC codes

The number of minimum calculations needed to calculate the absolute value of all outgoing messages for all values of w_r from the previous table is shown in the following figure.

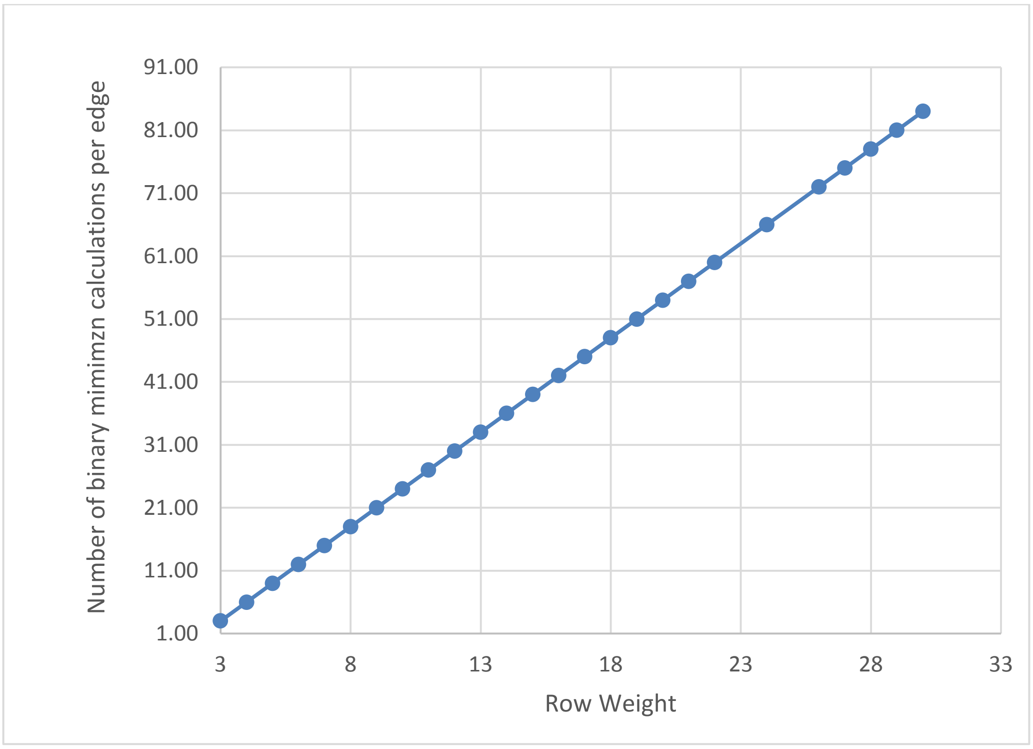


Figure 24: Number of binary min. operations per check node per iteration

Clearly, the number of performed binary minimum operations per check node per iteration increases linearly with the number of variable nodes the check node is connected to (weight row):

$$N_{bin-min} = 3 \cdot w_r - 6 \tag{69}$$

The check nodes with higher weights are producing more outputs, so the number of minimum operations per outgoing message (edge) might be more intuitive way to represent the computational complexity. This is shown in the following figure.

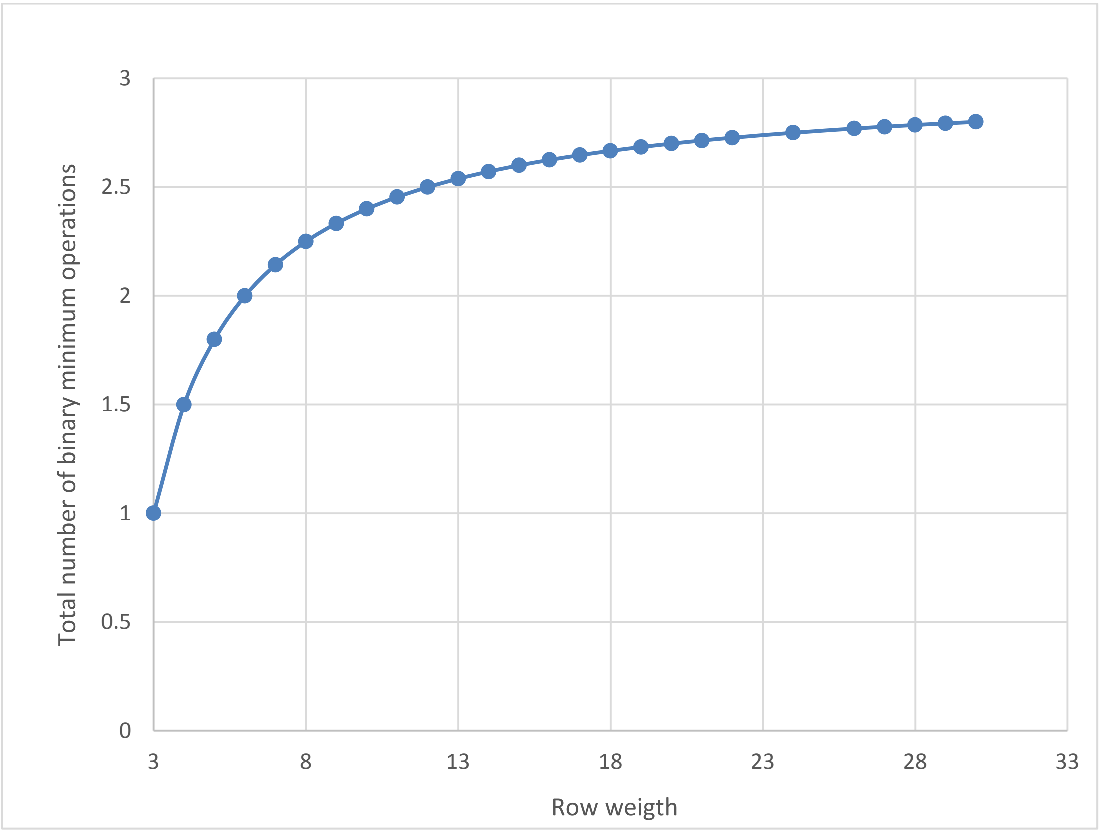


Figure 25: Binary min. operations per edge (sequential decoding)

It can be seen that the computational complexity per edge increases with the increase of row weight. As it can be seen from Table 5, the DVB-S2 and DVB-S2X parity check matrices used for higher code rates typically have higher row weights than the ones used for lower code rates which is why the check node update of the frames encoded with higher code rates is computationally slightly more demanding than check node update of frames encoded with lower code rates. However, the number of check nodes decreases as the code rate increases, which overpowers the more complex check node update of higher code rates and results in higher decoder complexity for lower code rates than for the higher ones, as expected.

The use of the AVX2 binary minimum instruction enables the decoding of 32 frames in parallel. This means that 32 times more outgoing messages are calculated using the same number of binary minimum operations. We can look at this as if there were 32 times more edges, although not all of them are used for decoding of the same frame. The following figure shows the needed number of binary minimum instructions per edge using the AVX instruction set.

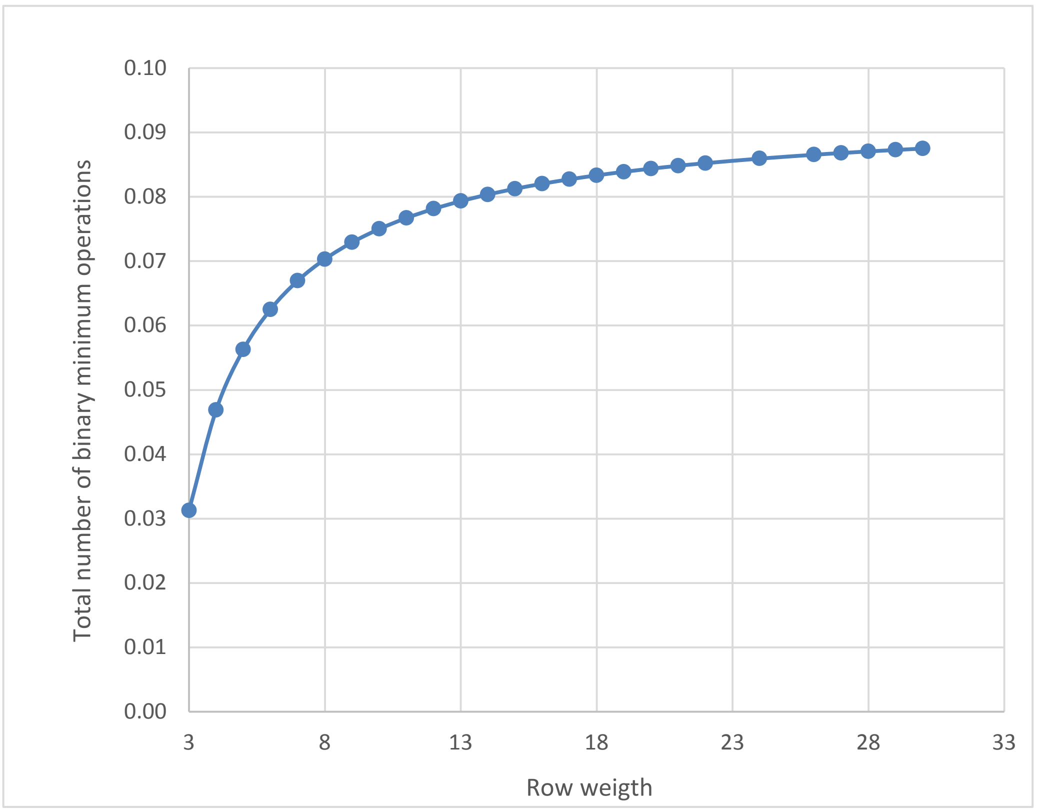


Figure 26: Binary min. operations per edge, decoding of 32 frames in parallel

4.4.1.2. Computation of the Sign of Outgoing Messages

As already mentioned, AVX2 instruction set offers a convenient and efficient way to compute the sign of outgoing messages. First, the signs of all incoming messages are computed using `_mm256_sign_epi8` instruction. The output is a length-w_r array of AVX2 vectors. Vector elements have unity absolute values and the sign which is equal to the sign of the corresponding incoming message. After that, all the signs are combined using the same instruction. This is equivalent to the multiplication of w_r number with unity absolute values. At the end, the sign of an outgoing message is obtained by the multiplication (using `_mm256_sign_epi8)` of the combined signed of all incoming messages by the sign of the incoming messages which originates from the variable node which is the destination of the outgoing message whose sign is being calculated.

4.4.1.3. Summary

In the sequel, a part of the source code is used for check node updates in the case when $w_r = 6$.

```
min01 = _mm256_min_epi8(VCs[0], VCs[1]);
sign = _mm256_sign_epi8(sign, VC_signs[1]);
min23 = _mm256_min_epi8(VCs[2], VCs[3]);
sign = _mm256_sign_epi8(sign, VC_signs[2]);
min45 = _mm256_min_epi8(VCs[4], VCs[5]);
sign = _mm256_sign_epi8(sign, VC_signs[3]);
min0123 = _mm256_min_epi8(min01, min23);
sign = _mm256_sign_epi8(sign, VC_signs[4]);
min2345 = _mm256_min_epi8(min23, min45);
sign = _mm256_sign_epi8(sign, VC_signs[5]);
min0145 = _mm256_min_epi8(min01, min45);

sign_buf0 = _mm256_sign_epi8(VC_signs[0], sign);
abs_buf0 = _mm256_min_epi8(min2345, VCs[1]);

sign_buf1 = _mm256_sign_epi8(VC_signs[1], sign);
abs_buf1 = _mm256_sign_epi8(min2345, VCs[0]);

sign_buf2 = _mm256_sign_epi8(VC_signs[2], sign);
abs_buf2 = _mm256_min_epi8(min0145, VCs[3]);

sign_buf3 = _mm256_sign_epi8(VC_signs[3], sign);
abs_buf3 = _mm256_min_epi8(min0145, VCs[2]);

sign_buf4 = _mm256_sign_epi8(VC_signs[4], sign);
abs_buf4 = _mm256_min_epi8(min0123, VCs[5]);

sign_buf5 = _mm256_sign_epi8(VC_signs[5], sign);
abs_buf5 = _mm256_min_epi8(min0123, VCs[4]);

Hcn[i][0].CV = _mm256_sign_epi8(abs_buf0,
sign_buf0);
Hcn[i][1].CV = _mm256_sign_epi8(abs_buf1,
sign_buf1);
Hcn[i][2].CV = _mm256_sign_epi8(abs_buf2,
sign_buf2);
```

As it can be seen from the source code, in order to avoid latency caused by the dependence of the instructions to be executed on the results of the previously executed instructions, the instructions used for the calculation of the absolute value and the sign of the outgoing messages are executed alternately.

After the signs and the absolute values of the outgoing messages have been calculated, the check node update is complete, if min-sum algorithm is used. Normalized min-sum algorithm requires w_r additional multiplications, and offset min-sum requires w_r additional maximum operations and w_r additional subtractions (see Table 4). The following table shows the number of computations needed to perform a check node update for 32 frames in parallel, using the min-sum algorithm, depending on the check node weight. The total number of computations depends on the number of check nodes, which depends on the frame length and the code rate.

Instruction	Number of instructions	Throughput
Addition	w_r	0.5
Comparison	w_r	0.5
Sign	$5w_r$	0.5
Absolute value	w_r	-
Minimum	$3w_r - 6$	-
Maximum	w_r	-
Total	$12w_r - 6$	$3.5w_r$

Table 6: Throughput, expressed in CPU cycles, of check node update for Min-Sum Algorithm

4.4.2. Variable Node Update

As mentioned before, variable node update is not time-critical and it is performed in the exactly same manner regardless of the decoding algorithm. In order to compute the soft output and outgoing messages, the following equations have to be implemented:

$$SO_v = LR_{in} + \sum_{c \in c_v} M_{c\to v} \tag{70}$$

$$M_{v\to c} = SO_v - M_{c\to v} \tag{71}$$

The computation of these two equations requires w_c additions and w_c subtractions per variable node per iteration, where w_c represent the column weight of the parity check matrix. The hard-output decision based on the value of soft-output requires one addition and one comparison. The addition is needed for reformatting the output of the AVX2 comparison instruction.

The following table shows the operations performed during one variable node update. As for check node update, the actual execution time depends on the frame length and code rate, i.e. on the parity check matrix.

Instruction	Number of instructions	Throughput
Addition	$w_c + 1$	0.5
Comparison	1	0.5
Subtraction	w_c	0.5
Total	$2w_c + 2$	$w_c + 1$

Table 7: Throughput, expressed in CPU cycles, of variable node update for Min-Sum Algorithm

4.4.3. Parity Check

After each iteration, the parity check is performed and if it is satisfied, the decoding process is stopped. The parity check equations are computed using SIMD instructions, like the rest of the decoding process. Only if all of the 32 frames satisfy the parity check conditions, an early stop is performed and the decoding of the next 32 frames starts.

Another approach would be to dynamically remove single frames that satisfy the parity check conditions from the AVX vector and replace them with new ones. Some problems that arise if this approach is used are as follows: The iteration count has to be tracked separately for each AVX vector element; the data would have to be loaded from AVX data type into standard integer arrays and vice versa very often; a high number of comparison SIMD instruction would have to be performed in order to cover all possible combinations of frames that do and that do not satisfy parity check conditions or alternatively, the parity check would have to be computed sequentially for 32 frames.

Because the efficient way to perform the parity check is not the main focus of this thesis and due to the implementation complications introduced by the dynamic approach and the uncertainty about its influence on the computational complexity of the decoding process, the static approach has been implemented.

4.4.4. Memory Access

Accessing memory linearly is faster than random memory access. One of the reasons for this is the fact that not only the requested value is loaded into the cache memory. One can either linearize variable node memory accesses or check node memory accesses and decrease the number of random memory accesses this way. Both approaches have been tested and because of the slightly better performance in terms of time efficiency, it was decided to linearize the check node accesses.

Additional improvement is achieved due to the quantization of LLRs to 8 bits. The quantization results in the lower amount of randomly accessed memory compared to the implementations using more accurate data representation. It is important to note that quantization to 8 bits does not cause any performance degradation in terms of FER.

4.4.5. Decoder Data Rate

The following table shows date rates achieved by the decoder implemented using the AVX2 instruction set and min-sum decoding algorithm on Intel i7-4790 CPU with 3.60 GHz processor base frequency. The data rate is increased by the factor less than four when the simulation framework is run in parallel. One of the reasons for this is the fact that increased (turbo) CPU frequency of 4 GHz is used in single core operation.

The data rates in the table below are achieved without the use of an early stop criterion because of the intention to investigate the data rates with the exact number of iterations performed. An early stop can significantly increase the data rate in the scenarios with high SNR, i.e. small FER. In the cases when the frame error rate satisfies the QEF criterion, the decoder data rates for any maximal number of iterations are very similar to each and very close to data rates achieved with 1-3 iterations (more than 300 Mbit/s with four Intel i7-4790 CPU cores)

Number of CPU Cores	Number of min-sum iterations	Data Rate [Mbit/s]
1	3	115
4	3	340
1	20	40
4	20	100
1	50	17
4	50	37

Table 8: Throughput, expressed in CPU cycles, of variable node update for Min-Sum Algorithm

4.5. Encoder Implementation

In order to reduce computational complexity of the encoding process, the algorithm presented in the Chapter 2.4.3 is implemented instead of the conventional one. As already mentioned, the encoding of LDPC codes using approximate linear triangulation has almost linear computational complexity. Using the AVX2 instruction set to encode 32 frames in parallel, date rates of about 300 Mbit/s are achieved on the four Intel i7-4790 CPU using only one CPU core. If the simulation framework is run in parallel on all four CPU cores, the encoder can achieve data rates as high as 800 Mbit/s.

5. Simulation Results

5.1. Comparison of Decoding Algorithms

The better performance of Normalized Min-Sum Algorithm and Offset-Min-Sum Algorithm compared to the Min-Sum-Algorithm are evident from the Figure 27. However, it can be seen that the performance improvement decreases with increasing SNR. The reason behind this is the fact that the optimal values of the normalizing factor *α* and the offset *β* change for different SNR values.

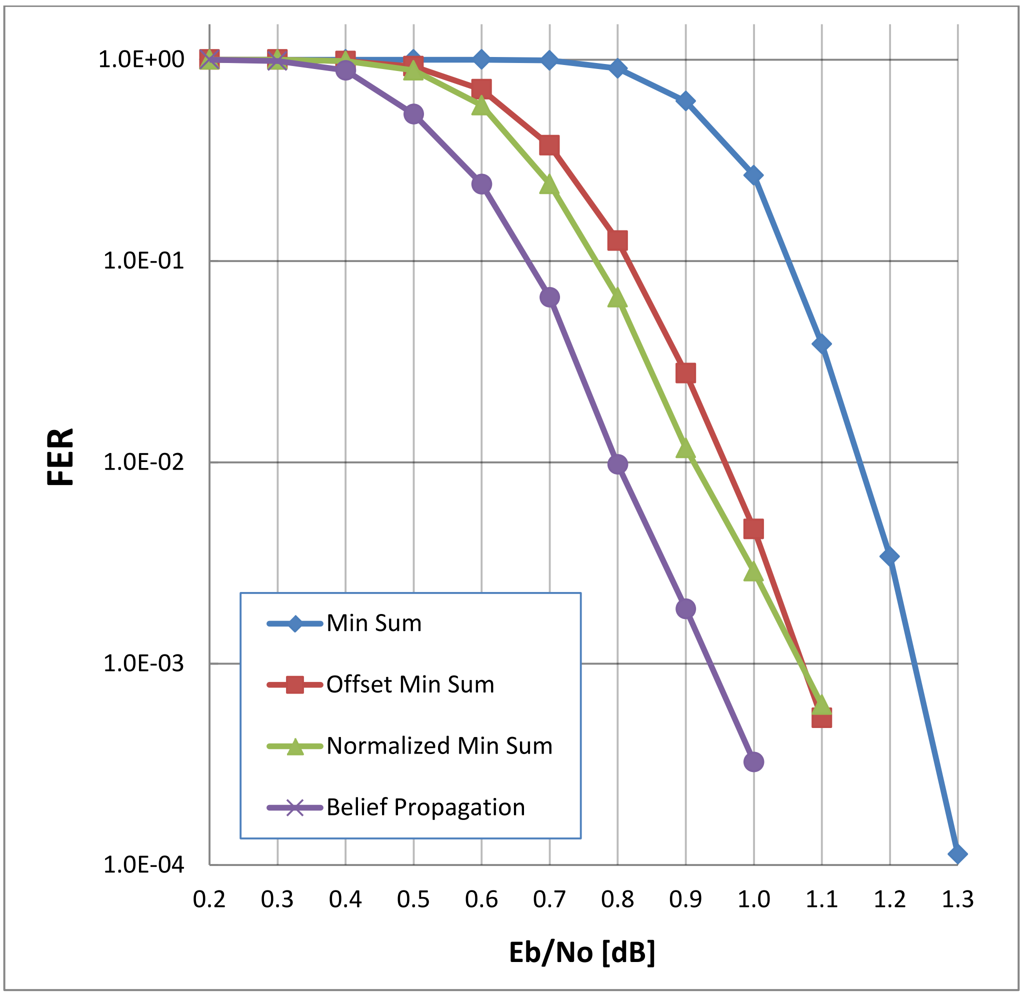


Figure 27: Short Frame, CR=1/3, 40 iterations

In [RD28] and [RD29] adaptive algorithms which optimize the values of *α* and *β* are presented. The implementation of such algorithms is beyond the scope of this thesis.

Instead, constant values *α* = *0.9* and *β* = *0.2* are used. These values are better suited for lower than higher SNR values, which explains the slight degradation of the performance improvement rate of the two algorithms, with increasing SNR, compared to the min-sum-algorithm. Several trials showed that higher SNR values require lower values of the normalizing factor *α* and higher values of the offset *β.*

5.2. Number of Iterations

The performance of the code for different number of iterations is shown in Figure 28. It can be seen that a larger number of iterations results in better performance. LDPC codes rely on the exchange of mutual information between check and variable nodes which is done twice per iteration. More iterations mean further information exchange, which results in better performance.

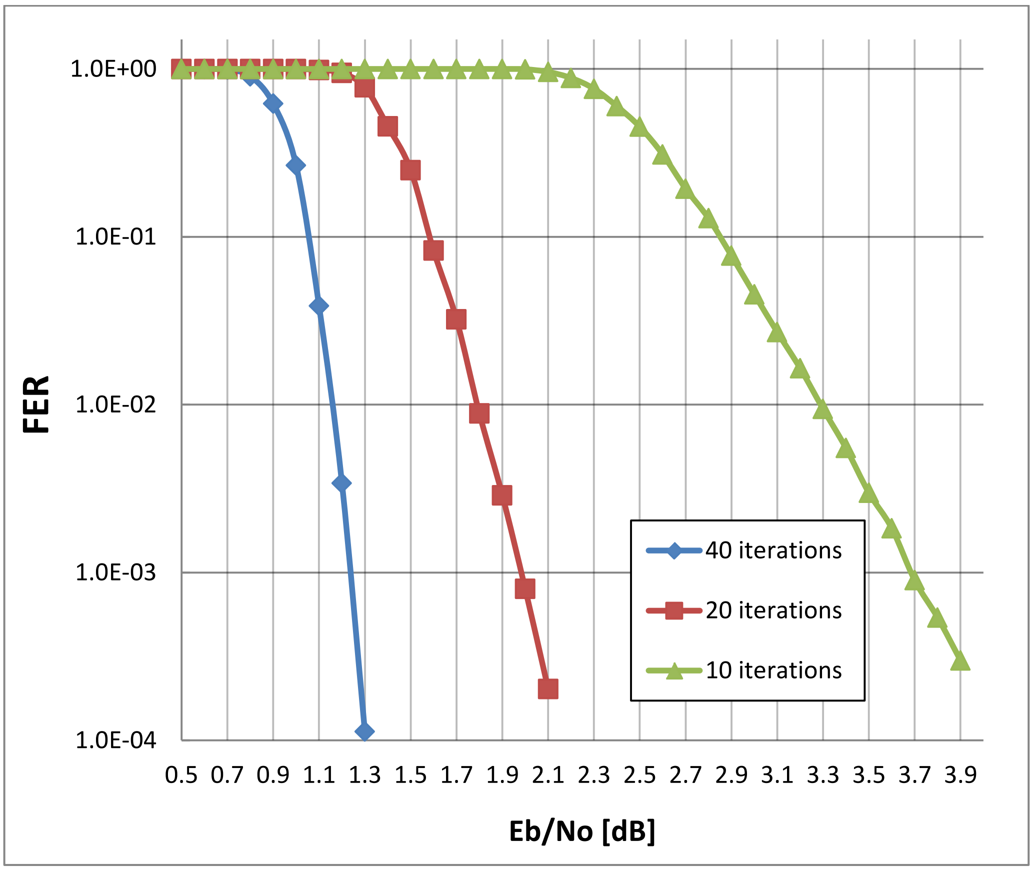


Figure 28: Short Frame, CR=1/3, Min-Sum Algorithm

However, performance improves more when the number of iterations is increased from 10 to 20 than when it is increased from 20 to 40, even though the increase in number of iterations is higher in the second case. Because the rate of performance improvement decreases with the number of iterations and each additional iteration requires additional computation, an upper bound on the number of iterations is set. The rate of performance improvement doesn't decrease as fast as in the case of turbo codes, which is why LDPC decoders usually use higher number of iterations than turbo decoders.

5.3. Code Rate

As expected, LDPC codes with lower code rates perform better, in terms of FER vs. SNR than the ones with higher code rates, because more redundancy enables better error correction. However, the FER curves of high CR codes are a bit steeper.

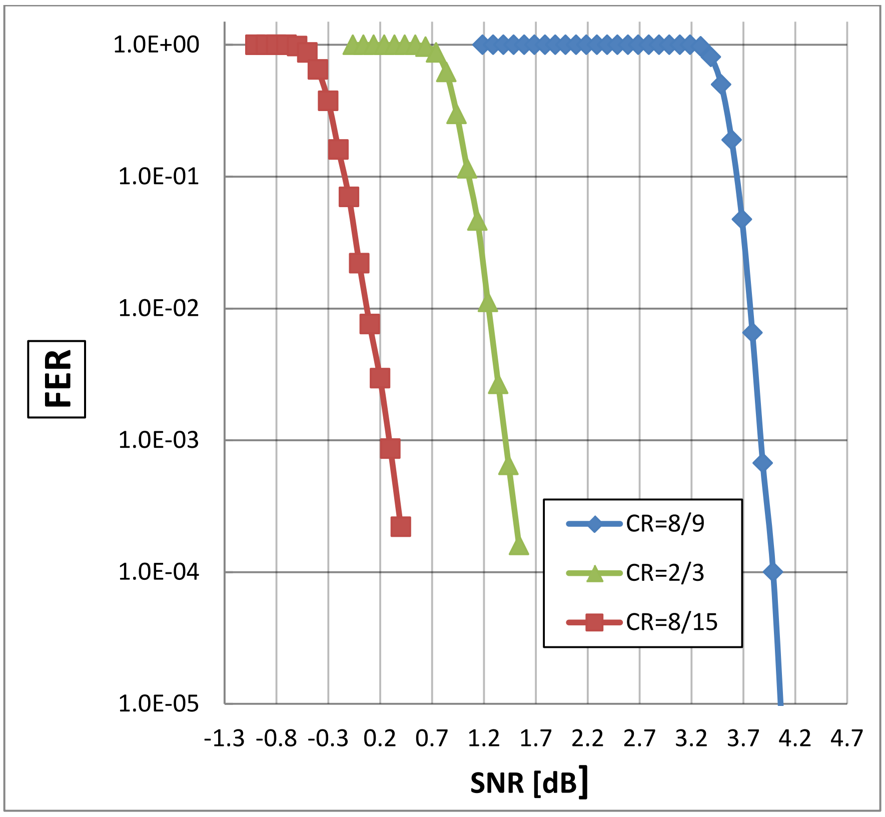


Figure 29: Short Frame, Min-Sum Algorithm, 10 iterations

5.4. Short Frame vs. Long Frame

As it can be seen in Figure 30, short frame operations shows worse performance than long frame operations. This is somewhat expected, because LDPC codes are famous for their good performance in the case of long frame lengths, which is one of the reason why they were chosen for the new DVB standards. In the DVB-S2 standard, it is stated that a performance degradation of 0.2 dB to 0.3 dB should be expected for short frames compared to normal frames. The simulation results confirm this statement.

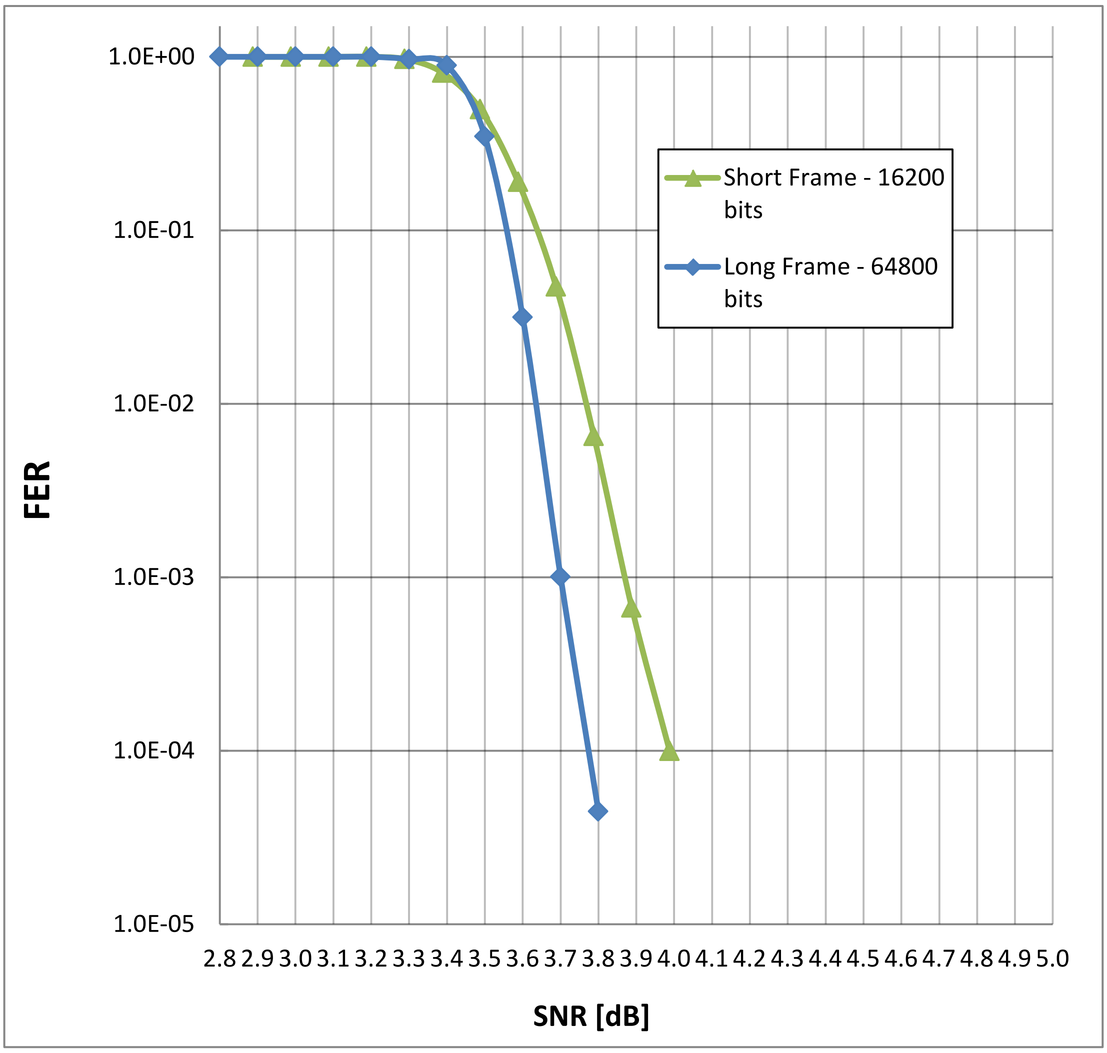


Figure 30: CR=1/3, Min-Sum Algorithm, 10 iterations

6. Conclusion

From the computational point of view, decoding of LDPC codes can very complex if the optimal decoding technique, the so-called Belief Propagation Algorithm, is used. This is the reason why the Min-Sum Algorithm or one of its modified variants are typically implemented.

In the case of software decoders, further performance improvement in terms of computational complexity can be achieved through the use of SIMD operations. The newest generation of Intel processors offers a single-instruction-multiple-data instruction extension to the standard x86 instruction set, called the Advanced Vector Extensions which offers registers that are 256 bits long. Because 8-bit data representation offers sufficient precision for LDPC decoding, 32 frames can be decoded in parallel. This results in decoding data rates of up to 150 Mbit/s on fourth generation Intel i-7 processors, if 20 iterations of the Min-Sum Algorithm are used.

Static parallel approach to the early stop criterion, as presented in the Chapter 5, will result in a higher number of iterations for some frames than it is needed and thus increases the decoding latency. The solution which dynamically replaces the frames that satisfied the parity check conditions with new ones, instead of replacing all 32 frames at the time when all of them have satisfied the parity check conditions, is beyond the scope of this thesis and represents an interesting topic for further research. The quantization of the offset and normalizing factors to 8 bits as well as the way to implement the multiplications needed for the Normalized Min-Sum Algorithm are issues that have to be addressed before implementing the Normalized Min-Sum and Offset Min-Sum Algorithms using AVX2 instruction set.