Early-Decision Decoding of LDPC Codes

Anton Blad
to TRUT64...
Abstract

Since their rediscovery in 1995, low-density parity-check (LDPC) codes have received wide-spread attention as practical capacity-approaching code candidates. It has been shown that the class of codes can perform arbitrarily close to the channel capacity, and LDPC codes are also used or suggested for a number of important current and future communication standards. However, the problem of implementing an energy-efficient decoder has not yet been solved. Whereas the decoding algorithm is computationally simple, with uncomplicated arithmetic operations and low accuracy requirements, the random structure and irregularity of a theoretically well-defined code does not easily allow efficient VLSI implementations. Thus the LDPC decoding algorithm can be said to be communication-bound rather than computation-bound.

In this thesis, a modification to the sum-product decoding algorithm called early-decision decoding is suggested. The modification is based on the idea that the values of the bits in a block can be decided individually during decoding. As the sum-product decoding algorithm is a soft-decision decoder, a reliability can be defined for each bit. When the reliability of a bit is above a certain threshold, the bit can be removed from the rest of the decoding process, and thus the internal communication associated with the bit can be removed in subsequent iterations. However, with the early decision modification, an increased error probability is associated. Thus, bounds on the achievable performance as well as methods to detect graph inconsistencies resulting from erroneous decisions are presented. Also, a hybrid decoder achieving a negligible performance penalty compared to the sum-product decoder is presented. With the hybrid decoder, the internal communication is reduced with up to 40% for a rate-1/2 code with a length of 1152 bits, whereas increasing the rate allows significantly higher gains.

The algorithms have been implemented in a Xilinx Virtex 5 FPGA, and the resulting slice utilization and energy dissipation have been estimated. However, due to increased logic overhead of the early decision decoder, the slice utilization increases from 14.5% to 21.0%, whereas the logic energy dissipation reduction from 499 pJ to 291 pJ per iteration and bit is offset by the clock distribution power, increased from 141 pJ to 191 pJ per iteration and bit. Still, the early decision decoder shows a net 16% estimated decrease of energy dissipation.
Acknowledgments

Welcome to the subjective part of this thesis, the only part containing the word “I” and adjectives with any kind of individually characterized meaning.

Being a PhD student has been a serious challenge the past few years. Having a lot of freedom at work is inspiring, but also very demanding, and I have often been annoyed at not being able to work more efficiently. At the same time, the rewards of seeing new ideas work, of getting papers accepted and presenting them at conferences are very enticing. During my time at Electronics Systems, there are many people I want to thank for making my life and work pleasant.

First of all, I thank my supervisor Oscar Gustafsson for his guidance, support and patience. His enthusiasm for his work and his positive attitude have been strong motivations for me.

Thanks also go to

- my family Bengt, Maj, Lisa and Tove Blad for always supporting me and being at my side.
- Peter Danielis for help with the sum-product decoder implementation.
- Kent Palmkvist for help with VHDL coding and synthesis.
- Greger Karlströms for help with the computer systems and programs in some of the courses I teach.
- my other colleagues at Electronics Systems for providing a nice and friendly working environment.
- Håkan Johansson, Christer Svensson, Jerzy Dabrowski, Rashad Ramzan, Timmy Sundström, Naveed Ahsan, Shakeel Ahmed and Jonas Fritzin for cooperative work in the direct sampling sigma-delta RF receiver front-end project (not included in this thesis).
- Fredrik Kuivinen for being a nice friend and lunch companion. Though I wonder what you will do to your thesis the day they find out that $P = NP$.
- Jakob Rosén for letting me chill out in his office between the hard work. You should learn to trust the dice to make all your bothering decisions for you, though.
- Soheil Samii and Carl-Fredrik Neikter for late-night company while writing this thesis. However, I can’t see why you insist on always eating at that shoddy place where they charge you for queuing to the tables.
Acknowledgments

• the geese and ducks at the pond for taking me down to the earth again when
  I feel lost in the mist.

• the ARA group at STMicroelectronics in Geneva for giving me new view-
  points of academic research and a nice working environment during my in-
  ternship in spring 2007.

• all my friends in Sweden and abroad for making my life pleasant.

Anton Blad

Linköping, February 2009
# CONTENTS

1 **INTRODUCTION**

1.1 Background ............................................... 1
1.2 Applications ........................................... 3
1.3 Scientific contributions ................................. 3
1.4 Thesis outline ........................................... 4
1.5 Publications ............................................. 5
  1.5.1 Publications included in the thesis ................. 5
  1.5.2 Publications not included in the thesis .......... 6

2 **ERROR CORRECTION CODING** 9

2.1 Digital communication ................................. 9
  2.1.1 Channel models .................................. 10
  2.1.2 Modulation formats .............................. 11
  2.1.3 Uncoded communication .......................... 12
2.2 Coding theory .......................................... 13
  2.2.1 Shannon bound .................................. 15
  2.2.2 Block codes ..................................... 16
2.3 LDPC codes ............................................. 19
  2.3.1 Tanner graphs .................................... 19
  2.3.2 Quasi-cyclic LDPC codes ......................... 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3</td>
<td>Randomized quasi-cyclic codes</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>Sum-product decoding</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Hardware implementation</td>
<td>24</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Parallel architecture</td>
<td>25</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Serial architecture</td>
<td>26</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Partly parallel architecture</td>
<td>27</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Finite wordlength considerations</td>
<td>27</td>
</tr>
<tr>
<td>2.5.5</td>
<td>Scaling of Φ((x))</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Early-decision decoding</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Early-decision algorithm</td>
<td>31</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Choice of threshold</td>
<td>33</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Handling of decided bits</td>
<td>34</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Bound on error correction capability</td>
<td>34</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Enforcing check constraints</td>
<td>35</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Enforcing check approximations</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>Hybrid decoding</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>Data representations</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Fixed wordlength</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Data compression</td>
<td>39</td>
</tr>
<tr>
<td>5</td>
<td>Decoder architecture</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Sum-product reference decoder architecture</td>
<td>43</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Architecture overview</td>
<td>44</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Memory block</td>
<td>45</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Variable node processing unit</td>
<td>45</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Check node processing unit</td>
<td>46</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Interconnection networks</td>
<td>48</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Memory address generation</td>
<td>50</td>
</tr>
<tr>
<td>5.1.7</td>
<td>Φ function</td>
<td>51</td>
</tr>
<tr>
<td>5.2</td>
<td>Early-decision decoder architecture</td>
<td>51</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Memory block</td>
<td>51</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Node processing units</td>
<td>52</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Early decision logic</td>
<td>53</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Enforcing check constraints</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>Hybrid decoder</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>Results</td>
<td>57</td>
</tr>
<tr>
<td>6.1</td>
<td>Floating-point simulations</td>
<td>57</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Choice of threshold</td>
<td>57</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Enforcing check constraints</td>
<td>65</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Hybrid decoding</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>Fixed-point simulations</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>Synthesis results</td>
<td>77</td>
</tr>
</tbody>
</table>
7 Conclusions and future work
  7.1 Conclusion ................................................. 79
  7.2 Future work ................................................ 80
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWGN</td>
<td>Additive White Gaussian Noise</td>
</tr>
<tr>
<td>BEC</td>
<td>Binary Erasure Channel</td>
</tr>
<tr>
<td>BER</td>
<td>Bit Error Rate</td>
</tr>
<tr>
<td>BLER</td>
<td>Block Error Rate</td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary Phase Shift Keying</td>
</tr>
<tr>
<td>BSC</td>
<td>Binary Symmetric Channel</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>CNU</td>
<td>Check Node processing Unit</td>
</tr>
<tr>
<td>DECT</td>
<td>Digital Enhanced Cordless Telecommunications</td>
</tr>
<tr>
<td>DTTB</td>
<td>Digital Terrestrial Television Broadcasting</td>
</tr>
<tr>
<td>DVB-S2</td>
<td>Digital Video Broadcasting - Satellite 2nd generation</td>
</tr>
<tr>
<td>$E_b/N_0$</td>
<td>Bit energy to noise spectral density (normalized SNR)</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correction Coding</td>
</tr>
<tr>
<td>ED</td>
<td>Early Decision</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning Satellite</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Programming</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LDPC</td>
<td>Low-Density Parity Check</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table</td>
</tr>
<tr>
<td>MUX</td>
<td>Multiplexer</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>QC-LDPC</td>
<td>Quasi-Cyclic Low-Density Parity-Check</td>
</tr>
<tr>
<td>QPSK</td>
<td>Quadrature Phase Shift Keying</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-Only Memory</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC (Very High Speed Integrated Circuit) Hardware Description Language</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
<tr>
<td>VNU</td>
<td>Variable Node processing Unit</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WPAN</td>
<td>Wireless Personal Area Network</td>
</tr>
</tbody>
</table>


Introduction

1.1 Background

Today, digital communication is used ubiquitously for transferring data between electronic equipment. Examples include cable and satellite TV, mobile phone voice and data transmissions, the communication between a DECT phone and its station, wired and wireless LAN, GPS, computer peripheral connections through USB and IEEE1394 and many more. The basic principles of a digital communication system are known, and one of the main advantages of digital communication systems over analog is the ability to use error correction coding (ECC) for the data transmission.

ECC is used in most digital communication systems to improve link performance and reduce transmitter power requirements [1]. By adding redundant data to the transmitted data stream, the system allows a limited amount of transmission errors to be corrected, resulting in a reduction of the number of errors in the transmitted information. However, for the digital data symbols that are received correctly, the received information is identical to that which is sent. This can be contrasted to analog communication systems, where transmission noise will irreversibly degrade the signal quality, and the only way to ensure a predefined signal quality at the receiver is to use enough transmitter power. Thus, the metrics used to measure the transmission quality are intrinsically different for digital and analog communication, with bit error rate (BER) or block error rate (BLER) for digital
CHAPTER 1. INTRODUCTION

As the quality metrics of digital and analog communication systems are different, the performance of an analog and a digital system can not be objectively compared with each other. However, it is often the case that a digital system with a quality subjectively comparable to that of an analog system requires significantly less power and/or bandwidth. One example is the switch from analog to digital TV, where image coding and ECC allow comparable or superior image quality using a transmission power of only 20% compared with the analog system.

A simple model of a digital communications system is showed in Fig. 1.1. The modeled system encompasses wireless and wired communications, as well as data storage, for example on optical disks and hard drives. However, the properties of the blocks are dependent on data rate, acceptable error probability, channel conditions, the nature of the data, and so on. In the communications system, data is usually first source coded (or compressed) to reduce the amount of data that needs to be transmitted, and then channel coded to add redundancy to protect against transmission errors. The modulator then converts the digital data stream into an analog waveform suitable for transmission. During transmission, the analog waveform is affected by channel noise, and thus the received signal differ to the sent. The result is that when the signal is demodulated, the digital data will contain errors. It is the purpose of the channel decoder to correct the errors using the redundancy introduced by the channel coder. Finally, the data stream is unpacked by the source decoder, recreating data suitable to be used by the application.

The work in this thesis considers the hardware implementation of the channel decoder for a specific class of codes called low-density parity-check (LDPC) codes. The channel decoder is often a significant contributor to the power dissipation of the receiver, and thus the complexity of the decoder becomes a limiting factor for the code performance. The need for low-power components is obviously high in battery-driven applications like hand helds and mobile phones, but becomes increasingly important also in stationary equipment like computers, computer peripherals and

Figure 1.1 Simple communications system model

systems, and signal-to-noise ratio (SNR) for analog systems. Whereas analog error correction is not principally impossible, analog communication systems are different enough on a system level to make practically feasible implementations hard to envisage.
1.2. APPLICATIONS

TV receivers, due to the need of removing the waste heat produced. Thus the focus of this work is on reducing the power dissipation of LDPC decoders, without sacrificing the error-correcting performance.

LDPC codes were discovered originally in 1962 by Robert Gallager [16]. He showed that the class of codes has excellent theoretical properties and he also provided a decoding algorithm. However, as the hardware of the time was not powerful enough to run the decoding algorithm efficiently, LDPC codes were not practically usable and were forgotten. They were rediscovered in 1995 [26,36], and have been shown to have a very good performance close to the theoretical Shannon limit [25,27]. Since the rediscovery, LDPC codes have been successfully used in a number of applications, and are suggested for use in a number of important future communication standards.

1.2 Applications

Today, LDPC codes are used or are proposed to be used in a number of applications with widely different characteristics and requirements. In 2003, a type of LDPC codes was accepted to be used for the DVB-S2 standard for satellite TV [46]. A similar type has also been accepted for the DTTB standard for digital TV in China [44]. The system-level requirements of these systems are relatively low, with low latency requirements as the communication is unidirectional, and relatively small constraints on power dissipation, as the user equipment is typically not battery-driven. Thus, the adopted code is complex with a resulting complex decoder implementation.

Opposite requirements apply for the WLAN IEEE 802.11n [48] and WiMax IEEE 802.16e [50] standards, for which LDPC codes have been chosen as optional ECC schemes. In these applications, communication is typically bi-directional, necessitating low latency. Also, the user equipment is typically battery-driven, making low power dissipation critical. For these applications, the code length is restricted directly by the latency requirements. However, it is preferable to reduce the decoder complexity as much as possible to save power dissipation.

Whereas these types of applications are seen as the primary motivation for the work in this thesis, LDPC codes are also used or suggested in several other standards and applications. Among them are the IEEE802.3an [51] standard for 10Gbit/s Ethernet, IEEE802.15.3c [13,14] ultra wide band proposals, and the gsfc-std-9100 [47] standard for deep-space communications. The IEEE802.15.3c is a WPAN protocol currently being standardized by the IEEE [49].

1.3 Scientific contributions

The major scientific contribution in this thesis is a modification to the usual sum-product decoding algorithm for LDPC codes, called the early-decision algorithm. The aim of the early-decision modification is to dynamically reduce the number of possible states of the decoder during decoding, and thereby reduce the amount
CHAPTER 1. INTRODUCTION

of internal communication of the hardware. However, this algorithm modification impacts the error correction performance of the code, and it is therefore also investigated how the modified decoding algorithm can be efficiently combined with the original algorithm to yield a resulting hybrid decoder which retains the performance of the original algorithm while still offering a reduction of internal communication.

A minor contribution is the observation of redundancy in the internal data format in a fixed-width implementation of the decoding algorithm. It is showed that a simple data encoding can further reduce the amount of internal communication.

The proposed architecture modifications are implemented and evaluated in software. It is verified that the modifications have an insignificant impact on the error correction performance, and the change in the internal communication is estimated. The early-decision algorithm has also been implemented in a Xilinx Virtex 5 FPGA for a class of LDPC codes somewhat similar to those used in the IEEE standards 802.16e and 802.11n. Using the implementation, the power dissipation has been estimated for the original and early-decision architectures.

1.4 Thesis outline

The outline of this thesis is as follows. In Chapter 2, the basics of digital communication and error correction coding systems are explained. General codes are defined through generator and parity check matrices, and the particular characteristics of LDPC codes are explained. Tanner graphs are introduced as a way to describe the structure of LDPC codes. The sum-product decoding algorithm is defined over the Tanner graph, and different choices of hardware architectures are explained.

In Chapter 3, the early decision decoding algorithm is presented. Choices of parameters are investigated, and performance limits of the algorithm are defined. It is also explained how the early decision algorithm can be combined with the sum-product decoding algorithm to reduce the performance losses.

In Chapter 4, a redundancy in the data representation is explained, and a decoder-internal coding of the data is introduced. It is shown in which cases usage of the internal data coding is beneficial.

In Chapter 5, a decoder architecture implementing the sum-product algorithm for a specific type of LDPC codes is described. It is also explained how to modify the architecture to implement the early decision algorithm, and the limits that the architecture impose on the choice of codes.

In Chapter 6, the performance of the early decision algorithm is shown. Computer simulations using floating-point precision are used to determine the behavior, the limitations and choice of suitable parameters of the algorithm. Simulations are also done using fixed-point precision to show that the algorithm is practically realizable. Furthermore, energy estimations of the hardware implementations of the sum-product and the early decision algorithm are presented.

Finally, in Chapter 7, conclusions are given and possible future work is discussed.
1.5 Publications

1.5.1 Publications included in the thesis

The following publications present work included in this thesis.


In this paper, the basic idea of the early decision algorithm as formulated in this thesis is presented, and simulations are included. The algorithm is also compared to similar previous work [45], and the differences are highlighted.


In this paper, the performance of the early decision algorithm is analyzed for different code parameters. Also, implementation considerations are mentioned.


In this paper, a modification to the choice of thresholds for the early decision algorithm is introduced, which yields a slight performance increase.


In this paper, the early decision-sum product hybrid algorithm is introduced. Due to the LDPC decoder’s ability to detect almost all decoding errors, the hybrid algorithm yields a negligible performance loss compared to the sum-product algorithm. Also, it is shown experimentally that, for fixed channel conditions, there is an optimal choice of threshold in terms of the computation complexity.

In this paper, the implementation aspects of the early decision decoder are elaborated. An architecture is suggested, and the complexity of the processing elements are evaluated and compared to those of the sum-product decoder.


In this paper, a redundancy in the data representation commonly used in LDPC decoders is described. It is shown how internal coding of the data can lead to a decrease of routed wires, with a small increase in complexity.

### 1.5.2 Publications not included in the thesis

The following publications present other work done by the author, but are not included in this thesis.


In this paper, a direct-sampling sigma-delta radio frontend for RF signals is suggested. The system-level requirements are analyzed for several radio protocols, and translated to requirements on the frontend.


In this paper, a method of analyzing analog-to-digital converters using parallel sigma-delta modulators is presented. The method can be used to analyze the analog-to-digital converter’s sensitivity to mismatch errors.


In this paper, the work in the above paper is explained in more detail. The formulation is exemplified for several analog-to-digital converter architectures, and it is shown that the technique may also be used to design new architectures with an inherent insensitivity to analog mismatch errors.

In this paper, the bit-level optimization of high-speed FIR decimation filters is considered. The optimization is formulated as an integer linear programming problem, minimizing a cost function defined on the number of full adders, half adders and registers required. Both pipeline registers and algorithmic delay registers are considered. The optimization algorithm is applied to the main FIR architectures for several filters, and their suitability are compared for several filter parameters.

• A. Blad and O. Gustafsson, "Integer linear programming-based bit-level optimization for high-speed FIR decimation filter architectures," *Circuits, Systems and Signal Processing - Special Issue on Low Power Digital Filter Design Techniques and Their Applications*, (accepted).

In this paper, the work in the above paper is extended to include optimization of the direct form FIR filter architecture utilizing the coefficient symmetry of linear-phase FIR filters. Also, signed-digit coefficient representations are considered, as well as signed data, and theoretical estimations of the complexities of the main architectures are provided.
In this chapter, the basics of digital communication systems and error correction coding are explained. General block codes are defined, and theoretical limits of communication systems are discussed. LDPC codes are defined as a special case of general block codes, and Tanner graphs are introduced as a way of visualizing the structure of a code. Finally, the sum-product decoding algorithm is explained, and possible hardware architectures for its implementation are discussed.

2.1 Digital communication

Consider a two-user digital communication system, such as the one shown in Fig. 2.1, where an endpoint A transmits information to an endpoint B. Whereas multi-user communication systems with multiple transmitting and receiving endpoints can be defined, only systems with one transmitter and receiver will be considered in this thesis. The system is digital, meaning that the information is represented by a sequence of symbols \( x_n \) from a finite discrete alphabet \( \mathcal{A} \). The sequence is mapped onto an analog signal \( s(t) \) which is transmitted to the receiver through the air, through a cable, or using any other medium. During transmission, the signal is distorted by noise \( n(t) \), and thus the received signal \( r(t) \) is not equal to the transmitted signal. By the demodulator, the received signal \( r(t) \) is mapped to symbols \( \tilde{x}_n \) from an alphabet \( \mathcal{B} \), which may or may not be the same as alphabet \( \mathcal{A} \).
and may be either discrete or continuous. Typically, if the output data stream is used directly by the receiving application, $\mathcal{B} = \mathcal{A}$. However, commonly some form of error coding is employed, which can benefit from including symbol reliability information in the reception alphabet $\mathcal{B}$.

### 2.1.1 Channel models

In analyzing the performance of a digital communication system, the chain in Fig. 2.1 is modeled as a probabilistic mapping $P(\tilde{X} = b \mid X = a), \forall a \in \mathcal{A}, b \in \mathcal{B}$, from the transmission alphabet $\mathcal{A}$ to the reception alphabet $\mathcal{B}$. The system modeled by the probabilistic mapping is formally called a channel, and $X$ and $\tilde{X}$ are stochastic variables denoting the input and output of the channel, respectively. For the channel, the following requirement must be satisfied for discrete reception alphabets

$$\sum_{b \in \mathcal{B}} P(\tilde{X} = b \mid X = a) = 1, \forall a \in \mathcal{A}, \quad (2.1)$$

or analogously for continuous reception alphabets

$$\int_{b \in \mathcal{B}} P(\tilde{X} = b \mid X = a) = 1, \forall a \in \mathcal{A}. \quad (2.2)$$

Depending on the characteristics of the modulator, demodulator, transmission medium, and the accuracy requirement of the model, different channel models are suitable. Some common channel models include

- the binary symmetric channel (BSC), a discrete channel defined by the alphabets $\mathcal{A} = \mathcal{B} = \{0, 1\}$, and the mapping

$$P(\tilde{X} = 0 \mid X = 0) = P(\tilde{X} = 1 \mid X = 1) = 1 - p$$
$$P(\tilde{X} = 1 \mid X = 0) = P(\tilde{X} = 0 \mid X = 1) = p,$$

where $p$ is the cross-over probability that the sent binary symbol will be received in error. The BSC is an adequate channel model in many cases when a hard-decision demodulator is used, as well as in early stages of a system design to compute the approximate performance of a digital communication system.
2.1. DIGITAL COMMUNICATION

- the binary erasure channel (BEC), a discrete channel defined by the alphabets \( A = \{0, 1\} \), \( B = \{0, 1, e\} \), and the mapping

\[
P(\tilde{X} = 0 \mid X = 0) = P(\tilde{X} = 1 \mid X = 1) = 1 - p
\]

\[
P(\tilde{X} = e \mid X = 0) = P(\tilde{X} = e \mid X = 1) = p
\]

\[
P(\tilde{X} = 1 \mid X = 0) = P(\tilde{X} = 0 \mid X = 1) = 0,
\]

where \( p \) is the erasure probability, i.e., the received symbols are either known by the receiver, or known that they are unknown. The binary erasure channel is commonly used in theoretical estimations of the performance of a digital communication system due to its simplicity, but can also be adequately used in low-noise system modeling.

- the additive white Gaussian noise (AWGN) channel with noise spectral density \( N_0 \), a continuous channel defined by a discrete alphabet \( A \) and a continuous alphabet \( B \), and the mapping

\[
P(\tilde{X} = b \mid X = a) = f_{(a,\sigma)}(b), \quad (2.3)
\]

where \( f_{(a,\sigma)}(b) \) is the probability density function for a stochastic variable with mean \( a \) and standard deviation \( \sigma = \sqrt{N_0/2} \). The size of the input alphabet is usually determined by the modulation format used, and is further explained in Sec. 2.1.2. The AWGN channel models real-world noise sources well, especially for cable-based communication systems.

- the Rayleigh and Rician fading channels. The Rayleigh channel is appropriate for modeling a wireless communication system when no line-of-sight is present between the transmitter and receiver, such as cellular phone networks and metropolitan area networks. The Rician channel is more appropriate when a dominating line-of-sight communication is available, such as for wireless LANs and personal area networks.

The work in this thesis considers the AWGN channel with a binary input alphabet only.

2.1.2 Modulation formats

The size of the transmission alphabet \( A \) for the AWGN channel is commonly determined by the modulation format used. Common modulation formats include

- the binary phase-shift keying (BPSK) modulation, using the transmission alphabet \( A = \{-\sqrt{E}, +\sqrt{E}\} \) and reception alphabet \( B = \mathbb{R} \). \( E \) denotes the symbol energy as measured at the receiver.

- the quadrature phase-shift keying (QPSK) modulation, using the transmission alphabet \( A = \sqrt{E/2}\{(-1 - i), (-1 + i), (+1 - i), (+1 + i)\} \) with complex symbols, and reception alphabet \( B = \mathbb{C} \). The binary source information is
mapped in blocks of two bits onto the symbols of the transmission alphabet. As the alphabets are complex, the probability density function in (2.3) is the probability density function for the two-dimensional Gaussian distribution.

- the quadrature amplitude (QAM) modulation, which is a generalization of the QPSK modulation to higher orders, using equi-spaced symbols from the complex plane.

In this thesis, BPSK modulation has been assumed exclusively. However, the methods are not limited to BPSK modulation, but may straightforwardly be applied on systems using other modulation formats as well.

### 2.1.3 Uncoded communication

In order to use the channel for communication of data, some way of mapping the binary source information to the transmitted symbols is needed. In the system using uncoded communication depicted in Fig. 2.2, this is done by the symbol mapper, which maps the source bits \( m_k \) to the transmitted symbols \( x_n \). The transmitted symbols may be produced at a different rate than the source bits are consumed.

On the receiver side, the end application is interested in the most likely symbols that were sent, and not the received symbols. However, the transmitted and received data are symbols from different alphabets, and thus a symbol demapper is used to infer the most likely transmitted symbols from the received ones, before mapping them back to the binary information stream \( \hat{m}_k \). In the uncoded case, this is done on a per-symbol basis.

For the BSC, the source bits are mapped directly to the transmitted symbols such that \( x_n = m_k \), where \( n = k \), whereas the BEC is not used with uncoded communication and is thus not discussed. For the AWGN with BPSK modulation, the source bits are conventionally mapped so that the bit 0 is mapped to the

---

**Figure 2.2** Model of uncoded digital communication system.
symbol $+\sqrt{E}$, whereas the bit 1 is mapped to the symbol $-\sqrt{E}$. For higher-order modulation, several source bits are mapped to each symbol, and the source bits are typically mapped using gray mapping so that symbols that are close in the complex plane differ by one bit. The optimal decision rules for the symbol demapper can be formulated as follows for different channels.

For the BSC,

$$\hat{m}_k = \begin{cases} \tilde{x}_n & \text{if } p < 0.5 \\ 1 - \tilde{x}_n & \text{if } p > 0.5, \end{cases}$$

(2.4)

where the case $p > 0.5$ is rather unlikely. For the AWGN channel using BPSK modulation,

$$\hat{m}_k = \begin{cases} 0 & \text{if } \tilde{x}_n > 0 \\ 1 & \text{if } \tilde{x}_n < 0. \end{cases}$$

(2.5)

Finally, if QPSK modulation with gray mapping of source bits to transmitted symbols is used,

$$\{\hat{m}_k, \hat{m}_{k+1}\} = \begin{cases} 00 & \text{if } \text{Re } \tilde{x}_n > 0, \text{Im } \tilde{x}_n > 0 \\ 01 & \text{if } \text{Re } \tilde{x}_n < 0, \text{Im } \tilde{x}_n > 0 \\ 11 & \text{if } \text{Re } \tilde{x}_n < 0, \text{Im } \tilde{x}_n < 0 \\ 10 & \text{if } \text{Re } \tilde{x}_n > 0, \text{Im } \tilde{x}_n < 0. \end{cases}$$

(2.6)

In analyzing the performance of a communication system, the probability of erroneous transmissions is interesting. For BPSK communication, the bit error probability can be defined as

$$P_{B,BPSK} = P(\hat{m}_k \neq m_k) = \frac{1}{2} \left[ P(\tilde{x}_n > 0 \mid x_n = 1)P(x_n = 1) + P(\tilde{x}_n < 0 \mid x_0 = 0)P(x_n = 0) \right] = Q\left(\sqrt{\frac{E}{\sigma}}\right) = Q\left(\sqrt{\frac{2E}{N_0}}\right),$$

(2.7)

where $Q(x)$ is the cumulative density function for normally distributed stochastic variables. However, it turns out that significantly lower error probabilities can be achieved by adding redundancy to the transmitted information, while keeping the total transmitter power unchanged. Thus, the individual symbol energies are reduced, and the saved energy is used to transmit redundant symbols computed from the information symbols according to some well-defined code.

### 2.2 Coding theory

Consider the error correction system in Fig. 2.3. As the codes in this thesis are block codes, the properties of the system are formulated assuming that a block code is used. Also, it is assumed that the symbols used for the messages are binary symbols. A message $m$ with $K$ bits is to be communicated over a noisy channel.
Figure 2.3 Error correction system overview

The message is encoded to the codeword $x$ with $N$ bits, where $N > K$. The codeword is then modulated to the analog signal $s(t)$ using BPSK modulation with an energy of $E$ per bit. During transmission over the AWGN channel, the noise signal $n(t)$ with a one-sided spectral density of $N_0$ is added to the signal to produce the received signal $r(t)$. The received signal is demodulated to produce the received vector $\tilde{x}$, which may contain either bits or scalars. The channel decoder is then used to find the most likely sent codeword $\hat{x}$, given the received vector $\tilde{x}$. From $\hat{x}$, the message bits $\hat{m}$ are then extracted.

For the system, a number of properties can be defined:

- The information transmitted is $K$ bits.
- The block size of the code is $N$ bits. Generally, in order to achieve better error correction performance, $N$ must be increased. However, a larger block size requires a more complex encoder/decoder and increases the latency of the system, and there is therefore a trade-off between these factors in the design of the coding system.
- The code rate is $R = K/N$. Obviously, increasing the code rate increases the amount of information transmitted for a fixed block size $N$. However, it is also the case that a reduced code rate allows more information to be transmitted for a constant transmitter power level (see Sec. 2.2.1), and the code rate is therefore also a trade-off between error correction performance and encoder/decoder complexity.
- The normalized SNR at the receiver is $E_b/N_0 = ER/N_0$ and is used instead of the actual SNR $E/N_0$ in order to allow a fair comparison between codes of different rates. The normalized SNR is denoted SNR in the rest of this thesis.
- The bit error rate (BER) is the fraction of differing bits in $m$ and $\hat{m}$, averaged over several blocks.
2.2. CODING THEORY

Figure 2.4 Capacity for binary-input AWGN channel using different SNR.

- The block error rate (BLER) is the fraction of blocks where $m$ and $\hat{m}$ differs.

Coding systems are analyzed in depth in any introductory book on coding theory, e.g. [1, 37].

2.2.1 Shannon bound

In 1948, Claude E. Shannon proved the noisy channel coding theorem [31], that can be phrased in the following way.

For each channel, as defined in Sec. 2.1.1, there is associated a quantity called the channel capacity. The channel capacity is the maximum amount of information, as measured by the shannon unit, that can be transferred per channel use, guaranteeing error-free transmission. Moreover, error-free transmission at information rates above the channel capacity is not possible.

Thus, transmitting information at a rate below the channel capacity allows an arbitrarily low error rate, i.e., there are arbitrarily good error-correcting codes. Additionally, the noisy channel coding theorem states that above the channel capacity, data transmission can not be done without errors, regardless of the code used.

The capacity for the AWGN channel using BPSK modulation and assuming equi-probable inputs is given here without derivation, but calculations are found
e.g. in [1]. It is

\[
C_{BI\text{AWGN}} = \int_{-\infty}^{\infty} f_{\sqrt{2E/N_0}}(y) \log_2 \left( \frac{2f_{\sqrt{2E/N_0}}(y)}{f_{\sqrt{2E/N_0}}(y) + f_{-\sqrt{2E/N_0}}(y)} \right) dy,
\]

(2.8)

where \( f_{\pm\sqrt{2E/N_0}}(y) \) are the probability density functions for Gaussian stochastic variables with means \( \pm \sqrt{E} \) and standard deviation \( \sqrt{N_0/2} \). In Fig. 2.4 the capacity of a binary-input AWGN channel is plotted as a function of the normalized SNR \( E_b/N_0 = ER/N_0 \), and it can be seen that reducing the code rate allows error-free communication using less energy even if more bits are sent for each information bit.

Shannon’s theorem can be rephrased in the following way: for each information rate (or code rate) there is a limit on the channel conditions, above which communication can achieve an arbitrarily low error rate, and below which communication must introduce errors. This limit is commonly referred to as the Shannon limit, and is commonly plotted in code performance plots to show how far the code is from the theoretical limit. The Shannon limit can be found numerically for the binary input AWGN channel by iteratively solving (2.8) for the argument \( \sqrt{E/N_0} \) that yields the desired information rate.

### 2.2.2 Block codes

There are two standard ways of defining block codes: through a generator matrix \( G \) or through a parity-check matrix \( H \). For a message length of \( K \) bits and block length of \( N \) bits, \( G \) has dimensions of \( K \times N \), and \( H \) has dimensions of \( M \times N \), where \( M = N - K \). Denoting the set of code words by \( C \), \( C \) can be defined in the following two ways:

\[
C = \{ x = mG \mid m \in \{0, 1\}^K \}
\]

(2.9)

\[
C = \{ x \in \{0, 1\}^N \mid Hx^T = 0 \}
\]

(2.10)

The most important property of a code regarding performance is the minimum Hamming distance \( d \), which is the minimum number of bits that two code words may differ in. Moreover, as the set of code words \( C \) is linear, it is also the weight of the lowest-weight code word which is not the all-zero code word. The minimum distance is important because all transmission errors with a weight strictly less than \( d/2 \) can be corrected. However, for practical codes \( d \) is often not known exactly, as it is often difficult to calculate theoretically, and exhaustive searches are not realistic with block sizes of thousands of bits. Also, depending on the type of decoder used, the actual error-correcting ability may be both above and below \( d/2 \). Thus the performance of modern codes is usually determined experimentally by simulations over a noisy channel and by measuring the actual bit- or block-error rate at the output of the decoder.
Figure 2.5 Error correcting performance of short codes. The Hamming and Reed-Solomon curves are estimations for hard-decision decoding, whereas the LDPC curves are obtained using simulations with soft-decision decoding.

Figure 2.6 Error correcting performance of long codes.
A simple example of a block code is the \((N,K,d) = (7,4,3)\) Hamming code defined by the parity-check matrix

\[
H = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
\end{pmatrix}.
\] (2.11)

The code has a block length of \(N = 7\) bits, and a message length of \(K = 4\) bits. Thus the code rate is \(R = K/N = 4/7\). It can easily be shown that the minimum-weight codeword has a weight of \(d = 3\), which is therefore the minimum distance of the code. The error correcting performance of this code over the AWGN channel is shown in Fig. 2.5. As can be seen, the code performance is just somewhat better than uncoded transmission. There exists a Hamming code with parameters \((N,K,d) = (2^m - 1,2^m - m - 1,3)\) for every integer \(m \geq 2\), and their parity-check matrices are constructed by concatenating every nonzero \(m\)-bit vector. The advantage of these codes is that decoding is very simple, and they are used e.g. in memory chips.

To decode a received block using Hamming coding, consider for example the \((7,4,3)\) Hamming code and a received vector \(\hat{x}\). Then the syndrome of the received vector is \(H\hat{x}^T\), which is a three-bit vector. If the syndrome is zero, the received vector is a valid codeword, and decoding is finished. If the syndrome is non-zero, the received vector could become a codeword if the bit corresponding to the column in \(H\) that matched the syndrome is flipped. It should thus be noted that the columns of \(H\) contain every non-zero three-bit vector, and thus every received vector \(\hat{x}\) will be at a distance of at most one from a valid codeword. Thus decoding will consist of changing at most one bit, determined by the syndrome if it is non-zero.

To increase the error correcting performance, the code needs to be able to correct more than single bit errors, and then the above decoding technique does not work. While the method could be generalized to determine the bits to flip by finding the minimum set of columns whose sum is the syndrome, this is usually not efficient. Thus the syndrome is usually computed only to determine if a given vector is a codeword or not.

The performance of other short codes are also shown in Fig. 2.5. The Hamming and Reed-Solomon curves are estimations for hard-decision decoding obtained using the MATLAB\textsuperscript{TM} function \texttt{bercoding}. The LDPC codes are randomly constructed \((3,6)\)-regular codes (as defined in Sec. 2.3). Ensembles of 100 codes were generated, and their minimum distances computed using integer linear programming optimization. Among the codes with the largest minimum distances, the codes with the best performance under the sum-product algorithm were selected.

The performance of some long codes are shown in Fig. 2.6. The performance of the \(N = 10^7\) LDPC code is from [10], whereas the performance of the \(N = 10^6\) codes are from [29]. It is seen that at a block length of \(10^6\) bits, LDPC codes perform better than Turbo codes. The \(N = 10^7\) code is a highly optimized irregular LDPC code with variable node degrees up to 200, and performs within 0.04 dB of the Shannon limit at a bit error rate of \(10^{-6}\). At shorter block lengths of 1000-10000 bits, the performance of Turbo codes and LDPC codes are generally comparable.
The (9216, 3, 6) code is a randomly constructed regular code, also used in the simulations in Chapter 6.

For block codes, there are three general ways in which a decoding attempt may terminate:

- **Decoder successful:** The decoder has found a valid codeword, and the corresponding message $\hat{m}$ equals $m$.

- **Decoder error:** The decoder has found a valid codeword, and the corresponding message $\hat{m}$ differs from $m$.

- **Decoder failure:** The decoder was unable to find a valid codeword using the resources specified.

For both the error and the failure result, the decoder has been unable to find the correct sent message $m$. However, the key difference is that decoder failures are detectable, whereas decoder errors are not. Thus, if, for example, several decoder algorithms are available, the decoding could be retried with another algorithm when a decoder failure occurs.

### 2.3 LDPC codes

A low-density parity-check (LDPC) code is a code defined by a parity-check matrix with low density, i.e., the parity-check matrix $H$ has a low number of 1s. It has been shown [16] that there exists classes of such codes that asymptotically reach the Shannon bound with a density tending to zero as the block length tends to infinity. Moreover, the theorem also states that such codes are generated with a probability approaching one if the parity-check matrix $H$ is just constructed randomly. However, the design of practical decoders is greatly simplified if some structure can be imposed upon the parity-check matrix. This seems to often negatively impact the error-correcting performance of the codes, and it is therefore still an active research area how to construct codes which are both theoretically good and practical.

#### 2.3.1 Tanner graphs

LDPC codes are commonly visualized using Tanner graphs [33], because the most common decoding algorithm is defined directly on the graph (see Sec. 2.4). The Tanner graph consists of nodes representing the columns and rows of the parity-check matrix, with an edge between two nodes if the element in the intersection of the corresponding row and column in the parity-check matrix is 1. Nodes corresponding to columns are called variable nodes, and nodes corresponding to rows are called check nodes. As there are no intersections between columns and between rows, the resulting graph is bipartite with all the edges between variable nodes and check nodes. An example of a Tanner graph is shown in Fig. 2.7, and its corresponding parity-check matrix is shown in Fig. 2.8. Comparing to (2.11), it is seen that the matrix is that of the $(7, 4, 3)$ Hamming code.
CHAPTER 2. ERROR CORRECTION CODING

Figure 2.7 Example of Tanner graph for the $(7,4,3)$ Hamming code.

```
<table>
<thead>
<tr>
<th></th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Figure 2.8 Parity-check matrix $H$ for $(7,4,3)$ Hamming code.

Having defined the Tanner graph, there are some properties which are interesting for the decoding algorithm and architecture.

- A **check node regular** code is a code for which all check nodes have the same degree.
- A **variable node regular** code is a code for which all variable nodes have the same degree.
- A $(j,k)$-regular code is a code which is variable node regular with variable node degree $j$ and check node regular with check node degree $k$.
- The **girth** of a code is the length of the shortest cycle in its Tanner graph.
- The **diameter** of a code is the largest distance between two nodes in its Tanner graph.

Using a regular code can simplify the decoder architecture. However, it has also been conjectured [16] that regular codes can not be capacity-approaching under message-passing decoding. The conjecture will be proved if it can be showed that cycles in the code can not enhance the performance of the decoder on average. Furthermore, it has also been showed [9,11,24,30] that codes need to have a wide range of node degree distributions in order to be capacity-approaching. Therefore, assuming that the conjecture is true, there is a trade-off between code performance and decoder complexity regarding the regularity of the code.

The sum-product decoding algorithm for LDPC codes is optimal when the code’s Tanner graph is free of cycles [21]. However, it can also be shown that the
2.4. **SUM-PRODUCT DECODING**

The sum-product decoding algorithm is defined directly on the Tanner graph of the code [16,21,26,36]. It is an iterative algorithm, consecutively propagating bit probabilities and parity-check constraint satisfiability likelihoods until the algorithm

---

2.3.2 Quasi-cyclic LDPC codes

One common way of imposing structure on an LDPC code is to construct the parity-check matrix from equally sized sub-matrices which are either all zeros or cyclically shifted identity matrices. Methods of constructing such types of codes include algebraic methods [19, 28, 34] and geometric methods [20, 23, 34]. These types of codes, denoted quasi-cyclic (QC-LDPC) codes, tend to have decent performance while also allowing the implementation to be efficiently parallelized. The block size may be easily adapted by changing the size of the component sub-matrices. Also, certain construction methods can ensure that the girth of the code is at least 8 [35]. QC-LDPC codes have been chosen as error-correcting codes for several standards, including the WLAN 802.11n [48] standard and the WiMax 802.16e [50] standard.

2.3.3 Randomized quasi-cyclic codes

The performance of regular quasi-cyclic codes can be increased relatively easily by the addition of a randomizing layer in the hardware architecture. This type of codes resulted from an effort of joint code and decoding architecture design [40,42]. The codes are (3, k)-regular, with the general structure shown in Fig. 2.9, and have a girth of at least six. In the figure, I represents $L \times L$ identity matrices, where $L$ is a scaling constant, and $P$ represents cyclically shifted $L \times L$ identity matrices. The column weight is 3, and the row weight is $k$. Thus there are $k^2$ each of the I- and P-type matrices. The bottom part is a partly randomized matrix, also with row weight $k$. The submatrix is obtained from a quasi-cyclic matrix by moving some of the ones within their columns according to certain constraints. The constraints are best described directly by the decoder implementation, described in Sec. 5.1. This type of codes has been used for the hardware implementations in this thesis, with $k = 6$ which results in rate-1/2 codes.

2.4 Sum-product decoding

The sum-product decoding algorithm is defined directly on the Tanner graph of the code [16,21,26,36]. It is an iterative algorithm, consecutively propagating bit probabilities and parity-check constraint satisfiability likelihoods until the algorithm

---

graph must contain cycles for the code to have more than minimal error correcting performance [15]. Specifically, it is shown that for a code $C$ with parameters $(N, K, d)$ and rate $R = K/N$, the following conditions apply. If $R \geq 0.5$, then $d \leq 2$, and if $R < 0.5$, then $C$ is obtained from a code with $R \geq 0.5$ and $d \leq 2$ by repetition of certain symbols. Thus, as cycles are needed for the code to have good theoretical properties, but also inhibit the performance of the practical decoder, the concept of girth is important. Using a code with large girth and small diameter is generally expected to improve the performance, and codes are therefore usually designed so that the girth is at least six. However, the importance of increasing the girth further has not been scientifically proved.
converges to a valid code word, or a predefined maximum number of iterations is reached. A number of variables are defined:

- The prior probabilities $p_n^0$ and $p_n^1$ denote the probabilities that bit $n$ is zero and one, respectively, considering only the received channel information and not the code structure.

- The variable-to-check messages $q_{nm}^0$ and $q_{nm}^1$ are defined for each edge between a variable node $n$ and a check node $m$. They denote the probabilities that bit $n$ is zero and one, respectively, considering the prior variable probabilities and the likelihood that parity-check relations other than $m$ involving bit $n$ are satisfied.

- The check-to-variable messages $r_{mn}^0$ and $r_{mn}^1$ are defined for each edge between a check node $m$ and a variable node $n$. They denote the likelihoods that parity-check relation $m$ is satisfied considering variable probabilities for the other involved bits given by their variable-to-check messages, and given that bit $n$ is zero and one, respectively.

- The pseudo-posterior probabilities $q_n^0$ and $q_n^1$ are updated in each iteration and denote the probabilities that bit $n$ is zero and one, respectively, considering the information propagated so far during the decoding.

- The hard-decision vector $\hat{x}_n$ denotes the most likely bit values, considering bit $n$ and its surrounding. The number of surrounding bits considered increases with each iteration.

**Figure 2.9** Parity-check matrix structure of randomized quasi-cyclic codes through joint code and decoder architecture design.
2.4. SUM-PRODUCT DECODING

Decoding a received vector consists of three phases: initialization phase, variable node update phase, and check node update phase. In the initialization phase, shown in Fig. 2.10, the messages are cleared and the prior probabilities are initialized to the individual bit probabilities based on received channel information. In the variable node update phase, shown in Fig. 2.11, the variable-to-check messages are computed for each variable node from the prior probabilities and the check-to-variable messages along the adjoining edges. Also, the pseudo-posterior probabilities are calculated, and the hard-decision bits are set to the most likely bit values based on the pseudo-posterior probabilities. In the check node update phase, shown in Fig. 2.12, the check-to-variable messages are computed based on
the variable-to-check messages, and all check node relations are evaluated based on the hard-decision vector. If all check node constraints are satisfied, decoding stops, and the current hard-decision vector is output.

Decoding continues until either a valid code-word is found, or a preset maximum number of iterations is reached. In the latter case, a decoding failure occurs, whereas the former case results in either a decoder success or a decoder error. However, for well-defined codes with block lengths of at least 1000 bits, decoder errors are extremely rare. Therefore, when a decoding attempt is unsuccessful, it will almost always be known.

Decoding is usually performed in the log-likelihood ratio domain using the variables \( \gamma_n = \log(p_0^n/p_1^n) \), \( \alpha_{nm} = \log(q_{nm}^0/q_{nm}^1) \), \( \beta_{mn} = \log(r_{mn}^0/r_{mn}^1) \) and \( \lambda_n = \log(q_n^0/q_n^1) \). In this domain, the variable update equations can be written [21]

\[
\alpha_{nm} = \gamma_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \beta_{mn} \tag{2.12}
\]

\[
\beta_{mn} = \left( \prod_{n' \in \mathcal{N}(m) \setminus m} \text{sign} \alpha_{nm} \right) \cdot \Phi \left( \sum_{n' \in \mathcal{N}(m) \setminus m} \Phi (|\alpha_{nm}|) \right) \tag{2.13}
\]

\[
\lambda_n = \gamma_n + \sum_{m' \in \mathcal{M}(n)} \beta_{mn} \tag{2.14}
\]

where \( \mathcal{M}(n) \) denotes the neighbors to variable node \( n \), \( \mathcal{N}(m) \) denotes the neighbors to check node \( m \), and \( \Phi(x) = \log \tanh(x/2) \).

2.5 Hardware implementation

To achieve good theoretical properties, the code is typically required to have a certain degree of randomness or irregularity. However, this makes efficient hardware implementations difficult [38]. For example, a direct instantiation of the Tanner graph of a 1024-bit code in a 0.16 \( \mu m \) CMOS process resulted in a chip with more
2.5. HARDWARE IMPLEMENTATION

than 26000 wires with an average length of 3 mm, and a routing overhead of 50% [8, 18]. It is also the case that the required numerical accuracy of the computations is low. Thus, the sum-product algorithm can be said to be communication-bound rather than computation-bound.

The architectures for the sum-product decoding algorithm can be divided into three main types [17]: the parallel, the serial, and the serial/parallel (or partly parallel) architecture. These are briefly described here.

2.5.1 Parallel architecture

Directly instantiating the Tanner graph of the code yields the parallel architecture, as shown in Fig. 2.13. As the check node computations, as well as the variable
node computations, are intraindependent (i.e. the check node computations depend only on the result of variable node computations, and vice versa), the algorithm is inherently parallelizable. All check node computations can be done in parallel, followed by computations of all the variable nodes. An example implementation is the above mentioned 1024-bit code decoder, achieving a throughput of 1 Gb/s while performing 64 iterations. The chip has an active area of 52.5 mm$^2$ and a power dissipation of 690 mW, and is manufactured in a 0.16 µm CMOS process [8]. However, due to the graph irregularity required for good codes, the parallel architecture is hardly scalable to larger codes. Also, the irregularity of purely random codes makes it difficult to time-multiplex the computations efficiently.

2.5.2 Serial architecture

Another obvious architecture is the serial architecture, shown in Fig. 2.14. In the serial architecture, the messages are stored in a memory between generation and consumption. Control logic is used to schedule the variable node and check node computations, and the code structure is realized through the memory addressing. However, in a code with good theoretical properties, the sets of check-to-variable node messages that a set of variable nodes depend on are largely disjunctive (e.g. in a code with girth six, at most one check-to-variable message is shared between the dependencies of any two variable nodes), which makes an efficient code schedule difficult and requires that the memory contains most of the messages. Moreover, in a general code, increasing the throughput by partitioning the memory is made difficult by the irregular dependencies of node computations, although certain code
constructions methods (e.g. QC-LDPC codes) can ensure that such a partitioning can be done. Still, the performance of the serial architecture is likely to be severely limited by memory accesses. In [39], iteration-level loop unrolling was used to achieve 1 Gb/s throughput of a serial-like decoder for an \((N, j, k) = (4608, 4, 36)\) code, but with memory requirements of 73728 words per iteration.

### 2.5.3 Partly parallel architecture

A third possible architecture is the serial/parallel, or partly parallel architecture, shown in Fig. 2.15, which can be seen as either a time-multiplexed parallel decoder, or an interleaved serial decoder. The serial/parallel architecture retains the speed achievable with parallel processing, while also allowing longer codes without resulting in excessive routing. However, neither the parallel nor the serial architecture can usually be efficiently transformed using a general random code. Thus, the use of a serial/parallel architecture usually requires using a joint code and decoder design flow. Generally, the QC-LDPC codes (see Sec. 2.3.2) obtained through various techniques are suitable to be used with a serial/parallel architecture. Examples of this kind of architecture include a 3.33 Gb/s (1200, 720) code decoder with a power dissipation of 644 mW, manufactured in a 0.18\( \mu \text{m} \) technology [22], and a 250 Mb/s (1944, 972) code decoder dissipating 76 mW, manufactured in a 0.13\( \mu \text{m} \) technology [32]. The implementation in this thesis is based on a serial/parallel decoder using QC-LDPC codes with an additional scrambling layer to increase the code irregularity and performance. An FPGA implementation of this architecture achieves a throughput of 54 Mbps for a (9216, 4608) code, using a clock frequency of 56 MHz and performing 18 iterations [41].

### 2.5.4 Finite wordlength considerations

Earlier investigations have shown that the sum-product algorithm for LDPC decoding have relatively low requirements on data wordlength [43]. Even using as few as 4-6 bits yields only a fraction of a dB as SNR penalty for the decoder performance. Considering the equations (2.12)–(2.14), the variable node computations, (2.12) and (2.14), are naturally done using two’s complement arithmetic, whereas the check node computations (2.13) are more efficiently carried out using signed-magnitude arithmetic. This is the solution chosen for the finite-wordlength implementations in this thesis, and data representation converters are therefore used between the variable node and check node processing elements. It should also be noted that due to the inverting characteristic of the domain transfer function \(\Phi(x)\), shown in Fig. 2.16, there is a big difference between positive and negative zero in the signed-magnitude representation, as the sum in (2.13) is given directly as an argument to \(\Phi(x)\). This makes the signed-magnitude representation particularly suited for the check-node computations.

Considering \(\Phi(x)\), the fixed-point implementation is not trivial. The functions obtained through rounding of the function values are shown for some different data representations in Fig. 2.16. Whereas other quantization rules such as truncation,
**Figure 2.16** Figure showing $\Phi(x)$ and discrete functions obtained through rounding. For the discrete functions, the format is $(w_i, w_f)$, where $w_i$ is the number of integer bits and $w_f$ is the number of fractional bits.

Rounding towards infinity, or arbitrary rules obtained through simulations can be considered, rounding to nearest has been used in this thesis, and the problem is not further considered. However, it can be noted that because of the highly non-linear nature of $\Phi(x)$, many numbers do not occur as function values for the fixed-point implementations. This fact can be exploited, and in Sec. 4.2 a compression scheme is introduced.

### 2.5.5 Scaling of $\Phi(x)$

Considering the $\Phi(x)$ function in (2.13), using the natural base for the logarithm is not necessary. However, when the natural base is used, the inverse function $\Phi^{-1}(x) = 2\arctanh \exp(x)$ is identical to $-\Phi(x)$, and thus in (2.13) the inverse transformation can be done using $\Phi(x)$. However, the arithmetic transformation of the check node update rule to a sum of magnitudes work equally well with any other logarithm base. The resulting difference between the forward transformation
function $\Phi(x)$ and the reverse transformation function $\Phi^{-1}(x)$ may or may not be a problem in the implementation. In a fixed-point implementation, changing the logarithm base can be seen as a scaling of the inputs and outputs to the CNU, which can often be done without any overhead if separate implementations are already used for the forward and reverse transformation functions. In [12], it is shown that such a scaling can improve the performance of the sum-product decoder using fixed-point data. In Sec. 4.2, it is shown how the benefits of internal data coding depend on the choice of logarithm base.
In this chapter, a modification to the sum-product algorithm that the authors call the early-decision decoding algorithm is introduced. Basically, the idea is to reduce internal communication of the decoder by early decision of bits that already have a high enough probability of being either zero or one. This is done in several steps: by defining a measure of bit reliabilities, by defining how bits are decided based on their reliabilities, and by defining the processing of decided bits in the decoding process.

Secs. 3.1.1, 3.1.2, 3.1.3 and 3.2 have previously been published in [4], [7], [3] and [5], whereas the contents of Secs. 3.1.4 and 3.1.5 are previously unpublished.

3.1 Early-decision algorithm

During the first iterations of the decoding of a block, when the messages are still independent, the probability that the hard-decision variable is wrong is \( \min(q_0^n, q_1^n) \). Assuming that this value is small, which is the case in the circumstances considered, it can be approximated as \( \min(q_0^n/q_1^n, q_1^n/q_0^n) \), which can be rewritten as \( \exp(-|\lambda_n|) \). Thus, a measure of the reliability of a bit during decoding can be defined as

\[
c_n = |\lambda_n|, \quad (3.1)
\]
(a) \((N, j, k) = (144, 3, 6)\), successful decoding.

(b) \((N, j, k) = (144, 3, 6)\), unsuccessful decoding.

(c) \((N, j, k) = (1152, 3, 6)\), successful decoding.

(d) \((N, j, k) = (1152, 3, 6)\), unsuccessful decoding.

**Figure 3.1** Typical reliabilities \(c_n\) during decoding of different codes. The gray lines show the individual bit reliabilities, and the thick black lines are the magnitude averages.
3.1. EARLY-DECISION ALGORITHM

with the interpretation that the hard-decision bit is correct with a probability of $1 - \exp(-c_n)$. Typical values of $c_n$ during decoding of a block are shown for two different codes in Fig. 3.1. The slowly increasing average reliabilities in Figs. 3.1(a) and 3.1(c) are typical for successfully decoded blocks. The slope is generally steeper for longer codes, as the reliabilities escalate quickly in local parts of the code graph where the decoder has converged. Similarly, the oscillating behavior in Figs. 3.1(b) and 3.1(d) is common for unsuccessfully decoded blocks. The key point to recognize in these figures is that high reliabilities are unlikely to change sign, i.e., their corresponding bits are unlikely to change value.

Early decision decoding is introduced through the definition of a threshold $t$ denoting the minimum required reliability of a bit to consider it to be sufficiently well determined. The threshold is in the simplest case just a constant, but may also be a function of the iteration, the node degree, or other values. Different choices of the threshold are discussed in Sec. 3.1.1.

When a bit is decided, incoming messages to the corresponding node are ignored, and a fixed value is chosen for the outgoing messages. The value will be an approximation of the actual bit probability and will affect the probability computations of bits in subsequent iterations. Different choices of values for decided bits are discussed in Sec. 3.1.2.

Early decision adds an alternative condition for finishing the decoding of a block, which is when a decision has been made for all bits. However, when the decoder makes an erroneous decision, this tends to lock the algorithm by rendering its adjacent bits undecidable as the bit values in the graph become inconsistent. In Sec. 3.1.4, detecting the introduced inconsistencies in an implementation-friendly way is discussed.

3.1.1 Choice of threshold

The simplest choice of a threshold is a constant $t = t_0$. In a cycle-free graph this is a logical choice, as the pseudo-posterior probabilities $q_n$ are valid. However, in a graph with cycles, the messages will be correlated after $g/4$ iterations, where $g$ is the girth of the code. As the girth is often at most 6 for codes used in practice, this will be already after the first iteration. Detailed analysis of the impact of the correlation of messages on the probabilities is difficult to do. However, as the pseudo-posterior probabilities of a cycle-less graph increase with the size of the graph, it can be assumed that the presence of cycles causes an escalation of the pseudo-posterior probabilities. Thus a dynamic threshold is defined as $t = t_0 + t_d i$, where $i$ is the current iteration. However, no attempt is made to justify this definition in this thesis. Dynamic thresholds are investigated in [3], but no results are presented in this thesis as the early saturation of the magnitudes of the messages in fixed-point representations makes it difficult to achieve gains in a practical implementation.
3.1.2 Handling of decided bits

When computations of probabilities stop for a bit, the outgoing messages from the node will not represent the correct probabilities, and subsequent computations for nearby nodes will therefore be done using incorrect data. The most straightforward way is to stop computing the variable-to-check messages, and use the values from the iteration in which the bit was decided in subsequent iterations. However, messages must still be communicated along the edges, so the potential gain is small in this case.

There are two other natural choices of variable-to-check messages for early decided bits. One is to set the message $\alpha_{nm}$ to $t$ or $-t$, depending on the hard-decision value. The other is to set $\alpha_{nm}$ to $\infty$ or $-\infty$, thereby essentially regarding the bit as known during the subsequent iterations. The first choice can be expected to introduce the least amount of noise in the decoding, and thereby yield more accurate decoding results, whereas the second choice can be expected to propagate increased reliabilities and thus decide bits faster. Similar ideas are presented in [45]. In this thesis, results are only provided for the second case, as simulations have shown the difference in performance to be insignificant. Results are provided in Sec. 6.1.

3.1.3 Bound on error correction capability

Considering early decision applied only on the pseudo-posterior probabilities before the first iteration, a lower bound on the error correction capability over a white Gaussian channel with standard deviation $\sigma$ can be calculated. Before any messages have been sent, the pseudo-posterior probabilities are equal to the prior probabilities, $q_n = p_n$. An incorrect decision will be made if the reliability of a bit is above the threshold, but the hard decision bit is not equal to the sent bit. The probability of this occurring can be written

$$B_{\text{bit}} = P(c_n > t, \hat{x}_n \neq x_n) = P(\gamma_n > t, x = -1) + P(\gamma_n < -t, x = 1). \quad (3.2)$$

Assuming that the bits 0 and 1 are equally probable, the error probability can be rewritten

$$B_{\text{bit}} = P(\gamma_n > t|x = -1) P(x = -1) + P(\gamma_n < -t|x = 1) P(x = 1) =$$

$$= P(\gamma_n < -t|x = 1) = P(r_n < -\frac{t\sigma^2}{2}|x = 1) =$$

$$= Q\left(-\frac{t\sigma^2}{2}\right). \quad (3.3)$$

If no corrections of erroneous decisions are done, an incorrect bit decision will result in the whole block being incorrectly decoded. The probability of this happening for a block with $N$ bits is

$$B_{\text{block}} = \left(1 - (1 - B_{\text{bit}})^N\right) \quad (3.4)$$
3.1. EARLY-DECISION ALGORITHM

3.1.4 Enforcing check constraints

When an erroneous decision is made, usually the decoder will neither converge on a codeword nor manage to decide every bit. Thus the decoder will continue to iterate the maximum number of iterations, and if erroneous decisions are frequent, the average number of iterations can increase significantly and severely reduce the throughput of the decoder. However, if the graph inconsistency can be discovered, the decoder can be aborted early and the block can be retried using a less approximative algorithm.

The principle of how graph inconsistencies can be discovered is shown in Fig. 3.2.
In the example, bits $v_0, \ldots, v_4$ have been decided. Thus, the incoming messages $\alpha_{0,0}, \ldots, \alpha_{4,0}$ to check node $c_0$ will not change, and the message $\beta_{0,5}$ from check node $c_0$ to variable node $v_5$ will also be constant. As the decided conditions are passed along with the messages $\alpha_{0,0}, \ldots, \alpha_{4,0}$, this is known to the check node, and an enforcing condition can therefore be flagged for $\beta_{0,5}$, meaning that the receiving variable node must take the value of the message for the graph to remain consistent. Similarly, variable nodes $v_6, \ldots, v_{10}$ are also decided, resulting in that message $\beta_{1,5}$ is also enforcing. However, to satisfy check node $c_0$, $v_5$ must take the value 0, and to satisfy $c_1$, $v_5$ must take the value 1, and as the variable nodes $v_0, \ldots, v_4, v_6, \ldots, v_{10}$ can not change, an incorrect decision has been discovered.

However, it is not known exactly which bit has been incorrectly decided, and thus the error can not be simply corrected. In this thesis, the event is handled by aborting the decoding process, but other options are also possible, e.g., to remove all decisions and continue with the current extrinsic information, or to use the current graph state as a starting point for the general sum-product algorithm.

As an enforcing check constraint requires that the receiving variable node must take the value of the associated check-to-variable message for the graph to remain consistent, an early decision can be made for the receiving variable node. In floating-point simulations with large dynamic range this is not necessary as the large magnitude of the check-to-variable message ensures that a decision will be made based on the pseudo-posterior probability. However, in fixed-point implementations, a decision may have to be explicitly made as the magnitude of the check-to-variable message is limited by the data representation.

### 3.1.5 Enforcing check approximations

In a practical implementation, using an extra bit to denote enforcing checks is relatively costly, and thus approximations using the absolute value of the check-to-variable messages are suggested. Consider the check node $c_0$ with $k$ inputs $v_0, \ldots, v_{k-1}$, where $v_0, \ldots, v_{k-2}$ have been decided. Assume that the values $\pm z$ are used for the messages from the decided variables. The absolute value of the check-to-variable message $\beta_{0,k-1}$ from $c_0$ to $v_{k-1}$ will then be

$$|\beta_{0,k-1}| = \left| \left( \prod_{n=0}^{k-2} \text{sign} \alpha_{n,0} \right) \cdot \Phi \left( \sum_{n=0}^{k-2} \Phi \left( |\alpha_{n,0}| \right) \right) \right|$$

$$= 2 \arctanh \exp \left( \sum_{n=0}^{k-2} \log \tanh \left( \frac{z}{2} \right) \right)$$

$$= 2 \arctanh \left( \tanh \left( \frac{z}{2} \right)^{k-1} \right). \quad (3.5)$$

Using this approximation to determine enforcing checks, the check-to-variable message $\beta_{mn}$ from a $k$-input check node is enforcing if

$$|\beta_{mn}| \geq 2 \arctanh \left( \tanh \left( \frac{z}{2} \right)^{k-1} \right). \quad (3.6)$$
In a fixed-point implementation, the condition can be written

\[ |\beta_{mn}| \geq \Phi[(k - 1)\Phi[z]], \]

where \( \Phi[x] \) denotes the discrete function obtained through rounding of the argument and function values of \( \Phi(x) \), as in Fig. 2.16. Typically, \( z \) is the maximum representable value in the data representation, and thus \( \Phi[z] = 0 \), and \( \Phi[(k - 1)\Phi[z]] \) is also the maximum representable value. However, due to the limited dynamic range of the data, \( z \) will be a common value also for variables that are not decided. Thus check-to-variable messages will often be enforcing when the involved variables are not decided, which inhibits performance. An alternative implementation is described in Sec. 5.2.2.

### 3.2 Hybrid decoding

For the sum-product algorithm, undetected errors are extremely rare, and this property is retained with the early-decision modification. Thus, it is almost always detectable that a wrong decision has been made, but not for which bit. However, the unsuccessfully decoded block can be retried using the regular sum-product algorithm. As the block is then decoded twice, the resources required for that block will be significantly increased, and there is therefore a trade-off adjusted by the threshold level. Lowering the threshold will increase the number of bit decisions, but also increase the probability that a block is unsuccessfully decoded and will require an additional decoding pass. Determining the optimal threshold level analytically is difficult, and in this thesis simulations are used for a number of codes to determine its impact.

Using an adequate threshold, redecoding a block is relatively rare, and thus the average decoding latency is only slightly increased. However, the maximum latency is doubled, which might make the hybrid decoding algorithm unsuited for applications sensitive to the communication latency.

Results for the hybrid algorithm are provided in Sec. 6.1.3.
In this chapter, the representation of the data in a fixed-point decoder implementation is discussed. It is showed that the usual data representation is redundant, and that in many cases coding of the data can be applied to reduce the width of the data buses without sacrificing the error-correcting performance of the decoder.

The results in this chapter have previously been published in [2].

4.1 Fixed wordlength

The sum-product algorithm, as formulated in the log-likelihood domain in (2.12)–(2.14), lends itself well towards fixed wordlength implementations. Usually, decent performance is achieved using wordlengths as short as 4–6 bits [43]. Thus, the additions can be efficiently implemented using ripple-carry adders, and the domain transfer function $\Phi(x)$ can be implemented using direct gate-level synthesis. Simulations of the performance degradation using different wordlengths are provided in Sec. 6.2.

4.2 Data compression

The function $\Phi(x)$ is shown in Fig. 4.1. In a hardware implementation, usually the output of this function is stored in memories between the phases, and it is therefore
Chapter 4. Data Representations

Figure 4.1 The domain transfer function $\Phi(x)$.

Figure 4.2 Data-path in a sum-product decoder.

Figure 4.3 Data-path in a sum-product decoder with compression.
of interest to represent the information efficiently. However, as the function is very non-linear, all possible words will not be represented at the output, and there is therefore sometimes an opportunity to code the data.

A general model of the data flow in a sum-product decoder is shown in Fig. 4.2. The VNU block performs the computation of the $\alpha_{nm}$ messages in (2.12), whereas the CNU block performs the additions in the $\beta_{mn}$ messages in (2.13). $\Phi(x)$ is implemented by the LUT blocks, and can be part of either the VNU block or the CNU block. The dashed lines 1 and 2 show the common partitions used in decoder implementations. Instead of these partitions, the implementation in Fig. 4.3 is possible, where an encoder is used to convert the messages to a more compact representation which is used to communicate the messages between the processing units. At the destination, a decoder is used to convert the coded data back to the original message.

Denoting the number of integer bits in the uncoded data representation by $w_i$, and the number of fractional bits by $w_f$, the number of redundant bits in the

**Figure 4.4** Redundant bits $B(w_i, w_f)$ of LUT value entries for different data representations.
output of the LUT blocks can be written
\[
B(w_i, w_f) = \log_2 (2^{w_i} + 2^{w_f} - N_o),
\]
where \(N_o\) is the number of unique function values in the LUT. \(B(w_i, w_f)\) is plotted for different parameters in Fig. 4.4.

Obviously \(B(w_i, w_f)\) also depends on the logarithm base in \(\Phi(x)\) described in Sec. 2.5.5. However, as the domain transfer functions differ when the logarithm is not the natural logarithm, \(B(w_i, w_f)\) will be different for the two LUTs. The representation with \(w_i = 2\) and \(w_f = 2\) was shown in [43] to be a good trade-off between the performance and complexity of the decoder, and the number of redundant bits as a function of the logarithm base is shown in Fig. 4.5. As shown, compression can be combined with logarithm scaling for bases from 2.4 to 3.2 for \(w_f = 2\), for the considered representation, without precision loss. Outside this interval, approximation of the domain transfer function is needed in one direction.
To estimate the power dissipation savings obtainable with the methods in this thesis, the algorithms have been implemented in VHDL and synthesized to a Xilinx Virtex 5 FPGA. As a basis for the architecture, the FPGA implementation in [41] has been used. The codes that can be decoded by the architecture are similar to QC-LDPC codes, but a configurable interconnection networks allows some degree of randomness, resulting in the type of codes described in Sec. 2.3.3. The reference sum-product decoder is described in Sec. 5.1 and the modifications done to implement the early decision algorithm is described in Sec. 5.2.

The implementation of the early decision algorithm has previously been published in [6].

5.1 Sum-product reference decoder architecture

In this section, the sum-product reference decoder is presented. The section includes an architecture overview, description of memory blocks and implementations of variable node and check node processing units, as well as the interconnection networks.
5.1.1 Architecture overview

An overview of the architecture is shown in Fig. 5.1. The architecture contains \( k^2 \) memory blocks with the memory banks used during decoding and the variable node processing unit. The memory blocks are connected to \( 3k \) check node processing units through two regular and fixed interconnection networks (\( \pi_1 \) and \( \pi_2 \)), and one randomized and configurable interconnection network (\( \pi_3 \)). The purpose of the variable node units is to perform the computations of the variable-to-check node messages \( \alpha_{nm} \), as in (2.12), and the pseudo-posterior probabilities \( \lambda_n \), as in (2.14). Similarly, the purpose of the check node units is to perform the computation of the check-to-variable messages \( \beta_{mn} \), as in (2.13), and to compute the check parities using the hard-decision variables \( \hat{x}_n \) obtained from the pseudo-posterior probabilities. However, in the implementation, the computation of \( \Phi(\alpha_{nm}) \) is moved from the check node units to the variable node units, and thus the implementation deviates slightly from the formal formulation of the algorithm.
5.1. Memory block

The structure of the memory blocks is shown in Fig. 5.2, where $w$ denotes the wordlength of the prior probabilities $\gamma_n$. One memory block contains one INT_RAM of width $w$ for storing $\gamma_n$, one DEC_RAM of width 1 to store the hard-decision values $\hat{x}_n$, and three EXT_RAMs of width $w+1$ to store the extrinsic information $\alpha_{nm}$ and $\beta_{mn}$ during decoding. Thus, there are a total of $5k^2$ memories. All memories are of size $L$. The EXT_RAMs are dual-port memories, read and written every clock cycle in both the variable node update phase and the check node update phase, whereas the INT_RAM and DEC_RAM are single-port memories used only in the variable node update phase. The messages are stored in two’s complement format in INT_RAM and in signed magnitude format in EXT_RAMs. On the output of the EXT_RAMs, there are two registers. Thus switching of signals in the CNU can be disabled in the VNU phase and vice versa.

5.1.3 Variable node processing unit

The implementation of the variable node processing unit is relatively straightforward, as shown in Fig. 5.3. As the extrinsic information is stored in signed magnitude format, the data are first converted to two’s complement. An adder network performs the computations of the variable-to-check node messages and the pseudo-posterior probabilities, and the variable-to-check node messages are then converted back to signed magnitude format and truncated to $w$ bits. Finally, the $\Phi(x)$ function (see Fig. 2.16) of the magnitude is computed, and the results are joined with the hard-decision bit to form the $w+1$-bit hybrid data.
5.1.4 Check node processing unit

The implementation of the check node processing unit for $k = 6$ is shown in Fig. 5.4. First, the hard-decision bits are extracted and xored to compute the parity of the check constraint. Then the check-to-variable messages are computed using the function $S(x, y)$ defined as

$$S(x, y) = \text{sign}(x) \cdot \text{sign}(y) \cdot (|x| + |y|).$$

(5.1)

Finally, the results are truncated to $w$ bits, and the $\Phi(x)$ function of the magnitude is computed.
Figure 5.4 Implementation of check node processing unit for the \((3, k)\)-regular sum-product decoder.
5.1.5 Interconnection networks

The functions of the fixed interconnection networks are shown in Figs. 5.5 and 5.6. \( \pi_1 \) connects the messages from Block \((x,y)\) with the same \(x\) coordinate to the same CNU, whereas \( \pi_2 \) connects the messages with the same \(y\) coordinate to the same CNU.

The implementation of the configurable interconnection network \( \pi_3 \) is shown in Fig. 5.7 for the forward path. \( a_{m,n} \) denotes the message from Block \(m,n\). \( a_{m,n} \) are permuted to \( b_{m,n} \) by the permutation functions \( \Psi_{l,n} \), where \( l \) is the value of AG\(^1\), defined in Sec. 5.1.6. \( \Psi_{l,n} \) is either the identity permutation or the fixed permutation \( R_n \) depending on \( l \) and \( n \). Thus, formally

\[
b_{m,n} = \begin{cases} 
a_{m,n} & \text{if } \psi_{l,n} = 0 \\ 
a_{R_n(m),n} & \text{if } \psi_{l,n} = 1 \end{cases},
\]

where \( \psi_{l,n} \) are values stored in a ROM. Similarly, \( b_{m,n} \) are permuted to \( c_{m,n} \) through the permutation functions \( \Omega_{l,m} \), which are either the identity permutations or the fixed permutations \( C_m \) depending on \( l \) and \( m \). The permutation can
Figure 5.7 Forward path of $\pi_3$ interconnection network, with $a_{m,n}$ denoting the message from memory block $(m,n)$. $\Psi_{l,n}$ and $\Omega_{l,m}$ denote permutation functions that are either the identity permutation or fixed permutations $R_n$ and $C_m$, respectively.
be formalized

$$c_{m,n} = \begin{cases} b_{m,n} & \text{if } \omega_{l,m} = 0 \\ b_{m,C_{m}(n)} & \text{if } \omega_{l,m} = 1 \end{cases}$$

(5.3)

where $\omega_{l,m}$ are values stored in a ROM. Finally, the values $c_{m,n}$ are connected to CNU$_{3,n}$.

### 5.1.6 Memory address generation

The addresses to the EXT_RAM$_i$ memories are given by mod-$L$ binary counters $AG^i_{x,y}, i = 1, 2, 3$, where $AG^1$ is also used for the INT_RAM and DEC_RAM memories. The counters are initialized to 0 at the beginning of variable node processing, and to the values $C^i_{x,y}$ at the beginning of check node processing, which are chosen according to the constraints

$$C^1_{x,y} = 0$$

(5.4)

$$C^2_{x,y} = ((x - 1) \cdot y) \mod L$$

(5.5)

$$C^3_{x,y_1} \neq C^3_{x,y_2}, \forall y_1, y_2 \in \{1, \ldots, k\}, y_1 \neq y_2$$

(5.6)

$$C^3_{x_1,y} - C^3_{x_2,y} \neq ((x - 1) \cdot y) \mod L, \forall x_1, x_2 \in \{1, \ldots, k\}, x_1 \neq x_2.$$  

(5.7)

The purpose of the constraints are to ensure that the code results in a code with girth of at least six, and are further motivated in [41].

The architecture as described results in the code shown in Fig. 5.8, where $I_{x,y}$ denotes an $L \times L$ identity matrix, $P_{x,y}$ denotes an $L \times L$ identity matrix circularly right-shifted by $C^2_{x,y}$, and $R_{x,y}$ denotes an $Lk \times L$ matrix with column weight one and row weight at most one, subject to further constraints imposed by $C^3_{x,y}$ and $\pi_3$. 

**Figure 5.8**

![Figure 5.8](image)
The extrinsic data corresponding to $I_{x,y}$ is stored in EXT_RAM1 in Block$_{x,y}$, the data corresponding to $P_{x,y}$ in EXT_RAM2 in Block$_{x,y}$, and the data corresponding to $R_{x,y}$ in EXT_RAM3 in Block$_{x,y}$.

### 5.1.7 Φ function

The definition of the $Φ[x]$ function used in the sum-product reference decoder architecture is shown in Table 5.1. The same function is used for the early decision decoder without enforcing check constraints.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$Φ[x]$</th>
<th>$x$</th>
<th>$Φ[x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.1 Definition of $Φ[x]$ function for $(w_i, w_f) = (3, 2)$ fixed point data representation.

The early-decision decoder architecture

In this section, the changes made to the sum-product reference decoder to realize the early decision decoder are explained. In the early decision decoder architecture, additional registers are introduced which are not shown in the reference architecture. However, as the implementation is pipelined in several stages, the registers are present also in the reference architecture, and do not therefore constitute additional hardware resources for the early decision architecture.

### 5.2 Early-decision decoder architecture

In this section, the changes made to the sum-product reference decoder to realize the early decision decoder are explained. In the early decision decoder architecture, additional registers are introduced which are not shown in the reference architecture. However, as the implementation is pipelined in several stages, the registers are present also in the reference architecture, and do not therefore constitute additional hardware resources for the early decision architecture.

#### 5.2.1 Memory block

In the early-decision decoder architecture, additional memories DIS_RAM of size $L/8 \times 8$ bits are added to the memory blocks to store the early decisions, as seen in Fig. 5.9. The reason that the memories are implemented with a wordlength of eight is that decisions must be read for all CNUs in the check node update phase, which would thus require three-port memories if a wordlength of one is used. However, as data are always read sequentially due to the quasi-cyclic structure of the codes, parallelizing the memory accesses by increasing the wordlength is easy. In addition, the DEC_RAMS have been changed to the same size to be able to utilize the same addressing mechanism. The parallelized accesses to the hard decision and early
decision memories are handled by the ED_LOGIC block which uses an additional output signal from the VNU to determine early decisions.

In the VNU update phase, the EXT_RAMs are addressed with the same addresses and the three-bit disabled signal from the ED_LOGIC block is identical for all memories. Thus, when a bit is disabled, the registers on the memory outputs are disabled, and the VNU will be idle. In the CNU update phase, the wordlength of the signals from the EXT_RAMs to the CNUs have been increased by one to include the disabled bits. As data are read from different memory positions in the three memories, the disabled bits are not necessarily identical. When a bit is disabled, the registers on the memory outputs are disabled, and only the hard decision bit is used in the CNU computations.

5.2.2 Node processing units

In the variable node processing unit, shown in Fig. 5.10, the magnitude of the pseudo-posterior likelihood is compared to the threshold value to determine if the bit will be disabled in further iterations.

In the check node processing unit, shown in Fig. 5.11, MUXes at the input determine if the variable-to-check messages or the hard decision bits are used for the computations. At the CNU outputs, registers are used to disable driving of the check-to-variable messages for bits that are disabled.

If enforcing check constraints, as defined in Sec. 3.1.4, are implemented, additional changes are made to the VNU and CNU. These are described in Sec. 5.2.4.
5.2. EARLY-DECISION DECODER ARCHITECTURE

Figure 5.10 Implementation of variable node processing unit for the $(3, k)$-regular early-decision decoder.

5.2.3 Early decision logic

The implementation of the ED_LOGIC block is shown in Fig. 5.12, and consists mostly of a set of shift registers. During the VNU phase, hard_decision and early_decision bits are read from DEC_RAM and DIS_RAM and stored in one shift register each. Newly made hard decisions and early decisions arrive from the VNU unit. However, for the bits that are disabled, the VNU computations are turned off, and thus the stored values from previous iterations are recycled. During the CNU phase, the memories are only read. However, the three address generators produce different addresses, and thus three shift registers with different contents are used. Because of this implementation, an additional constraint is imposed on the code structure. As DEC_RAM and DIS_RAM are only single-port memories, simultaneous reading from several memory locations is not allowed. The implication of this
Figure 5.11 Implementation of check node processing unit for the $(3,k)$-regular early-decision decoder.
is the following three constraints:

\[
\begin{align*}
C^{1}_{x,y} &\neq C^{2}_{x,y} \pmod{8} \\
C^{2}_{x,y} &\neq C^{3}_{x,y} \pmod{8} \\
C^{1}_{x,y} &\neq C^{3}_{x,y} \pmod{8}
\end{align*}
\] (5.8)

However, if needed, the architecture can be modified to remove the constraints either by using dual-port memories, or by introducing an additional pipeline stage.

### 5.2.4 Enforcing check constraints

The principle of enforcing check constraints is described in Sec. 3.1.4, and an approximation in Sec. 3.1.5. The enforcing check constraint approximation has been simulated on a bit-true level, with some additional changes to improve the performance. The modifications, and their associated hardware requirements are explained in detail in this section.

First, with a requirement of \( 3k - 4 \) and gates, the enforcing conditions of the check-to-variable messages are computed explicitly in the CNU from the disabled bits of the incoming variable-to-check messages. Then, using \( k w \)-bit 2-to-1 MUXes,
the magnitude of the check-to-variable message is set to the highest possible value in the representation for enforcing messages. Also, in order to make this value unique, the function value of $\Phi[x]$ for $x = 0$ is reduced by one. As the slope of the continuous $\Phi(x)$ function is steep for $x$ close to zero, this change can be expected to introduce a small amount of errors, as the “true” value of the check-to-variable message is anywhere between $\Phi(2^{-(W_f+1)})$ and infinity. Using this change, enforcing check constraints can be detected by the receiving VNU, as the magnitude of the incoming check-to-variable message is maximized only for check-to-variable messages that are enforcing.

Second, the VNU will use the knowledge of enforcing check constraints both to detect contradicting enforcing check constraints, and to make additional early decisions for variables that receive enforcing check-to-variable messages. The contradiction detection can be straightforwardly implemented for a three-input VNU using $3(w-1) + 6$ and gates and $3$ xor gates, where $w$ is the wordlength of the check-to-variable messages. The logic for the additional early decisions can be implemented using e.g. three one-bit two-to-one MUXes.

Third, the CNU detects the case that all incoming variable-to-check messages are decided, but the parity-check constraint is not satisfied. This can be done using one additional and gate to determine if all incoming messages are decided, and one and gate to determine the inconsistency.

### 5.3 Hybrid decoder

With the early decision decoder as defined in this chapter, the sum-product algorithm can be implemented using the same hardware simply by adjusting the threshold. Thus, the hybrid decoder can be implemented with some controlling logic to clear the extrinsic message memories and change the threshold level when a decoder failure occurs for the early decision decoder.

As seen in Sec. 6, the hybrid decoder offers gains only in the high SNR region, as most blocks will fail to be decoded by both decoders in the low SNR region. Thus, if channel information is available, the SNR estimation can be used to determine if the receiver will use the early decision algorithm or not, and could also be used to set a suitable threshold. However, this idea has not been further investigated in this thesis.
In this chapter, simulation results are provided. The chapter is divided in three main sections: floating point simulations, fixed point simulations and synthesis results. In the first section, the theoretical limitations of the early decision and hybrid algorithm are analyzed for three different codes. In the second section, the performance of the hybrid algorithm under fixed wordlength conditions is simulated. In the last section, slice utilization and power dissipation estimations are provided from synthesis results for a Xilinx Virtex 5 FPGA.

Parts of the results in this section have previously been published in [3–7]. The unpublished results are those of enforcing check constraints and the synthesis results.

6.1 Floating-point simulations

6.1.1 Choice of threshold

Figures 6.1–6.4 show early decision decoding results for a (3,6)-regular rate-1/2 code with a block length of 1152 bits. In all decoding instances, the maximum number of iterations was set to 100, and the early decision algorithm with thresholds of 4, 5 and 6 were simulated. While 100 iterations is generally not practically feasible, the number was chosen to highlight the behaviour of the sum-product algorithm, and
Figure 6.1 Decoding of \((N, j, k) = (1152, 3, 6)\) code, block error rate.

Figure 6.2 Decoding of \((N, j, k) = (1152, 3, 6)\) code, bit error rate.
6.1. FLOWING-POINT SIMULATIONS

Figure 6.3 Decoding of \((N,j,k) = (1152,3,6)\) code, average number of internal messages.

Figure 6.4 Decoding of \((N,j,k) = (1152,3,6)\) code, average number of iterations.
the complications resulting from the early decision modification.

It is obvious that early decision impacts the block error rate, shown in Fig. 6.1, significantly more than the bit error rate, shown in Fig. 6.2. This is to be expected, as the decoding algorithm may correct most of the bits even if some bit is erroneously decided. The behaviour is consistent through all performed simulations, and thus only the block error rate is shown in most cases.

If the LDPC code is used as an outer code in an error correction system, the small amount of bit errors introduced by the early decision decoder can be corrected by the decoder for the inner code. In other cases, hybrid decoding, as explained in Sec. 3.2, may be efficiently utilized.

The average number of internal messages sent per block is shown in Fig. 6.3, and the average number of iterations is shown in Fig. 6.4. It is evident that, whereas the internal communication is reduced with reducing thresholds, the decoding time is increased as the decoder may get stuck when an incorrect decision is made. In the following figures, usually a measure of the relative communication reduction is shown instead of the absolute number of internal messages. The measure is defined as \( \text{comm.red.} = 1 - \frac{C_{\text{mod}}}{C_{\text{ref}}} \), where \( C_{\text{mod}} \) is the number of internal messages using the modified algorithm, and \( C_{\text{ref}} \) is the number of internal messages using the reference algorithm. Results of reducing the number of iterations using enforcing check constraints is showed in Sec. 6.1.2.

The performance of the early decision algorithm for a rate-3/4 \((N, j, k) = (1152, 3, 12)\).
(1152, 3, 12) code is shown in Fig. 6.5, and for a rate-1/2 \((N, j, k) = (9216, 3, 6)\) code in Fig. 6.6. The performance for the higher rate code is significantly better than for the rate-1/2 code of the same block size. This can be intuitively explained by the larger number of variable nodes connected to each check node, thus increasing the amount of information used to compute the bit reliabilities and decreasing the chance of making wrong decisions. Similarly, the better error correction capability of the longer code allows a noisier signal which increases the chance of making wrong decisions.

As described in Sec. 3.1.3, early decision decoding results in an error floor as defined by (3.4). The error floor has been computed for two different thresholds for the \((N, j, k) = (1152, 3, 6)\) code, and is shown for \(t = 3\) in Fig. 6.7 and for \(t = 4\) in Fig. 6.8. As seen, the theoretical limit is tight when decisions are performed only on the initial data (iters = 0). However, if decisions are performed in later iterations, the block error rate increases rapidly. The reasons for this are two. First, the messages no longer have a Gaussian distribution after the first iteration, and the decision error probability is therefore no longer valid. Second, the dependence of the messages caused by cycles in the graph causes the reliabilities of the bits to escalate.
Figure 6.7  Error floor resulting from early decision decoding using $t = 3$ and making decisions only in the initial iterations.

Figure 6.8  Error floor resulting from early decision decoding using $t = 4$ and making decisions only in the initial iterations.
6.1. FLOATING-POINT SIMULATIONS

Figure 6.9 Decoding of \((N, j, k) = (1152, 3, 6)\) code using enforcing checks.
Figure 6.10  Decoding of $\langle N, j, k \rangle = (1152, 3, 12)$ code using enforcing checks.
6.1. FLOATING-POINT SIMULATIONS

6.1.2 Enforcing check constraints

Enforcing check constraints were introduced in Sec. 3.1.4 as a way to reduce the number of iterations required by the early decision algorithm when decoding of a block fails. The results are shown for the \((N, j, k) = (1152, 3, 6)\) code in Fig. 6.9 and for the \((N, j, k) = (1152, 3, 12)\) code in Fig. 6.10. By comparing with Fig. 6.1 and Fig. 6.5 it can be seen that the error correction capability of the decoder is not additionally worsened by the enforcing check constraints modification. However, the average number of iterations is significantly reduced, and for low SNRs the early decision algorithm requires an even lower number of iterations than the standard algorithm as graph inconsistencies are discovered for blocks that the standard algorithm will not be able to successfully decode.

The enforcing check approximation is investigated in Figs. 6.11 and 6.12 for the \((N, j, k) = (1152, 3, 6)\) and \((N, j, k) = (1152, 3, 12)\) codes, respectively. It is apparent that the approximation does not reduce the decoder performance, and thus that additional state information to attribute check-to-variable messages as enforcing is not needed.

6.1.3 Hybrid decoding

In this section, results using the hybrid algorithm explained in Sec. 3.2 is presented. The maximum number of iterations were set to 100 for both the early decision pass
and the standard algorithm. For the early decision algorithm, enforcing check constraints have been used to reduce the number of iterations. Simulations have been done on the three example codes with \((N, j, k) = (1152, 3, 6), (1152, 3, 12),\) and \((9216, 3, 6).\) For each code, first the SNR has been kept constant while the threshold has been varied. It can be seen that setting the threshold too low increases the communication (reduces the communication reduction) as the early decision algorithm will fail on most blocks. Similarly, setting the threshold too high also increases the communication as bits with high reliabilities are not decided. Thus, for a fixed SNR there will be an optimal choice of the threshold with respect to the internal communication. For each code, SNRs corresponding to block error rates of around \(10^{-2}\) and \(10^{-4}\) with the standard algorithm have been used.

Using the optimal threshold for a block error rate of \(10^{-4}\), simulations with varying SNR have been done. It can be seen that the block error rate performance of the hybrid algorithm is indistinguishable from that of the standard algorithm (in the SNR region simulated), whereas the internal communication of the decoder is much lower. In fact, for the rate-1/2 codes, no decoder errors have occurred, and therefore all errors introduced by wrong decisions of the early decision algorithm were detected. However, for the rate-3/4 code, undetected errors did occur, and thus it is possible that the performance of the hybrid algorithm is inferior to that of the standard algorithm.

Simulations for the \((N, j, k) = (1152, 3, 6)\) code are presented in Figs. 6.13
and 6.14, for the \((N,j,k) = (1152,3,12)\) code in Figs. 6.15 and 6.16, and for the \((N,j,k) = (9216,3,6)\) code in Figs. 6.17 and 6.18. In the plots, the average number of iterations denote the sum of the number of iterations required for the early decision pass and the standard algorithm pass.

### 6.2 Fixed-point simulations

The performance of a randomized QC-LDPC code with parameters \((N,j,k) = (1152,3,6)\) has been simulated using a fixed-point implementation. Among an ensemble of 300 codes constructed using random parameters, the code with the best block error correcting performance at an SNR of 2.6 was selected. The random parameters are defined in Sec. 5.1, and were the initial values \(C_{2x,y}^2\) and \(C_{3x,y}^3\) of the address generators \(AG_{2x,y}^2\) and \(AG_{3x,y}^3\), the permutation functions \(\Psi_{l,n}\) and \(\Omega_{l,m}\), and the contents of the row and column permutation ROMs for \(\pi_{3}, \psi_{l,n}\) and \(\omega_{l,m}\). Furthermore, the initial values of the address generators were chosen to satisfy (5.7) to ensure high girth, as well as (5.8) to ensure that an early decision decoder implementation is possible.

In Fig. 6.19, the performance of the randomized QC-LDPC code is compared with the random LDPC code used in the earlier sections. It is seen that the performances are comparable down to a block error rate of \(10^{-6}\). In the same figure, the performances using a fixed-point implementation with wordlengths of 5 and 6 are shown. For both fixed-point simulations, two integer bits were used, as increasing the number of integer bits did not increase the performance. Down to a block error rate of \(10^{-4}\), both fixed-point formats show a performance hit of less than 0.1 dB. The performance difference between wordlengths of 5 and 6 bits is rather small, which is consistent with previous results [43], and thus \(w = 5\) has been used for the following simulations. It is apparent that the fixed-point implementations show an increased error floor, which is likely due to clipping of the message magnitudes.

In Fig. 6.20, the performance of the hybrid early decision-sum product decoder is simulated for SNRs of 2.0 and 2.5, while the threshold is varied. Similarly to the floating point case, for a fixed SNR there is an optimal choice of the threshold that decreases the internal communications of the decoder. The limited increase of the number of iterations for low thresholds is due to the use of enforced check constraints. The fixed-point implementation uses a different scaling of the input due to the limited range of the data representation, and thus the thresholds used in the fixed-point simulations are not directly comparable to the ones used in the floating-point simulations.

Keeping the threshold constant and varying the SNR yields the results in Fig. 6.21, where it can be seen that the use of early decision gives a somewhat lower block error probability than using only the sum-product algorithm. This can be explained by the fact that the increased probabilities of the early decided bits increases the likelihood of the surrounding bits to take consistent values. Thereby in some cases, the early decision decoder converges to the correct codeword, where
Chapter 6. Results

Figure 6.13 Decoding of \((N, j, k) = (1152, 3, 6)\) code using hybrid algorithm showing performance as function of threshold.
6.2. FIXED-POINT SIMULATIONS

![Graph showing Block error rate and Communication reduction](image)

(a) Block error rate, communication reduction.

![Graph showing Average number of iterations](image)

(b) Average number of iterations.

**Figure 6.14** Decoding of \((N, j, k) = (1152, 3, 6)\) code using hybrid algorithm showing performance as function of SNR for threshold 4 (optimal) and 5.
Figure 6.15 Decoding of \((N,j,k) = (1152,3,12)\) code using hybrid algorithm showing performance as function of threshold.
6.2. FIXED-POINT SIMULATIONS

![Graph](image)

(a) Block error rate, communication reduction.

![Graph](image)

(b) Average number of iterations.

**Figure 6.16** Decoding of $(N,j,k) = (1152,3,12)$ code using hybrid algorithm showing performance as function of SNR for threshold $3.75$ (optimal) and $4.5$. 
CHAPTER 6. RESULTS

Figure 6.17 Decoding of \((N, j, k) = (9216, 3, 6)\) code using hybrid algorithm showing performance as function of threshold.
6.2. FIXED-POINT SIMULATIONS

(a) Block error rate, communication reduction.

(b) Average number of iterations.

Figure 6.18 Decoding of \((N, j, k) = (9216, 3, 6)\) code using hybrid algorithm showing performance as function of SNR for threshold 5 (optimal) and 7.
CHAPTER 6. RESULTS

Figure 6.19 Decoding of $(N, j, k) = (1152, 3, 6)$ randomized QC-LDPC code, using floating-point and fixed-point precision. $(w, w_f)$ denotes quantization to $w$ bits where $w_f$ bits are fraction bits.
Figure 6.20 Decoding of \((N, j, k) = (1152, 3, 6)\) randomized QC-LDPC code using \((w, w_f) = (5, 2)\) precision and early decision.
Figure 6.21 Decoding of \((N, j, k) = (1152, 3, 6)\) randomized QC-LDPC code using \((w, w_f) = (5, 2)\) precision and early decision.
6.3. SYNTHESIS RESULTS

The reference sum-product decoder and the early decision decoder as described in Chapter 5 have been implemented in a Xilinx Virtex 5 FPGA for the randomized QC-LDPC code in the previous section. A threshold of $t = 8$ was used at an SNR of 3 dB. The utilization of the FPGA is shown in Table 6.1, along with energy estimations obtained with Xilinx’ power analyzer XPower. As seen, the early-decision algorithm reduces the logic energy dissipation drastically. Unfortunately, the control overhead is relatively large, resulting in a slice utilization overhead of 45% and a net energy reduction of only 16% when increased clock distribution energy is considered. However, some points are worth mentioning:

- In order not to restrain the architecture to short codes, the hard decision and early decision memories and accompanying logic in each memory block were designed in a general and scalable way. However, with the code size used, this resulted in two four-byte memories with eight eight-bit shift registers used to schedule the memory accesses (c.f. Fig. 5.12). As the controlling logic is independent of the code size, it can be expected that the early decision overhead would be reduced with larger codes. Alternatively, for short codes, implementing the hard decision and early decision memories using individual registers might also significantly reduce the overhead.

- The addresses to the hard decision and early decision memories were generated individually from the address generators, along with control signals to the shift registers. However, a more efficient implementation utilizing sharing between different memory blocks is expected to be possible.

<table>
<thead>
<tr>
<th></th>
<th>Reference decoder</th>
<th>Early-decision decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice utilization [%]</td>
<td>14.5</td>
<td>21.0</td>
</tr>
<tr>
<td>Number of block RAMs</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>Maximum frequency [MHz]</td>
<td>302</td>
<td>220</td>
</tr>
<tr>
<td>Clock energy [pJ/iteration/bit]</td>
<td>141</td>
<td>190</td>
</tr>
</tbody>
</table>

Table 6.1 Synthesis results for the reference sum-product decoder and the early-decision decoder in a Xilinx Virtex 5 FPGA.

the sum-product decoder would fail. Regarding the reduction of internal communication, results close to those obtained using floating-point data representation are achieved.

6.3 Synthesis results
Conclusions and future work

In this chapter, the work presented in this thesis is concluded, and suggestions for future work are given.

7.1 Conclusion

In this thesis, the early decision modification to the sum-product decoding algorithm for LDPC codes was suggested. Furthermore, the early decision modification was combined with the sum-product algorithm to form a hybrid decoding algorithm which does not visibly reduce the performance of the decoder. With the modified algorithm, the internal communication of the decoder can be reduced significantly. For a regular \((N, j, k) = (1152, 3, 6)\) code, an internal communication reduction of 40\% was achieved at a block error rate of \(10^{-6}\). Increasing the rate decreases the communication further, whereas increased block size increases the communication.

The hybrid algorithm was synthesized to a Xilinx Virtex 5 FPGA, and the logic utilization was analyzed and the power dissipation estimated. However, due to increased logic to handle the early decisions, the actual achievable energy reduction was limited to 16\% by the increased clock distribution network.

A minor contribution of the thesis was the identification of a redundancy in the internal data representation commonly used in LDPC decoders. Exploiting the redundancy allows a reduction of the data wordlength of the internal messages,
with a minimal logic overhead.

7.2 Future work

Some concrete ideas for future work are:

- investigating ways of removing early decisions when graph inconsistencies are encountered. This is likely necessary for the early decision algorithm to work well with codes longer than 1000-2000 bits.

- analyzing the performance of early decision decoding over fading channels. As one of the main suggestions for early decision decoding is wireless mobile devices where minimizing the power dissipation is required, a decent performance over fading channels is important.

- to determine how to adequately choose the threshold under changing channel conditions. Assuming that the SNR is known at the receiver, the information can be used to select a threshold giving optimal performance.

- investigating the performance of early decision decoding and the choice of thresholds when irregular codes are used and the probability density functions of messages from nodes with different node degrees may differ.
References


[48] “IEEE 802.11 wireless LANs, WWiSE proposal: High throughput extension to the 802.11,” IEEE Std. 11-04-0866-00-000n.


