Linköping Studies in Science and Technology. Dissertations, No. 1385

Low Complexity Techniques for Low Density Parity Check Code Decoders and Parallel Sigma-Delta ADC Structures

Anton Blad

Linköping University
INSTITUTE OF TECHNOLOGY

Department of Electrical Engineering
Linköping University
SE–581 83 Linköping
Sweden

Linköping 2011

Low Complexity Techniques for Low Density Parity Check Code Decoders and Parallel Sigma-Delta ADC Structures

Anton Blad

Linköping Studies in Science and Technology. Dissertations, No. 1385

ISBN 978-91-7393-104-5
ISSN 0345-7524

e-mail: anton.blad@gmail.com
thesis url: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-69432

Department of Electrical Engineering
Linköping University
SE–581 83 Linköping
Sweden

Printed by LiU-Tryck, Linköping, Sweden 2011

Abstract

In this thesis, contributions are made in the area of receivers for wireless communication standards. The thesis consists of two parts, focusing on implementations of forward error correction using low-density parity-check (LDPC) codes, and high-bandwidth analog-to-digital converters (ADC) using sigma-delta modulators.

LDPC codes have received wide-spread attention since 1995 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 current and future communication standards, including the 802.16e WiMAX standard, the 802.11n WLAN standard, and the second generation of digital TV standards DVB-x2. The first part of the thesis contains two main contributions to the problem of decoding LDPC codes, denoted the early-decision decoding algorithm and the check-merging decoding algorithm. The early-decision decoding algorithm is a method of terminating parts of the decoding process early for bits that have high reliabilities, thereby reducing the computational complexity of the decoder. The check-merging decoding algorithm is a method of reducing the code complexity of rate-compatible LDPC codes and increasing the efficiency of the decoding algorithm, thereby offering a significant throughput increase. For the two algorithms, architectures are proposed and synthesized for FPGAs and the resulting performance and logic utilization are compared with the original algorithms.

Sigma-delta ADCs are the natural choice of low-to-medium bandwidth applications that require high resolution. However, suggestions have also been made to use them for high-bandwidth communication standards which require either high sampling rates or several ADCs operating in parallel. In this thesis, two contributions are made in the area of high-bandwidth ADCs using sigma-delta modulators. The first is a general formulation of parallel ADCs using modulation of the input data. The formulation allows a system's sensitivity to analog mismatch errors in the channels to be analyzed and it is shown that some systems can be made insensitive to certain matching errors, whereas others may require matching of limited subsets of the channels, or full matching of all channels. Limited sensitivity to mismatch errors reduces the complexity of the analog parts. Simulation results are provided for a time-interleaved ADC, a Hadamard-modulated ADC, a frequency-band decomposed ADC, as well as for a new modulation scheme that is insensitive to channel gain mismatches. The second contribution relates to the implementation of high-speed digital filters, where a typical application is decimation filters for a high-bandwidth sigma-delta ADC. A bit-level optimization algorithm is proposed that minimizes a cost function defined as a weighted sum of the number of full adders, half adders and registers. Simulation results show comparisons between bit-level optimized filters and structures obtained using common heuristics for carry-save adder trees.

Populärvetenskaplig sammanfattning

De flesta av dagens trådlösa kommunikationssystem bygger på digital överföring av data. Denna avhandling innehåller två delar, vilka ger bidrag till två olika delar i moderna mottagare för digitala kommunikationssystem.

I den första delen behandlas energieffektiv avkodning av felrättande koder. Felrättande kodning är en av de främsta fördelarna med digital kommunikation gentemot analog kommunikation, och bygger på att man i överföringen lägger till redundans till informationen man skickar. Under överföringen blir det ovillkorligt fel, men med hjälp av redundansen är det möjligt att räkna ut var det gick fel, och man kan på så sätt öka pålitligheten i överföringen. Felrättande koder med stor förmåga att upptäcka och rätta fel är lätta att konstruera, men att bygga praktiska implementeringar av en avkodare är svårare, och avkodaren använder ofta en stor del av strömförbrukningen i IC-kretsar för mottagare. I den här avhandlingen behandlas en specifik typ av felrättande koder (LDPC-koder) som kommer mycket nära den teoretiska gränsen för felrättande förmåga, och två förbättringar av avkodningsalgoritmer föreslås. I båda fallen är de föreslagna algoritmerna mer komplexa, men minskar effektförbrukningen i en implementering.

I den andra delen av avhandlingen behandlas en specifik typ av analog-till-digital-omvandlare (ADC), som omvandlar den mottagna signalen till digital information. Sigma-delta är en typ av ADC som lämpar sig särskilt väl för integration med digitala system på en gemensam IC-krets. Nackdelen idag är dock att konverteringshastigheten är relativt låg. Ett sätt att öka hastigheten är att använda flera omvandlare parallellt, som då tar hand om delar av insignalen var för sig. Nackdelen är att sådana system ofta blir känsliga för variationer mellan de olika omvandlarna, och i den här avhandlingen föreslås en metod att modellera flera parallella sigma-delta-ADC:er för att analysera känslighetskraven. Det visar sig att vissa system är känsliga för variationer, medan andra kan kräva anpassning av begränsade delmängder av omvandlarna. Ett annat problem är att utdata från sigma-delta-omvandlare består av en dataström med mycket hög datatakt och mycket kvantiseringsbrus. För att kunna använda dataströmmen i en applikation måste den först decimeras. Avhandlingen innehåller också en metod att formulera skapandet sådana decimeringsfilter som ett optimeringsproblem, för att på så vis skapa filter med låg komplexitet.

Acknowledgments

It is with mixed feelings I look back at the 5+ years as a PhD student at Electronics Systems. This thesis is the result of many hours of work in a competent and motivating environment, an environment promoting independence and individualism while still offering abundant support and opportunities for discussion. Being a PhD student has been hard at times, but looking back I cannot imagine conditions better suited for a researcher at the start of his career.

There are many people who have given me inspiration and support through these years, and I want to take the opportunity here to say thanks to the following people:

- My supervisor Dr. Oscar Gustafsson for being a huge source of motivation by always taking his time to discuss my work and endlessly offering new insights and ideas.
- My very good friends Dr. Fredrik Kuivinen and M.Sc. Jakob Rosén for all the fun with electronics projects and retro gaming sessions in the evenings at campus.
- Prof. Christer Svensson for getting me in contact with the STMicroelectronics research lab in Geneva.
- Dr. Andras Pozsgay at the Advanced Radio Architectures group at STMicroelectronics in Geneva, for offering me the possibility of working with "practical" research for six months during spring 2007. It has been a very valuable experience.
- All the other people in my research group at STMicroelectronics in Geneva, for making my stay there a pleasant experience.
- Prof. Fei Zesong at the Department of Electrical Engineering at Beijing Institute of Technology in China, for giving me the possibility of two three-month PhD student exchanges in spring 2009 and winter 2010/2011.
- M.Sc. Wang Hao and M.Sc. Shen Zhuzhe for helping me with all the practicalities for my visits in Beijing.
- M.Sc. Zhao Hongjie for the cooperation during his year as a PhD student exchange at Linköping University.
- All the others at the Modern Communication Lab at Beijing Institute of Technology for their kindness and support in an environment that was very different to what I am used to.
- Dr. Kent Palmkvist for help with FPGA- and VHDL-related issues.
- M.Sc. Sune Söderkvist for the big contribution to the generally happy and positive atmosphere at Electronics Systems.

- All the other present and former colleagues at Electronics Systems.
- All the colleagues at the Electronic Components research group during my time there from 2006 to 2008.
- All the colleagues at Communications Systems during my time there as a research engineer in spring 2010.
- All my friends who have made my life pleasant during my time as a PhD student.
- Last but not least, I thank my parents Maj and Bengt Blad for all their encouragement and time that they gave me as a child, which is definitely part of the reason that I have come this far in life. I also thank my sisters Lisa and Tove Blad, who I don't see very often but still feel I am very close to when I do.

Anton Blad

Linköping, July 2011

Abbreviations

ADC	Analog-to-Digital Converter
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BLER	Block Error Rate
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CFU	Check Function Unit
CIC	Cascaded Integrator Comb
CMOS	Complementary Metal Oxide Semiconductor
CNU	Check Node processing Unit
CSA	Carry-Save Adder
CSD	Canonic Signed-Digit
DAC	Digital-to-Analog Converter
DECT	Digital Enhanced Cordless Telecommunications
DFT	Discrete Fourier Transform
DTTB	Digital Terrestrial Television Broadcasting
DVB-S2	Digital Video Broadcasting - Satellite 2nd generation

E_b/N_0	Bit energy to noise spectral density (normalized SNR)
ECC	Error Correction Coding
ED	Early Decision
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GPS	Global Positioning Satellite
ILP	Integer Linear Programming
k-SE	k-step enabled
k-SR	k-step recoverable
LAN	Local Area Network
LDPC	Low-Density Parity Check
LUT	Look-Up Table
MPR	McClellan-Parks-Rabiner
MSD	Minimum Signed-Digit
MUX	Multiplexer
OSR	Oversampling ratio
PSD	Power Spectral Density
QAM	Quadrature Amplitude Modulation
QC-LDPC	Quasi-Cyclic Low-Density Parity-Check
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
ROM	Read-Only Memory
SD	Signed-Digit
SNR	Signal-to-Noise Ratio
USB	Universal Serial Bus
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VLSI	Very Large Scale Integration
VMA	Vector Merge Adder
VNU	Variable Node processing Unit
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

Preface

Thesis outline

In this thesis, contributions are made in two different areas related to the design of receivers for radio communications, and the contents are therefore separated into two parts. Part I consists of Chapters 1–6 and offers contributions in the area of low density parity check (LDPC) code decoding, whereas Part II consists of Chapters 7–13 and offers contributions related to high-speed analog-to-digital conversion using $\Sigma\Delta$-ADCs.

The outline of Part I is as follows. In Chapter 1, a short background, possible applications and the scientific contributions are discussed. In Chapter 2, the basics of digital communications are described and LDPC codes are introduced. Also, two decoder architectures are described, which are used as reference implementations for the contributed work. In Chapter 3, early decision decoding is proposed as a method of reducing the computational complexity of the decoding algorithm. Performance issues related to the algorithm are analyzed, and solutions are suggested. Also, an implementation of the algorithm for FPGA is described, and the resulting estimations of area and power dissipation are included. In Chapter 4, an improved algorithm for decoding of rate-compatible LDPC codes are proposed. The algorithm offers a significant reduction of the average number of iterations required for decoding of punctured codes, thereby offering a significant increase in

throughput. An architecture implementing the algorithm is proposed, and simulation and synthesis results are included. In Chapter 5, a minor contribution in the data representation of a sum-product LDPC decoder is explained. It is shown how redundancy in the data representation can be used to reduce the required memory used for storage of messages between iterations. Finally, in Chapter 6, conclusions are given and future work is discussed.

The outline of Part II is as follows. In Chapter 7, an introduction to high-speed data conversion is given, and the scientific contributions of the second part of the thesis are described. In Chapter 8, a short introduction to finite impulse response (FIR) filters, multirate theory and FIR filter architectures is given. In Chapter 9, the basics of ADCs using $\Sigma\Delta$-modulators are discussed, and some high-speed structures using parallel $\Sigma\Delta$-ADCs are shown. In Chapter 10, a general model for the analysis of matching requirements in parallel $\Sigma\Delta$-ADCs is proposed. It is shown that some parallel systems may become alias-free with limited matching between subsets of the channels, whereas others may require matching between all channels. In Chapter 11, a short analysis of the relations between oversampling factors, $\Sigma\Delta$-modulator orders, required signal-to-noise ratio (SNR) and decimation filter complexity is contributed. In Chapter 12, an integer linear programming approach to the design of high-speed decimation filters for $\Sigma\Delta$-ADCs is proposed. Several architectures are discussed and their complexities compared. Finally, in Chapter 13, conclusions are given and future work is discussed.

Publications

This thesis contains research done at Electronics Systems, department of Electrical Engineering at Linköping University, Sweden. The work has been done between March 2005 and June 2011, and has resulted in the following publications [7–17]:

1. A. Blad, O. Gustafsson, and L. Wanhammar, "An LDPC decoding algorithm utilizing early decisions," in *Proc. National Conf. Radio Science*, Jun. 2005.

2. A. Blad, O. Gustafsson, and L. Wanhammar, "An early decision decoding algorithm for LDPC codes using dynamic thresholds," in *Proc. European Conf. Circuit Theory Design*, Aug. 2005, pp. 285–288.

3. A. Blad, O. Gustafsson, and L. Wanhammar, "A hybrid early decision-probability propagation decoding algorithm for low-density parity-check codes," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Oct. 2005.

4. A. Blad, O. Gustafsson, and L. Wanhammar, "Implementation aspects of an early decision decoder for LDPC codes," in *Proc. Nordic Event ASIC Design*, Nov. 2005.

5. A. Blad and O. Gustafsson, "Energy-efficient data representation in LDPC decoders," *IET Electron. Lett.*, vol. 42, no. 18, pp. 1051–1052, Aug. 2006.

6. A. Blad, P. Löwenborg, and H. Johansson, "Design trade-offs for linear-phase FIR decimation filters and sigma-delta modulators," in *Proc. XIV European Signal Process. Conf.*, Sep. 2006.

7. A. Blad, H. Johansson, and P. Löwenborg, "Multirate formulation for mismatch sensitivity analysis of analog-to-digital converters that utilize parallel sigma-delta modulators," *Eurasip J. Advances Signal Process.*, vol. 2008, 2008, article ID 289184, 11 pages.

8. A. Blad and O. Gustafsson, "Integer linear programming-based bit-level optimization for high-speed FIR decimation filter architectures," *Springer Circuits, Syst. Signal Process. - Special Issue on Low Power Digital Filter Design Techniques and Their Applications*, vol. 29, no. 1, pp. 81–101, Feb. 2010.

9. A. Blad and O. Gustafsson, "Redundancy reduction for high-speed FIR filter architectures based on carry-save adder trees," in *Proc. Int. Symp. Circuits, Syst.*, May 2010.

10. A. Blad, O. Gustafsson, M. Zheng, and Z. Fei, "Integer linear programming based optimization of puncturing sequences for quasi-cyclic low-density parity-check codes," in *Proc. Int. Symp. Turbo-Codes, Related Topics*, Sep. 2010.

11. A. Blad and O. Gustafsson, "FPGA implementation of rate-compatible QC-LDPC code decoder," in *Proc. European Conf. Circuit Theory Design*, Aug. 2011.

During the period, the following papers were also published, but are either outside the scope of this thesis or overlapping with the publications above:

1. A. Blad, C. Svensson, H. Johansson, and S. Andersson, "An RF sampling radio frontend based on sigma-delta conversion," in *Proc. Nordic Event ASIC Design*, Nov. 2006.

2. A. Blad, H. Johansson, and P. Löwenborg, "A general formulation of analog-to-digital converters using parallel sigma-delta modulators and modulation sequences," in *Proc. Asia-Pacific Conf. Circuits Syst.*, Dec. 2006, pp. 438–441.

3. A. Blad and O. Gustafsson, "Bit-level optimized high-speed architectures for decimation filter applications," in *Proc. Int. Symp. Circuits, Syst.*, May 2008.

4. M. Zheng, Z. Fei, X. Chen, J. Kuang, and A. Blad, "Power efficient partial repeated cooperation scheme with regular LDPC code," in *Proc. Vehicular Tech. Conf.*, May 2010.

5. O. Gustafsson, K. Amiri, D. Andersson, A. Blad, C. Bonnet, J. R. Cavallaro, J. Declerckz, A. Dejonghe, P. Eliardsson, M. Glasse, A. Hayar, L. Hollevoet,

C. Hunter, M. Joshi, F. Kaltenberger, R. Knopp, K. Le, Z. Miljanic, P. Murphy, F. Naessens, N. Nikaein, D. Nussbaum, R. Pacalet, P. Raghavan, A. Sabharwal, O. Sarode, P. Spasojevic, Y. Sun, H. M. Tullberg, T. Vander Aa, L. Van der Perre, M. Wetterwald and M. Wu, "Architecture for cognitive radio testbeds and demonstrators - An overview," in *Proc. Int. Conf. Cognitive Radio Oriented Wireless Networks Comm.*, Jun. 2010.

6. A. Blad, O. Gustafsson, M. Zheng, and Z. Fei, "Rate-compatible LDPC code decoder using check-node merging," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Nov. 2010.

7. M. Abbas, O. Gustafsson, and A. Blad, "Low-complexity parallel evaluation of powers exploiting bit-level redundancy," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Nov. 2010.

Contents

Part I

Decoding of low-density parity-check codes

1

Introduction

1.1 Background

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

ECC is used in almost all digital communications systems to improve link performance and reduce transmitter power requirements [3]. 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 communications systems, where transmission noise will irrevocably 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 communications, with bit error rate (BER) or block error rate (BLER) for

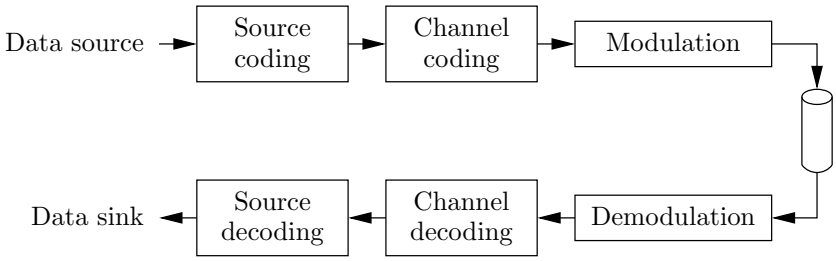


Figure 1.1 *Simple communications system model*

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

As the quality metrics of digital and analog communications systems are different, the performance of an analog and a digital system cannot easily 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 four standard definition channels of comparable quality in the same bandwidth as one channel in the analog TV.

A simple model of a digital communications system is shown 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 differs 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 part of the thesis considers the hardware implementation of the channel decoder for low-density parity-check (LDPC) codes. The decoding of LDPC codes is complex, and is often a major part of the baseband processing of a receiver. For example, the flexible decoder in [79] supports the LDPC codes in IEEE 802.11n and IEEE 802.16e, as well as the Turbo codes in 3GPP-LTE, but at a maximum power dissipation of 675 mW. 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 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-correction performance.

LDPC codes were discovered originally in 1962 by Robert Gallager [39]. 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 [74,101], and have been shown to have a very good performance close to the theoretical Shannon limit [73, 75]. 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 communications 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 [113]. The same type of code was then adopted for both the DVB-T2 [114] and DVB-C2 [115] standards for terrestial and cable-based TV, respectively. A similar type has also been accepted for the DTTB standard for digital TV in China [122]. The system-level requirements of these systems are relatively low, with relaxed 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 [118] and WiMax IEEE 802.16e [120] 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 part of the thesis, LDPC codes are also used or suggested in several other standards and applications. Among them are the IEEE 802.3an [121] standard for 10Gbit/s Ethernet, the IEEE 802.15.3c [119] mm-wave WPAN standard, and the gsfc-std-9100 [116] standard for deep-space communications.

1.3 Scientific contributions

There are two main scientific contribution in the first part of the thesis. The first is a modification to the sum-product decoding algorithm for LDPC codes, called the early-decision algorithm, and is described in Chapter 3. 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 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.

The second main contribution is an improved algorithm for decoding of rate-compatible LDPC codes, and is described in Chapter 4. Using rate-compatible LDPC codes obtained through puncturing, the higher-rate codes can trivially be decoded by the low-rate mother code. However, by defining a specific code by merging relevant check nodes for each of the punctured rates, the code complexity can be reduced at the same time as the propagation speed of the extrinsic information is increased. The result is a significant reduction in the convergence time of the decoding algorithm for the higher-rate codes.

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

The performance of the proposed algorithms have been evaluated in software. For the early-decision algorithm, it is verified that the modifications have an insignificant impact on the error correction performance, and the change in the internal communication is estimated. For the check-merging decoding algorithm, the modifications can be shown to even improve the error correction performance. However, these improvements are mostly due to the reduced convergence time, allowing the algorithm to converge for codewords which the original algorithm does not have sufficient time for.

The early-decision and check-merging algorithms have been implemented in a Xilinx Virtex 5 FPGA and an Altera Cyclone II FPGA, respectively. As similar implementations have not been published before, they have mainly been compared with implementations of the original reference decoders. For the early-decision decoder, the required overhead has been determined, and the power dissipation of both the original and proposed architecture have been simulated and compared using regular quasi-cyclic LDPC codes with an additional scrambling layer. For the check-merging decoder, the required overhead has been determined, and the increased throughput obtainable with the modification has been quantized for two different implementations geared for the IEEE 802.16e and IEEE 802.11n standards, respectively.

2

Error correction coding

In this chapter, the basics of digital communications systems and error correction coding are explained. In Sec. 2.1, a model of a system using digital communications is shown, and the channel model and different modulation methods are explained. In Sec. 2.2, coding theory is introduced as a way of reducing the required transmission power with a retained bit error probability, and block codes are defined. In Sec. 2.3, 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 an LDPC code. The sum-product decoding algorithm and the min-sum approximation are discussed in Sec. 2.4, and in Sec. 2.5, rate-compatible LDPC codes are introduced as a way of obtaining practical usable codes with a wide range of rates. Also, the implications of rate-compatibility on the decoding algorithm are discussed. In Sec. 2.6, several general decoder architectures with different parallelism degrees are discussed, including a serial architecture, a parallel architecture, and a partly parallel architecture. In Sec. 2.7, a partly parallel architecture for a specific class of regular LDPC codes is described, and is also used as a reference for the early decision algorithm proposed in Chapter 3. In Sec. 2.8, a partly parallel architecture using the min-sum decoding algorithm for general quasi-cyclic LDPC codes is described, and is used as a reference for the check merging decoding algorithm proposed in Chapter 4.

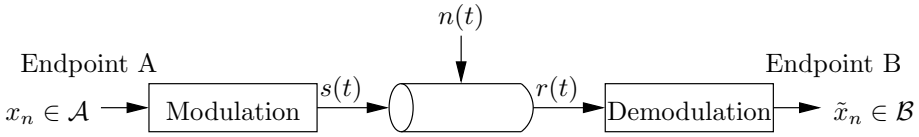


Figure 2.1 *Digital communications system model.*

2.1 Digital communications

Consider a two-user digital communications system, such as the one shown in Fig. 2.1, where an endpoint A transmits information to an endpoint B. Whereas multi-user communications 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 communications 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

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

or analogously for continuous reception alphabets

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

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 communications system.

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

$$P(\tilde{X} = b \mid X = a) = f_{(a,\sigma)}(b), \tag{2.3}$$

  where $f_{(a,\sigma)}(b)$ is the probability density function for a normally distributed 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 method used, and is further explained in Sec. 2.1.2. The AWGN channel models real-world noise sources well, especially for cable-based communications systems.

- the Rayleigh and Rician fading channels. The Rayleigh channel is appropriate for modeling a wireless communications 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 communications path 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.

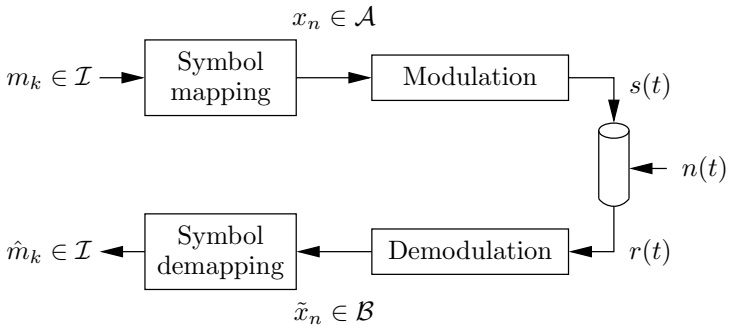


Figure 2.2 *Model of uncoded digital communications system.*

2.1.2 Modulation methods

The size of the transmission alphabet $\mathcal{A}$ for the AWGN channel is commonly determined by the modulation method used. Common modulation methods include

- the binary phase-shift keying (BPSK) modulation, using the transmission alphabet $\mathcal{A} = \{-\sqrt{E}, +\sqrt{E}\}$ and reception alphabet $\mathcal{B} = \mathbb{R}$. E denotes the symbol energy.
- the quadrature phase-shift keying (QPSK) modulation, using the transmission alphabet $\mathcal{A} = \sqrt{E/2}\{(-1-i), (-1+i), (+1-i), (+1+i)\}$ with complex symbols, and reception alphabet $\mathcal{B} = \mathbb{C}$. 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 straight-forwardly be applied to systems using other modulation methods 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 communications 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 communications 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 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,

$$\hat{m}_k = \begin{cases} \tilde{x}_n & \text{if } p < 0.5 \\ 1 - \tilde{x}_n & \text{if } p > 0.5, \end{cases} \tag{2.4}$$

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

$$\hat{m}_k = \begin{cases} 0 & \text{if } \tilde{x}_n > 0 \\ 1 & \text{if } \tilde{x}_n < 0. \end{cases} \tag{2.5}$$

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

$$\{\hat{m}_k, \hat{m}_{k+1}\} = \begin{cases} 00 & \text{if } \operatorname{Re}\tilde{x}_n > 0, \operatorname{Im}\tilde{x}_n > 0 \\ 01 & \text{if } \operatorname{Re}\tilde{x}_n < 0, \operatorname{Im}\tilde{x}_n > 0 \\ 11 & \text{if } \operatorname{Re}\tilde{x}_n < 0, \operatorname{Im}\tilde{x}_n < 0 \\ 10 & \text{if } \operatorname{Re}\tilde{x}_n > 0, \operatorname{Im}\tilde{x}_n < 0. \end{cases} \tag{2.6}$$

In analyzing the performance of a communications system, the probability of erroneous transmissions is interesting. For BPSK communications with equal symbol probabilities, the bit error probability can be defined as

$$\begin{aligned} P_{B,BPSK} &= P(\hat{m}_k \neq m_k) = \\ &= P(\tilde{x}_n > 0 \mid x_n = 1)P(x_n = 1) + P(\tilde{x}_n < 0 \mid x_0 = 0)P(x_n = 0) = \\ &= Q\left(\frac{\sqrt{E}}{\sigma}\right) = Q\left(\sqrt{\frac{2E}{N_0}}\right), \end{aligned} \tag{2.7}$$

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

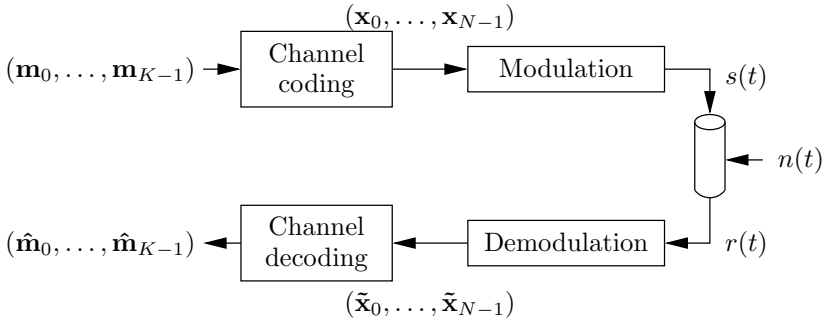


Figure 2.3 *Error correction system overview*

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 $\mathbf{m}$ with K bits is to be communicated over a noisy channel. The message is encoded to the codeword $\mathbf{x}$ 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 $\tilde{\mathbf{x}}$, which may contain either bits or scalars. The channel decoder is then used to find the most likely sent codeword $\hat{\mathbf{x}}$, given the received vector $\tilde{\mathbf{x}}$. From $\hat{\mathbf{x}}$, the message bits $\hat{\mathbf{m}}$ 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

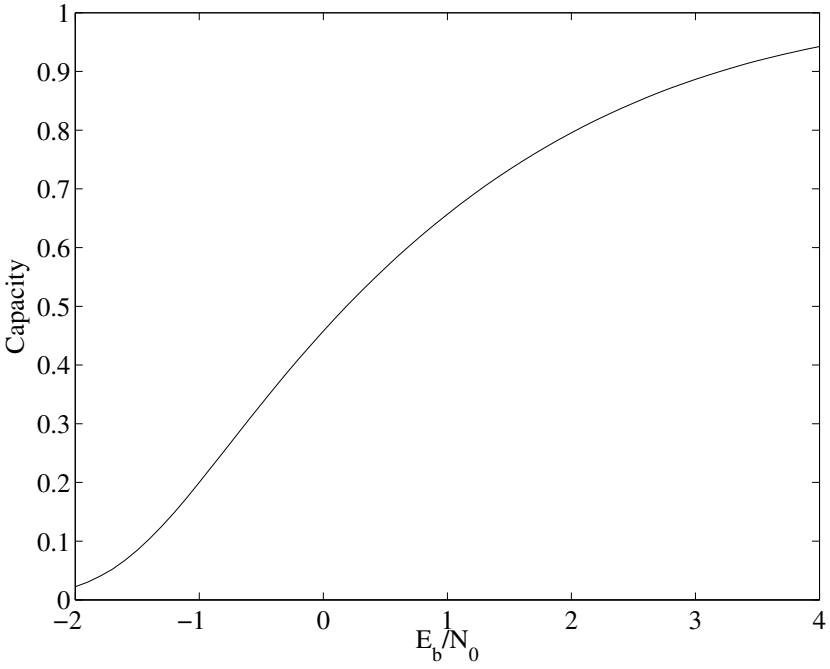


Figure 2.4 *Capacity for binary-input AWGN channel using different SNR.*

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 $\mathbf{m}$ and $\hat{\mathbf{m}}$, averaged over several blocks.

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

Coding systems are analyzed in depth in any introductionary book on coding theory, e.g. [3, 102].

2.2.1 Shannon bound

In 1948, Claude E. Shannon proved the noisy channel coding theorem [89], 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 [3]. It is

$$C_{BIAWGN} = \int_{-\infty}^{\infty} f_{\sqrt{2E/N_0}}(y) \log_2 \left(\frac{2 f_{\sqrt{2E/N_0}}(y)}{f_{\sqrt{2E/N_0}}(y) + f_{-\sqrt{2E/N_0}}(y)} \right) \mathrm{d}y, \qquad (2.8)$$

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 communications 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 codewords by C, C can be defined in the

following two ways:

$$C = \left\{ \mathbf{x} = \mathbf{mG} \,|\, \mathbf{m} \in \{0,1\}^K \right\} \tag{2.9}$$

$$C = \left\{ \mathbf{x} \in \{0,1\}^N \,|\, \mathbf{Hx}^T = \mathbf{0} \right\} \tag{2.10}$$

The most important property of a code regarding performance is the minimum Hamming distance d, which is the minimum number of bits that two codewords may differ in. Moreover, as the set of codewords C is linear, it is also the weight of the lowest-weight codeword which is not the all-zero codeword. 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.

A simple example of a block code is the $(N, K, d) = (7, 4, 3)$ Hamming code defined by the parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{2.11}$$

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 $\tilde{\mathbf{x}}$. Then the syndrome of the received vector is $\mathbf{H}\tilde{\mathbf{x}}^T$, 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 $\mathbf{H}$ that matched the syndrome is flipped. It should thus be noted that the columns of $\mathbf{H}$ contain every non-zero three-bit vector, and thus every received vector $\tilde{\mathbf{x}}$ 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

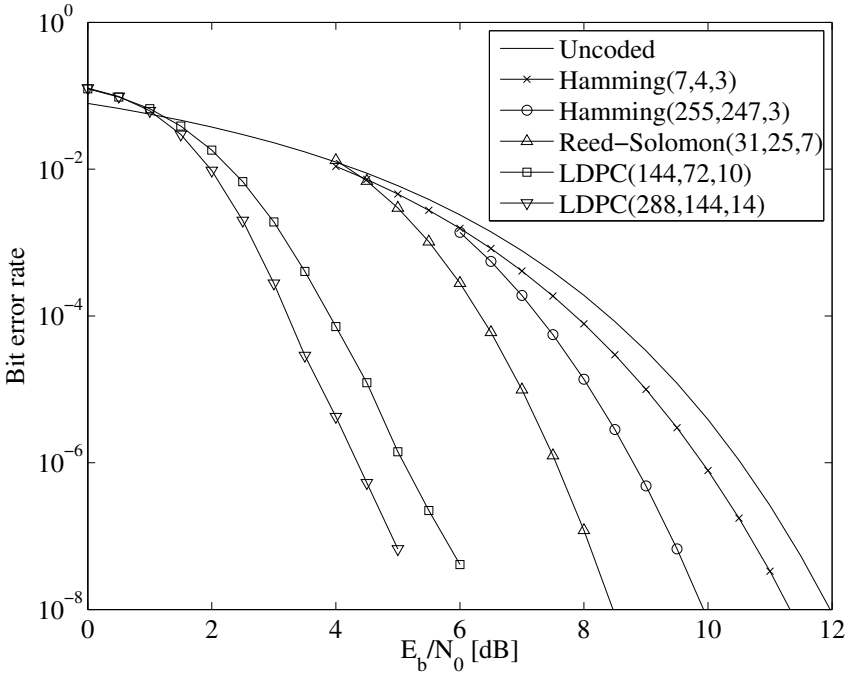


Figure 2.5 *Error correcting performance of short codes. The Hamming and Reed-Solomon curves are estimations for hard-decision decoding, whereas the LDPC curves are obtained using simulations with soft-decision decoding.*

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™ function `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 [26], whereas the performance of the $N = 10^6$ codes are from [86]. It is seen that at a block length of 10^6 bits, the LDPC code performs better than the Turbo code. 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

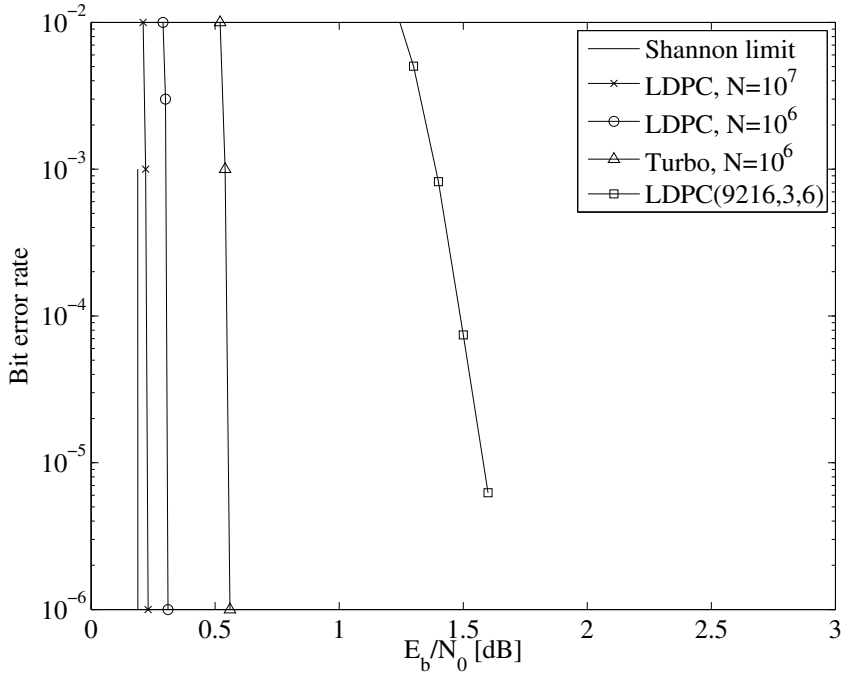


Figure 2.6 *Error correcting performance of long codes.*

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 Sec. 3.5.

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 $\hat{\mathbf{m}}$ equals $\mathbf{m}$.

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

- **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 $\mathbf{m}$. 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.

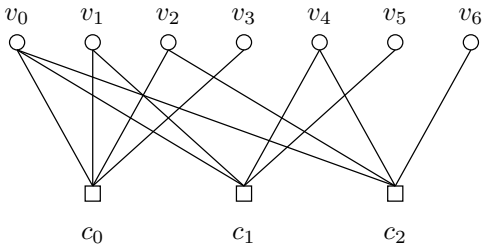


Figure 2.7 *Example of Tanner graph for the* $(7, 4, 3)$ *Hamming code.*

		Variable nodes						
		v_0	v_1	v_2	v_3	v_4	v_5	v_6
	c_0	1	1	1	1	0	0	0
Check nodes	c_1	1	1	0	0	1	1	0
	c_2	1	0	1	0	1	0	1

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

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 $\mathbf{H}$ has a low number of 1s. It has been shown [39] 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 $\mathbf{H}$ 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, leading to a trade-off between the performance of the code and the complexity of the encoder and decoder.

2.3.1 Tanner graphs

LDPC codes are commonly visualized using Tanner graphs [92]. Moreover, the iterative decoding algorithms are defined directly on the graph (see Sec. 2.4.1). 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.

Having defined the Tanner graph, there are some properties which are interesting for the decoding algorithms for LDPC codes.

- 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 [39] 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 [25, 27, 72, 87] 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 computes exact marginal bit probabilities when the code's Tanner graph is free of cycles [65]. However, it can also be shown that the graph must contain cycles for the code to have more than minimal error correcting performance [36]. Specifically, it is shown that for a cycle-free code $\mathcal{C}$ 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 $\mathcal{C}$ 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.

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. These types of LDPC codes are denoted quasi-cyclic (QC-LDPC) codes. Typically, QC-LDPC codes are defined from a base

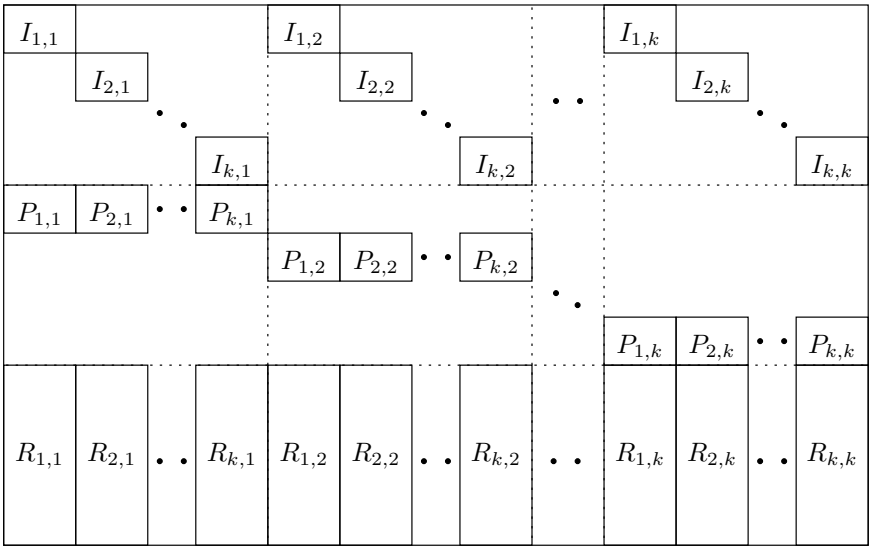


Figure 2.9 *Parity-check matrix structure of randomized quasi-cyclic codes through joint code and decoder architecture design.*

matrix $\mathbf{H}_b$ of size $M_b \times N_b$ with integer elements:

$$\mathbf{H}_b = \begin{pmatrix} \mathbf{H}_b(0,0) & \mathbf{H}_b(0,1) & \cdots & \mathbf{H}_b(0,N_b-1) \\ \mathbf{H}_b(1,0) & \mathbf{H}_b(1,1) & \cdots & \mathbf{H}_b(1,N_b-1) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{H}_b(M_b-1,0) & \mathbf{H}_b(M_b-1,1) & \cdots & \mathbf{H}_b(M_b-1,N_b-1) \end{pmatrix}. \tag{2.12}$$

For an expansion factor of z, a parity-check matrix $\mathbf{H}$ of size $M_b z \times N_b z$ is constructed from $\mathbf{H}_b$ by replacing each element with a square sub-matrix of size $z \times z$. The sub-matrix is the all zero matrix if $\mathbf{H}_b(m,n) = -1$, otherwise it is an identity matrix circularly right-shifted by $\phi(\mathbf{H}_b(m,n), z)$. $\phi(k,z)$ is commonly a scaling function, modulo function, or the identity function.

Methods of constructing QC-LDPC codes include algebraic methods [55, 77, 95], geometric methods [63, 71, 95], and random or optimization approaches [37, 98]. QC-LDPC codes tend to have decent performance while also allowing the implementation to be efficiently parallelized. The block size may easily be adapted by changing the exansion factor z. Also, certain construction methods can ensure that the girth of the code is at least 8 [96]. In all of the standards using LDPC codes that are referenced in this thesis, the codes are of QC-LDPC structure.

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 [107, 110]. 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. 2.7.

2.4 LDPC decoding algorithms

Normally, LDPC codes are decoded using a belief propagation algorithm. In this section, the sum-product algorithm and the common min-sum approximation are explained.

2.4.1 Sum-product algorithm

The sum-product decoding algorithm is defined directly on the Tanner graph of the code [39, 65, 74, 101]. It is an iterative algorithm, consecutively propagating bit probabilities and parity-check constraint satisfiability likelihoods until the algorithm converges to a valid codeword, 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.

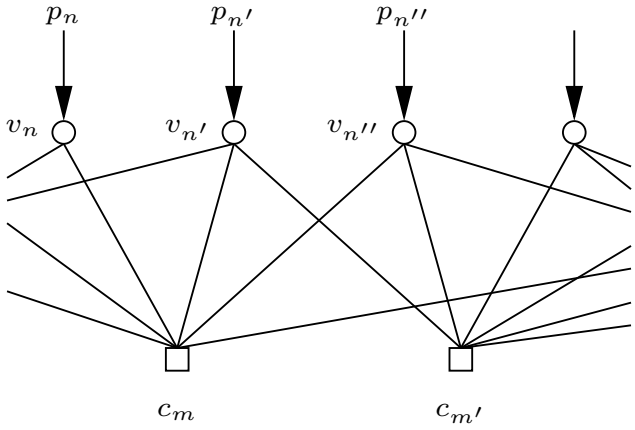


Figure 2.10 *Sum-product decoding: Initialization phase*

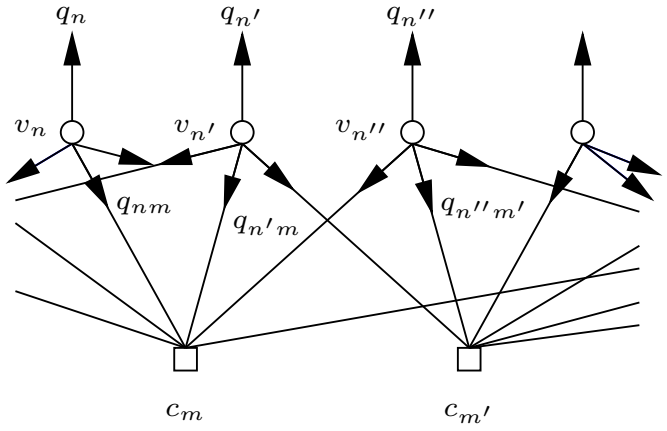


Figure 2.11 *Sum-product decoding: Variable node update phase*

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

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

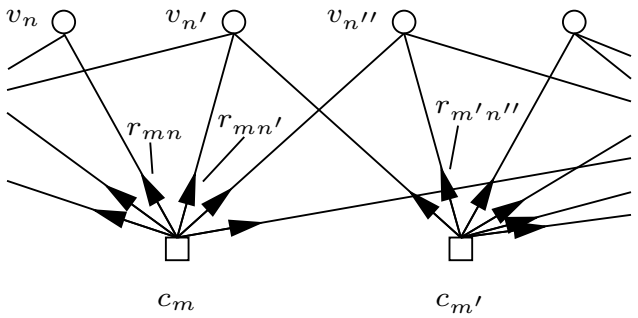


Figure 2.12 *Sum-product decoding: Check node update phase*

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 codeword 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 $\gamma_n = \log(p_n^0/p_n^1)$, $\alpha_{nm} = \log(q_{nm}^0/q_{nm}^1)$, $\beta_{mn} = \log(r_{mn}^0/r_{mn}^1)$ and $\lambda_n = \log(q_n^0/q_n^1)$. In this domain, the variable update equations can be written [65]

$$\alpha_{nm} = \gamma_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \beta_{mn} \tag{2.13}$$

$$\beta_{mn} = \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sign} \alpha_{nm} \right) \cdot \Phi \left(\sum_{n' \in \mathcal{N}(m) \setminus n} \Phi \left(|\alpha_{nm}| \right) \right) \tag{2.14}$$

$$\lambda_n = \gamma_n + \sum_{m' \in \mathcal{M}(n)} \beta_{mn}, \tag{2.15}$$

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

The sum-product algorithm is used in the implementation of the early-decision algorithm in Chapter 3.

2.4.2 Min-sum approximation

Whereas (2.13) and (2.15) consist of sums and are simple to implement in hardware, (2.14) is a bit more complex. One way of simplifying the hardware implementation is the use of the min-sum approximation [38] which replaces the check node operation by the minimum of the arguments. The min-sum approximation results in an overestimation of the reliabilities of messages, as only the probability for one message is used in the operation. This can be partly compensated for by adding an offset to variable-to-check messages [23], and results in the following equations:

$$\alpha_{nm} = \gamma_n + \sum_{m' \in \mathcal{M}(n) \backslash m} \beta_{mn} \tag{2.16}$$

$$\beta_{mn} = \left(\prod_{n' \in \mathcal{N}(m) \backslash n} \operatorname{sign} \alpha_{nm} \right) \cdot \max \left(\min_{n' \in \mathcal{N}(b) \backslash n} |\alpha_{nm} - \delta| , 0 \right) \tag{2.17}$$

$$\lambda_n = \gamma_n + \sum_{m' \in \mathcal{M}(n)} \beta_{mn}, \tag{2.18}$$

where δ is a constant determined by simulations. An additional result of the approximation is that the number of different messages from a specific check node is reduced to at most two. This enables a significant reduction in the memory requirements for storage of the check-to-variable messages in some architectures, especially when a layered decoding schedule is used.

The offset min-sum algorithm is used in the implementation of the rate-compatible decoder in Chapter 4.

2.5 Rate-compatible LDPC codes

A class of rate-compatible codes is defined as a set of codes with the same number of codewords but different rates, and where codewords of higher rates can be obtained from codewords of lower rates by removing bits at fixed positions [48]. Thus the information content is the same in the codes, but the amount of parity information differs. The benefits of rate-compatibility include better adaptation to channel environments and more efficient implementations of encoders and decoders. Better adaptation to channel environments is achieved through the large number of possible rates to choose from, whereas more efficient implementations are achieved through the reuse of hardware between the encoders and decoders of the different rates. An additional advantage is the possibility to use smart ARQ schemes where the retransmission consists of a small amount of extra parity bits rather than a completely recoded packet.

There are two main methods of defining such classes of codes: puncturing and extension. Using puncturing, a low-rate mother code is designed and the higher-rate codes are then defined by removing bits at fixed positions in the blocks. Using extension, lower-rate codes are defined from a high-rate mother code by adding additional parity bits.

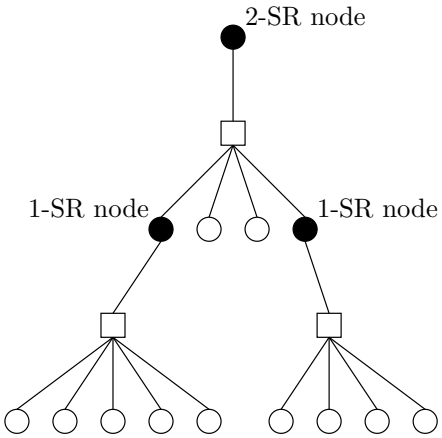


Figure 2.13 *Recovery tree of a 2-SR node. The circles are variable nodes and the squares are check nodes. The filled circles are punctured nodes.*

Disadvantages of rate-compatible codes include reduced performance of the code and decoder. Since it is difficult to optimize a class of rate-compatible codes for a range of different rates, there will generally be a performance difference between a rate-compatible and a dedicated code of a specific rate. However, better adaptation to channel conditions may still allow a decrease in the average number of transmitted bits.

2.5.1 SR-nodes

For LDPC codes, a straight-forward way of decoding rate-compatible codes obtained through puncturing is to use the parity-check matrix of the low-rate mother code and initialize the prior LLRs of the punctured nodes to zero. However, such nodes will delay the probability propagation of its check node neighbors until it receives a non-zero message from one of its neighbors. The concept of k-step recoverable (k-SR) nodes was introduced in [47] based on the assumption that the performance of an LDPC code using a particular puncturing pattern is mainly determined by the recovery time of the punctured nodes. The recovery time of a punctured variable node is defined as the minimum number of iterations required before a node can start to produce non-zero messages. A non-punctured node can thus be denoted a 0-SR node. A punctured node having at least one check node neighbor for which it is the only punctured node may receive a non-zero message from that node in the first iteration and is thus a 1-SR node. Generally, a k-SR node is reinitialized by its neighbors after k iterations. Figure 2.13 shows the recovery tree of a 2-SR punctured node. The 2-SR node has no other check node neighbors for which it is the only punctured node.

In [47], a greedy algorithm was proposed that successively chooses variable nodes that receive likelihood data from their neighbours as early as possible. Since then, several other greedy algorithms have been proposed. In [24], an algorithm that takes into account the reliability of the information used for reinitializing punctured nodes was suggested. However, it can only be used for parity-check matrices with dual-diagonal structures. In [46] and [83], algorithms maximizing the reinitialization reliability for general parity-check matrix structures were proposed. The algorithm in [46] is essentially a refinement of the one proposed in [47], sequentially puncturing k-SR nodes for increasing values of k while trying to choose nodes with high recovering reliability. In [83], the aims are similar, but instead of minimizing the number of iterations for recovering, the algorithm tries to choose nodes based on the number of unpunctured nodes used in the computation of the recovery information.

2.5.2 Decoding of rate-compatible codes

Rate-compatible LDPC codes can be straight-forwardly decoded by any LDPC decoder simply by initializing the a-priori LLR values of punctured nodes to zero. In that respect, any LDPC decoder is also a rate-compatible decoder. However, by reducing the number of different codes that must be supported by the decoder, the usage of rate-compatible codes may allow simplifications to the architecture. In [106], a rate-compatible decoder is presented for the WiMAX rate-1/2 code, where hard-wiring the information exchange between the nodes allows a low-complexity implementation.

In contrast, in Chapter 4 an alternative decoding algorithm is proposed, which removes the punctured nodes altogether from the code. The result is a significant increase of the propagation speed of the messages in the decoder, which together with a reduced code complexity allows a significant throughput increase.

2.6 LDPC decoder architectures

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 [99,104]. For example, a direct instantiation of the Tanner graph of a 1024-bit code in a 0.16 μm CMOS process resulted in a chip with more than 26000 wires with an average length of 3 mm, and a routing overhead of 50% [18,52]. 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 [44]: the parallel, the serial, and the partly parallel architecture. These are briefly described here.

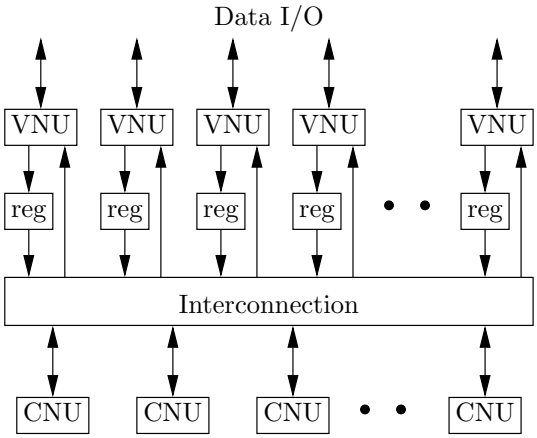


Figure 2.14 *Parallel architecture for sum-product decoding algorithm*

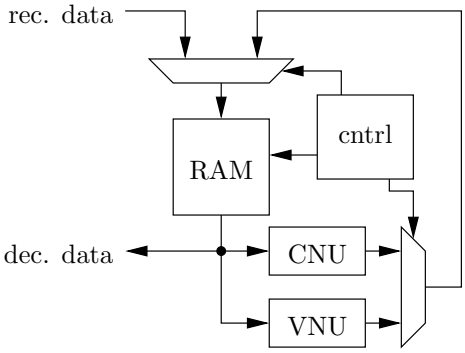


Figure 2.15 *Serial architecture for sum-product decoding algorithm.*

2.6.1 Parallel architecture

Directly instantiating the Tanner graph of the code yields the parallel architecture, as shown in Fig. 2.14. 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 [18]. 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

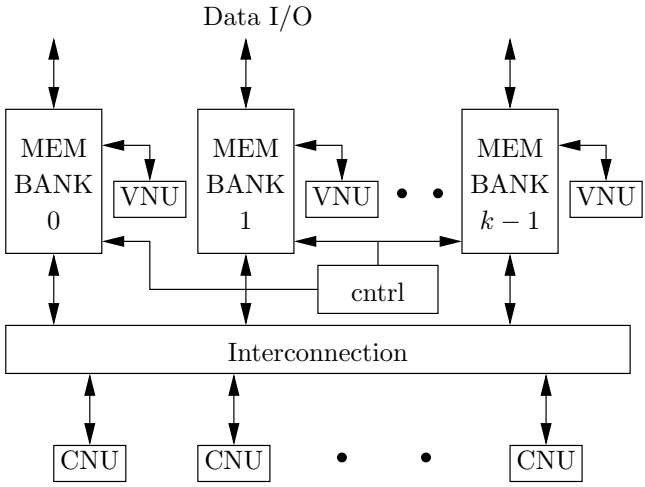


Figure 2.16 *Partly parallel architecture for sum-product decoding algorithm*

computations efficiently.

2.6.2 Serial architecture

Another obvious architecture is the serial architecture, shown in Fig. 2.15. 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 [105], 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.

2.6.3 Partly parallel architecture

A third possible architecture is the serial/parallel, or partly parallel architecture, shown in Fig. 2.16, which can be seen as either a time-multiplexed parallel decoder, or an interleaved serial decoder. The partly 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 partly 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 partly parallel architecture.

In the partly parallel architecture, parallelism can be achieved in a number of different ways, with different advantages and drawbacks. Considering a QC-LDPC code, the main parallelism choices are either inside or across the sub-matrices.

Examples of the partly parallel architecture include a 3.33 Gb/s $(1200, 720)$ code decoder with a power dissipation of 644 mW, manufactured in a 0.18μm technology [70], and a 250 Mb/s $(1944, 972)$ code decoder dissipating 76 mW, manufactured in a 0.13μm technology [90]. The implementations in this thesis are of the partly parallel type architecture. The implementation in Chapter 3 is based on an FPGA implementation achieving a throughput of 54 Mbps for a $(9216, 4608)$ code using a clock frequency of 56 MHz and performing 18 iterations [108]. For the implementation in Chapter 4, two degrees of parallelism are investigated: 24 and 81, respectively. The examined codes are those of the IEEE 802.16e WiMAX standard and the IEEE 802.11n WLAN standard. The achieved throughputs are rate-dependent, but throughputs starting from 33 Mbps and 100 Mbps are achieved in an FPGA implementation with a clock frequency of 100 MHz and performing 10 iterations. It is based on [66].

2.6.4 Finite wordlength considerations

Earlier investigations have shown that the sum-product algorithm for LDPC decoding has relatively low requirements on data wordlength [111]. 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 (2.13)–(2.15), the variable node computations, (2.13) and (2.15), are naturally done using two's complement arithmetic, whereas the check node computations (2.14) are more efficiently carried out using signed-magnitude arithmetic. This is the solution chosen for the implementation in Sec. 3.3, 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. 2.17, there is a big difference between positive and negative zero in the signed-magnitude representation, as the sum in (2.14) 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

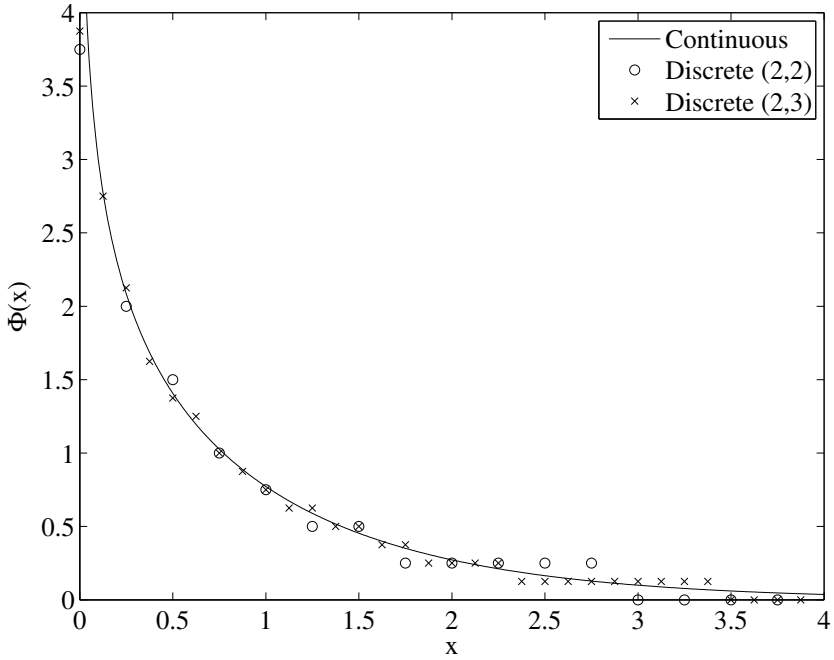


Figure 2.17 *Figure showing $\Phi(x)$ and discrete functions obtained through rounding. For the discrete functions, the notation is (w_i, w_f), where w_i is the number of integer bits and w_f is the number of fractional bits.*

obtained through rounding of the function values are shown for some different data representations in Fig. 2.17. Whereas other quantization rules such as truncation, 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. 5.2 a compression scheme is introduced.

2.6.5 Scaling of $\Phi(x)$

Considering the $\Phi(x)$ function in (2.14), using the natural base for the logarithm is not necessary. However, when the natural base is used, the inverse function $\Phi^{-1}(x) = 2\text{arctanh}\exp(x)$ is identical to $\Phi(x)$, and thus in (2.14) 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

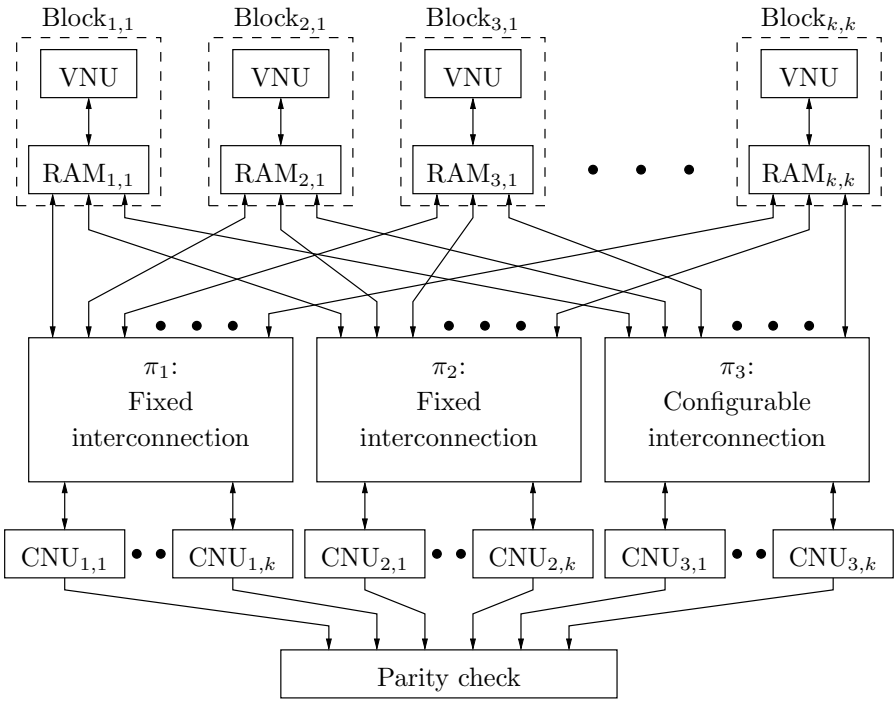


Figure 2.18 *Partially parallel $(3, k)$-regular sum-product decoder overview.*

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 [31], it is shown that such a scaling can improve the performance of the sum-product decoder using fixed-point data. In Sec. 5.2, it is shown how the benefits of internal data coding depend on the choice of logarithm base.

2.7 Sum-product reference decoder architecture

In this section, the sum-product reference decoder for the work in Chapter 3 is presented. The section includes an architecture overview, description of memory blocks and implementations of the variable node processing unit (VNU) and check node processing unit (CNU), as well as the interconnection networks.

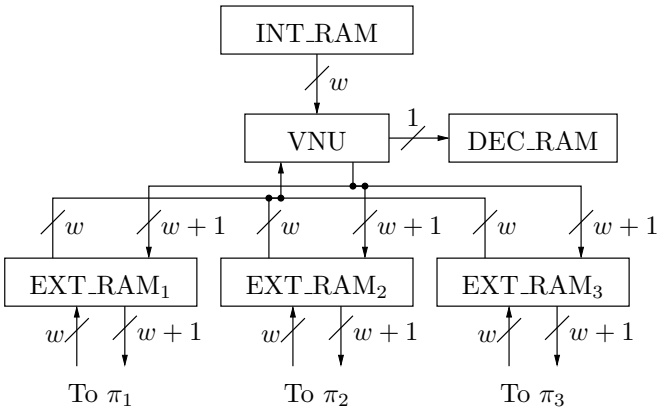


Figure 2.19 *Memory block of $(3, k)$-regular sum-product decoder, containing memories used during decoding, and the VNUs.*

2.7.1 Architecture overview

An overview of the architecture is shown in Fig. 2.18. The architecture contains k^2 memory blocks with the memory banks used during decoding and the VNUs. The memory blocks are connected to $3k$ CNUs through two regular and fixed interconnection networks (π_1 and π_2), and one randomized and configurable interconnection network (π_3). The purpose of the VNUs is to perform the computations of the variable-to-check node messages α_{nm}, as in (2.13), and the pseudo-posterior probabilities λ_n, as in (2.15). Similarly, the purpose of the CNUs is to perform the computation of the check-to-variable messages β_{mn}, as in (2.14), and to compute the check parities using the hard-decision variables $\hat{\mathbf{x}}_n$ obtained from the pseudo-posterior probabilities. However, in the implementation, the computation of $\Phi(\alpha_{nm})$ is moved from the CNUs to the VNUs, and thus the implementation deviates slightly from the formal formulation of the algorithm.

2.7.2 Memory block

The structure of the memory blocks is shown in Fig. 2.19, where w denotes the wordlength of the prior probabilities γ_n. One memory block contains one INT_RAM of width w for storing γ_n, one DEC_RAM of width 1 to store the hard-decision values $\hat{\mathbf{x}}_n$, and three EXT_RAM$_i$ of width $w + 1$ to store the extrinsic information α_{nm} and β_{mn} during decoding. Thus, there are a total of $5k^2$ memories. All memories are of size L. The EXT_RAM$_i$ 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 representation in INT_RAM and in signed magnitude representation

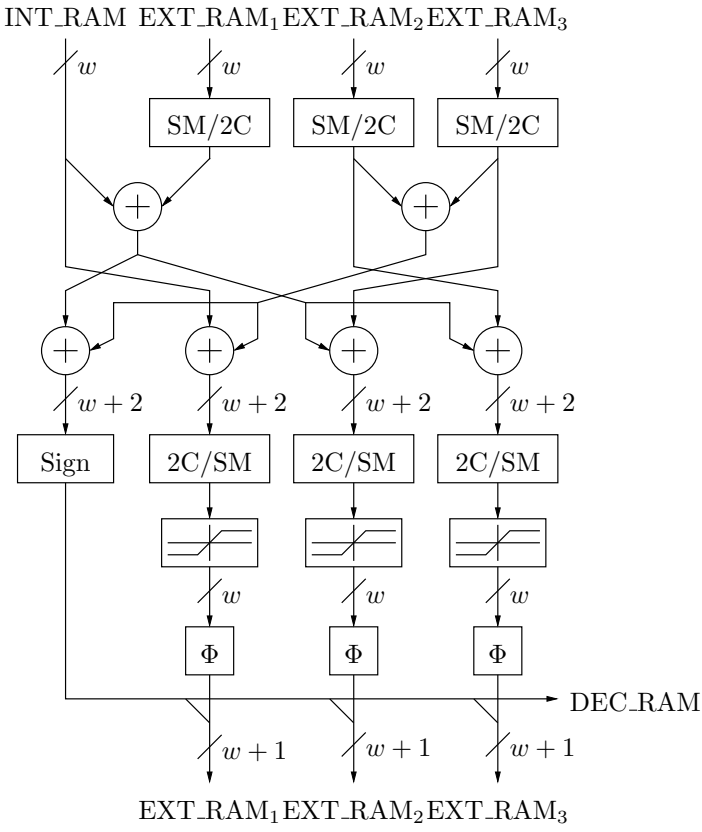


Figure 2.20 *Implementation of VNU for the* $(3, k)$*-regular sum-product decoder.*

in EXT_RAM$_i$. At the output of the EXT_RAM$_i$, there are two registers. Thus switching of signals in the CNU can be disabled in the VNU phase and vice versa.

2.7.3 Variable node processing unit

The implementation of the VNU is relatively straight-forward, as shown in Fig. 2.20. 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. 2.17) of the magnitude is computed, and the results are joined with the hard-decision bit to form the $w + 1$-bit hybrid data.

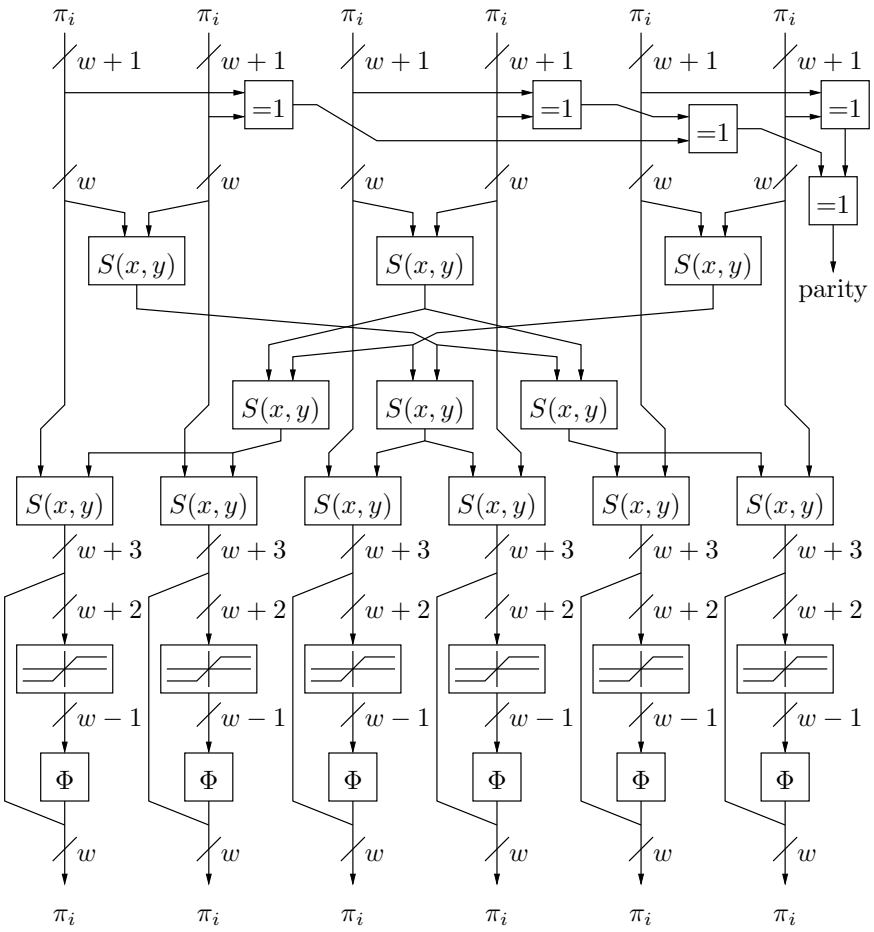


Figure 2.21 *Implementation of CNU for the* $(3,k)$*-regular sum-product decoder with* $k=6$.

2.7.4 Check node processing unit

The implementation of the CNU for $k=6$ is shown in Fig. 2.21. 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

$$S(x,y) = \operatorname{sign}(x) \cdot \operatorname{sign}(y) \cdot (|x| + |y|)\,. \tag{2.19}$$

Finally, the results are truncated to w bits, and the $\Phi(x)$ function of the magnitude is computed.

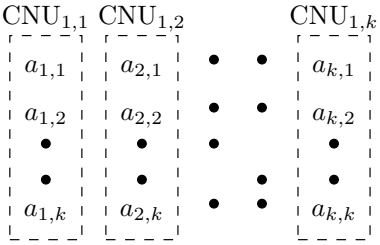


Figure 2.22 *Function of π_1 interconnection network, with $a_{m,n}$ denoting the message from/to memory block (m, n).*

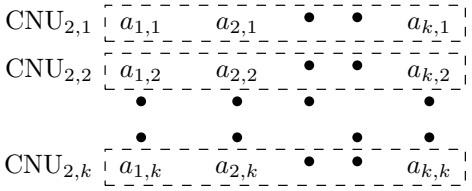


Figure 2.23 *Function of π_2 interconnection network, with $a_{m,n}$ denoting the message from/to memory block (m, n).*

2.7.5 Interconnection networks

The functions of the fixed interconnection networks are shown in Figs. 2.22 and 2.23. π_1 connects the messages from $\text{Block}_{x,y}$ with the same x coordinate to the same CNU, whereas π_2 connects the messages with the same y coordinate to the same CNU.

The function of the configurable interconnection network π_3 is shown in Fig. 2.24 for the forward path. $a_{m,n}$ denotes the message from $\text{Block}_{m,n}$. $a_{m,n}$ are permuted to $b_{m,n}$ by the permutation functions $\Psi_{l,n}$, where l is the value of AG^1, defined in Sec. 2.7.6. $\Psi_{l,n}$ is either the identity permutation or the fixed permutation R_n depending on l and n. Thus, formally

$$b_{m,n} = \begin{cases} a_{m,n} & \text{if } \psi_{l,n} = 0 \\ a_{R_n(m),n} & \text{if } \psi_{l,n} = 1 \end{cases}, \tag{2.20}$$

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 be formalized

$$c_{m,n} = \begin{cases} b_{m,n} & \text{if } \omega_{l,m} = 0 \\ b_{m,C_m(n)} & \text{if } \omega_{l,m} = 1 \end{cases}, \tag{2.21}$$

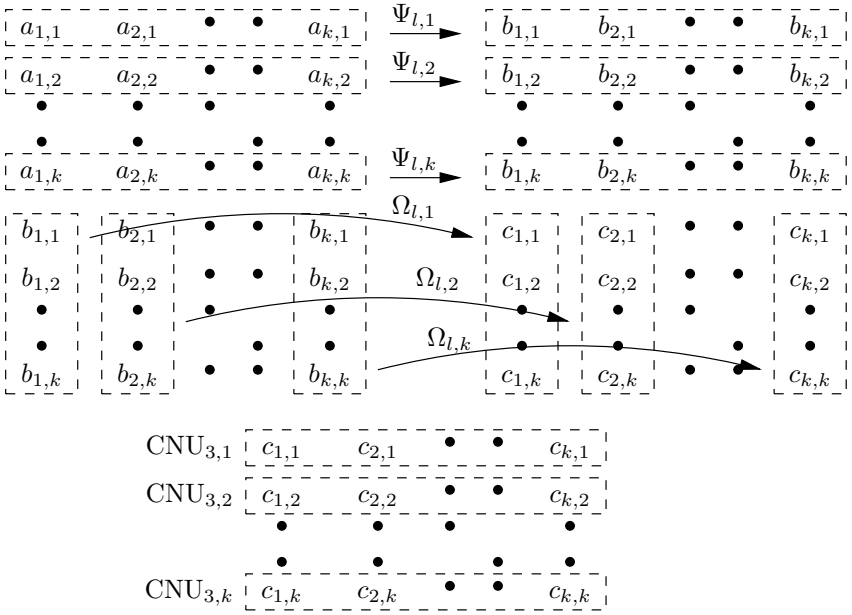


Figure 2.24 *Forward path of π_3 interconnection network, with $a_{m,n}$ denoting the message from memory block (m, n). $\Psi_{l,n}$ and $\Omega_{l,m}$ denote permutation functions that are either the identity permutation or fixed permutations R_n and C_m, respectively.*

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

2.7.6 Memory address generation

The addresses to the EXT_RAM_i memories are given by mod-L binary counters $\text{AG}^i_{x,y}, i = 1, 2, 3$, 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

$$C^1_{x,y} = 0 \tag{2.22}$$

$$C^2_{x,y} = ((x-1)\cdot y) \bmod L \tag{2.23}$$

$$C^3_{x,y_1} \neq C^3_{x,y_2}, \forall y_1, y_2 \in \{1, \ldots, k\}, y_1 \neq y_2 \tag{2.24}$$

$$C^3_{x_1,y} - C^3_{x_2,y} \neq ((x-1)\cdot y) \bmod L, \forall x_1, x_2 \in \{1, \ldots, k\}, x_1 \neq x_2. \tag{2.25}$$

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 [108].

x	$\Phi[x]$	x	$\Phi[x]$	x	$\Phi[x]$	x	$\Phi[x]$
0	15	4	3	8	1	12	0
1	8	5	2	9	1	13	0
2	6	6	2	10	1	14	0
3	4	7	1	11	1	15	0

Table 2.1 *Definition of $\Phi[x]$ function for $(w_i, w_f) = (3, 2)$ fixed point data representation.*

The architecture as described results in the code shown in Fig. 2.9, 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 π_3. The extrinsic data corresponding to $I_{x,y}$ is stored in EXT_RAM_1 in $\text{Block}_{x,y}$, the data corresponding to $P_{x,y}$ in EXT_RAM_2 in $\text{Block}_{x,y}$, and the data corresponding to $R_{x,y}$ in EXT_RAM_3 in $\text{Block}_{x,y}$.

2.7.7 Φ function

The definition of the $\Phi[x]$ function used in the sum-product reference decoder architecture is shown in Table 2.1. The same function is used for the early decision decoder without enforcing check constraints in Sec. 3.3.

2.8 Check-serial min-sum decoder architecture

In this section, the reference decoder [66] for the work in Chapter 4 is presented. The decoder handles all the LDPC codes of the WiMAX standard. It uses the min-sum algorithm ((2.16)–(2.18)) in a layered schedule, and uses serial check node operations. Using a layered schedule, bit-sums are updated directly after the completion of each layer, effectively replacing the variable-node update equation (2.16) with updates directly at the end of each check node computation. The result is a significant increase in the propagation rate of the messages which reduces the convergence time of the algorithm [50].

This section includes a description of the schedule, an architecture overview, and details on the check node function unit (CFU).

2.8.1 Decoder schedule

The schedule of the decoder and relevant definitions are shown in Fig. 2.25:

- A **layer** is a group of check nodes where all the involved variable nodes are independent. It may be bigger than the expansion factor of the base parity-check matrix.

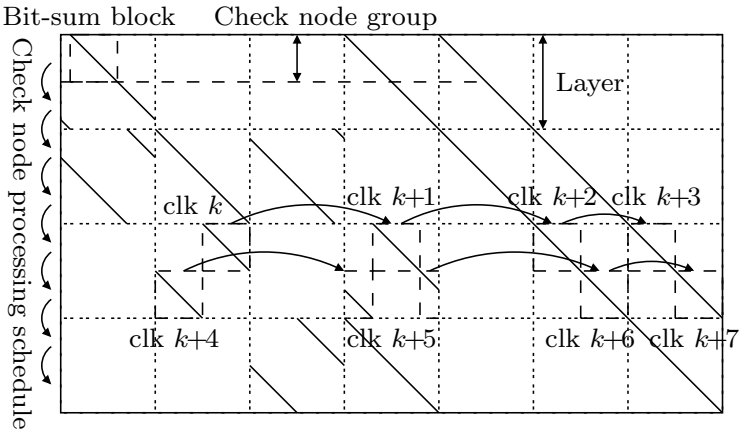


Figure 2.25 *Schedule of check-serial min-sum decoder.*

- A **check node group** is a number of check nodes that are processed in parallel. The variable nodes in the check node groups are processed serially, with the bit-sums for each variable node updated after each operation.
- A **bit-sum block** is the bit-sums of the variable nodes involved in the check-node groups. The bit-sums in a bit-sum block are accessed in parallel and are stored in different memories. One bit-sum block is read per clock cycle, and then re-written one per clock cycle after the processing delay of the routing networks and CFUs.

2.8.2 Architecture overview

An overview of the architecture is shown in Fig. 2.26. The data-path consists of a number of bit-sum memories, a cyclic shifter network, CFUs and a decision memory. The bit-sum memories store the pseudo-posterior bit probabilities as computed by (2.18). In each clock cycle, a bit-sum block is read from the bit-sum memories, routed to the correct CFU by the cyclic shifter, and the CFU computes updated bit-sums that are rewritten to the bit-sum memories. Also, hard decisions are done on the updated bit-sums and written to the decision memory. The CFUs also compute the parities of the signs of the input bit-sums, and decoding stops when all parities are satisfied in an iteration, or when a pre-defined maximum number of iterations is reached.

The following parameters are defined for the architecture:

- Q is the parallelization factor and the size of the bit-sum blocks
- w_q is the bit-sum wordlength
- w_r is the wordlength of the min-sum magnitudes

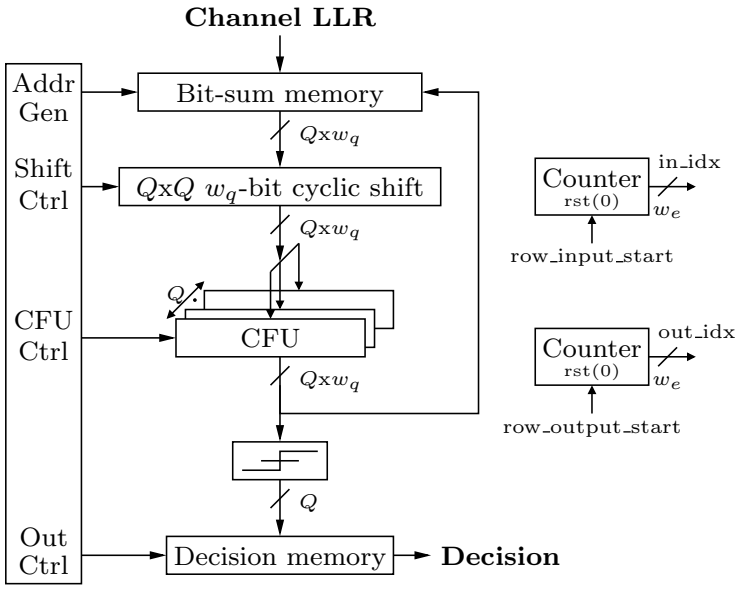


Figure 2.26 *Overview of check-serial min-sum decoder architecture.*

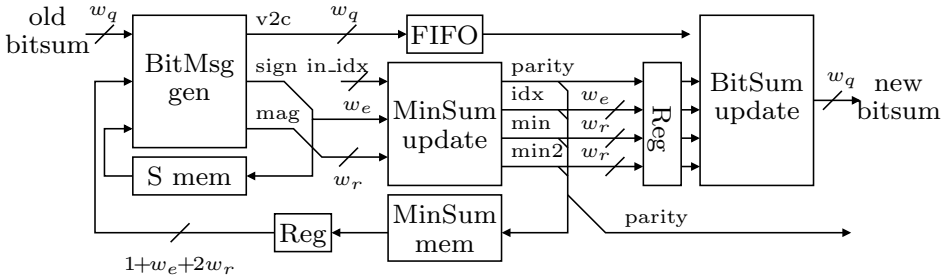


Figure 2.27 *CFU of check-serial min-sum decoder.*

- w_e is the edge index wordlength

2.8.3 Check node function unit

The implementation of the CFU is shown in Fig. 2.27. It contains a min-sum memory and a sign memory storing the check-to-variable message magnitudes and signs, respectively. The entries in the min-sum memory stores the check node parity, the magnitudes of the two smallest variable-to-check messages and the index of the variable node with the smallest variable-to-check message. The wordlength of the min-sum memory is thus $1 + 2w_r + w_e$.

At the input of the CFU, the bit message generator subtracts the old check-

to-variable message from the input bit-sum, forming the variable-to-check message according to (2.16). This is followed by the min-sum update that does a sequential search for the two smallest magnitudes of the variable-to-check messages, and the index of the smallest one. The result is a packed description of all the check-to-variable messages according to (2.17) and is stored in the min-sum memory. It is also used by the bit-sum update unit to compute the new bit-sum (2.18) from the old variable-to-check message and the new check-to-variable message.

For details of the CFU implementation, the reader is referred to [66].

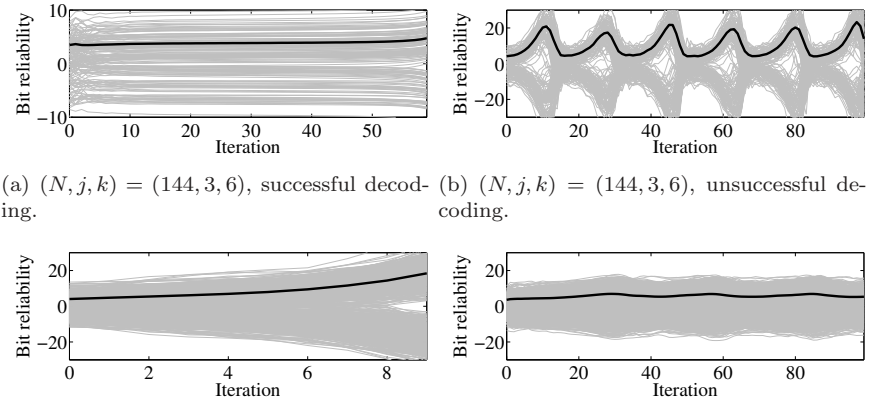
3

Early-decision decoding

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 in Sec. 3.1: 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. In Sec. 3.2, a hybrid algorithm is proposed, removing the performance penalty caused by the early-decision decoding algorithm.

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 Sec. 2.7 (also in [108]) 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 modifications done to implement the early decision algorithm is described in Sec. 3.3, and in Sec. 3.4, it is shown how the early decision decoder can implement the hybrid decoding algorithm.

In Sec. 3.5, floating-point and fixed-point simulation results are provided. The floating-point simulations show the performance of the early decision and hybrid algorithms for three different codes, whereas the performance of the hybrid algorithm under fixed wordlength conditions is shown by the fixed-point simulations.



(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.*

In Sec. 3.6, slice utilization and power dissipation estimations are provided from synthesis results for a Xilinx Virtex 5 FPGA.

Secs. 3.1.1, 3.1.2, 3.1.3 and 3.2 have been published in [14], [11] and [12], whereas Secs. 3.1.4 and 3.1.5 are previously unpublished. The implementations of algorithms, Sec. 3.3 and Sec. 3.4, have been published in [13].

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 $\min\left(q_n^0, q_n^1\right)$. Assuming that this value is small, which is the case in the circumstances considered, it can be approximated as $\min\left(q_n^0/q_n^1, q_n^1/q_n^0\right)$, which can be rewritten as $\exp(-|\lambda_n|)$. Thus, a measure of the reliability of a bit during decoding can be defined as

$$c_n = |\lambda_n| \,, \tag{3.1}$$

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. In [11], thresholds defined as $t = t_0 + t_d i$, where i is the current iteration, are investigated. However, in fixed-point implementations with course quantizations and low saturation limits significant gains are hard to achieve and therefore only constant thresholds are considered in this thesis.

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 α_{nm} to t or $-t$, depending on the hard-decision

value. The other is to set α_{nm} to ∞ 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 [112]. 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. 3.5.

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 σ 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

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

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

$$\begin{aligned} B_{bit} &= P\left(\gamma_n > t | x = -1\right) P\left(x = -1\right) + P\left(\gamma_n < -t | x = 1\right) P\left(x = 1\right) = \\ &= P\left(\gamma_n < -t | x = 1\right) = P\left(r_n < -\frac{t\sigma^2}{2} | x = 1\right) = \\ &= Q\left(-\frac{t\sigma^2}{2}\right). \end{aligned} \tag{3.3}$$

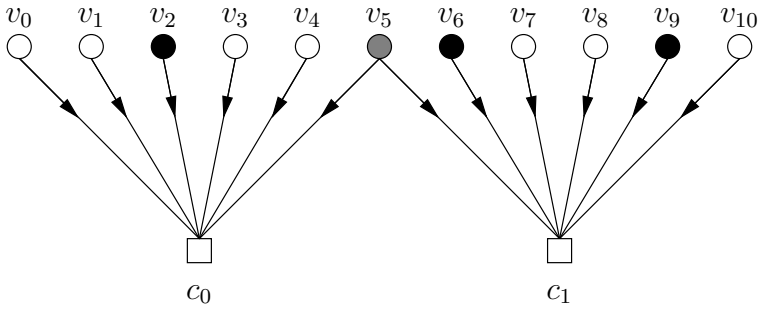
If no corrections of erroneous decisions are done, an incorrect bit decision will result in the whole block being incorrectly decoded. The probability of this happening for a block with N bits is

$$B_{block} = \left(1 - \left(1 - B_{bit}\right)^N\right) \tag{3.4}$$

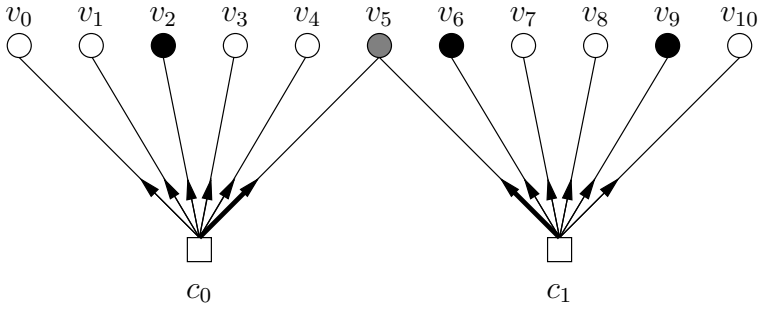
It can thus be seen that the early decision algorithm imposes a lower bound on the block error probability, which grows with increasing SNR (as B_{bit} grows with increasing SNR). This is important, as LDPC decoding is sometimes performed outside the SNR region which is practical to characterize with simulations, and the presence of an error floor will then reduce the expected decoder performance.

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



(a) Variable node update phase. White variable nodes have been decided to 0 and black variable nodes have been decided to 1. v_5 is still undecided.



(b) Check node update phase. Thick arrows indicate enforcing check-to-variable messages.

Figure 3.2 *Principle of enforcing check constraints.*

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

$$\begin{aligned} |\beta_{0,k-1}| &= \left| \left(\prod_{n=0}^{k-2} \operatorname{sign} \alpha_{n,0} \right) \cdot \Phi \left(\sum_{n=0}^{k-2} \Phi \left(|\alpha_{n,0}| \right) \right) \right| \\ &= 2\operatorname{arctanh} \exp \left(\sum_{n=0}^{k-2} \log \tanh \left(\frac{z}{2} \right) \right) \\ &= 2\operatorname{arctanh} \left(\tanh \left(\frac{z}{2} \right)^{k-1} \right). \end{aligned} \tag{3.5}$$

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

$$|\beta_{mn}| \geq 2\operatorname{arctanh} \left(\tanh \left(\frac{z}{2} \right)^{k-1} \right). \tag{3.6}$$

In a fixed-point implementation, the condition can be written

$$|\beta_{mn}| \geq \Phi \left[(k-1) \, \Phi \left[z \right] \right], \tag{3.7}$$

where $\Phi[x]$ denotes the discrete function obtained through rounding of the argument and function values of $\Phi(x)$, as in Fig. 2.17. Typically, z is the maximum representable value in the data representation, and thus $\Phi\left[z\right] = 0$, and $\Phi\left[(k-1)\,\Phi\left[z\right]\right]$ 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. 3.3.4.

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.

3.3 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.

3.3.1 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. 3.3. 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

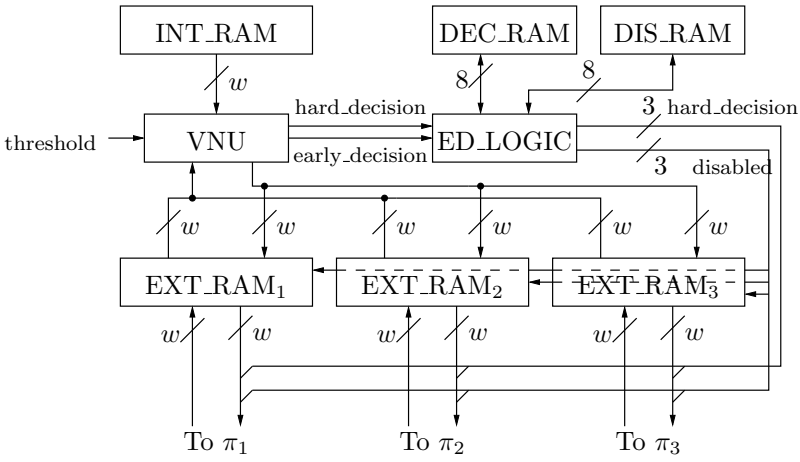


Figure 3.3 *Memory block of* $(3, k)$*-regular early-decision decoder, containing memories used during decoding, and the variable node processing unit.*

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.

3.3.2 Node processing units

In the variable node processing unit, shown in Fig. 3.4, 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. 3.5, 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. 3.3.4.

3.3.3 Early decision logic

The implementation of the ED_LOGIC block is shown in Fig. 3.6, 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. However, for the bits that are disabled, the VNU computations are turned

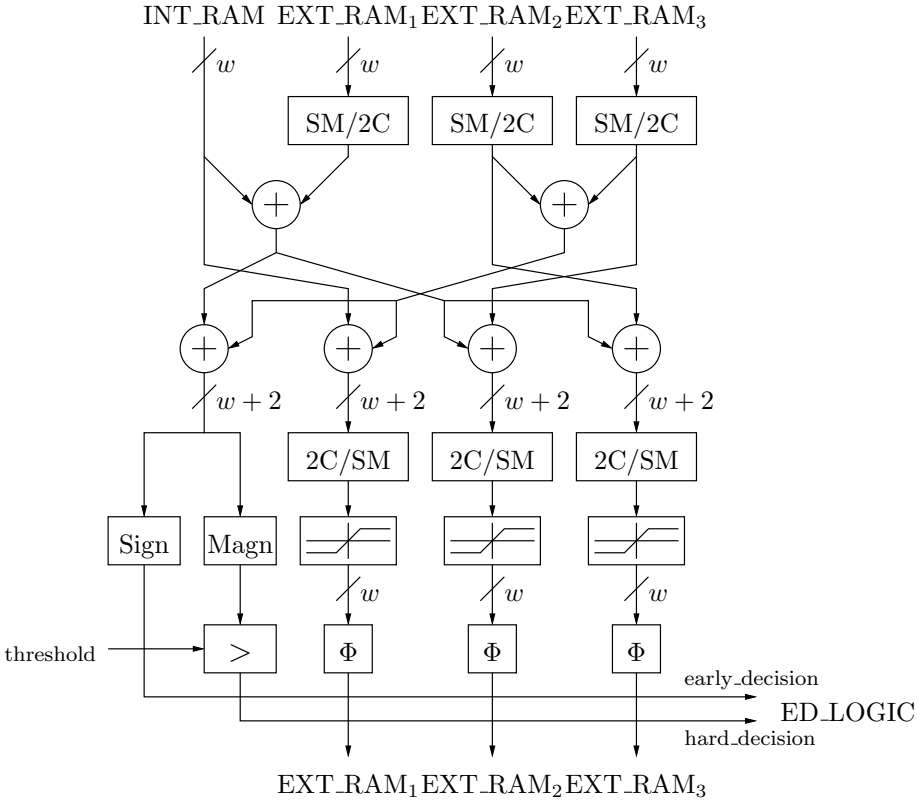


Figure 3.4 *Implementation of variable node processing unit for the* $(3, k)$*-regular early-decision decoder.*

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 is the following three constraints:

$$\begin{aligned} C^1_{x,y} &\neq C^2_{x,y} \pmod 8 \\ C^2_{x,y} &\neq C^3_{x,y} \pmod 8 \\ C^1_{x,y} &\neq C^3_{x,y} \pmod 8 \end{aligned} \tag{3.8}$$

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.

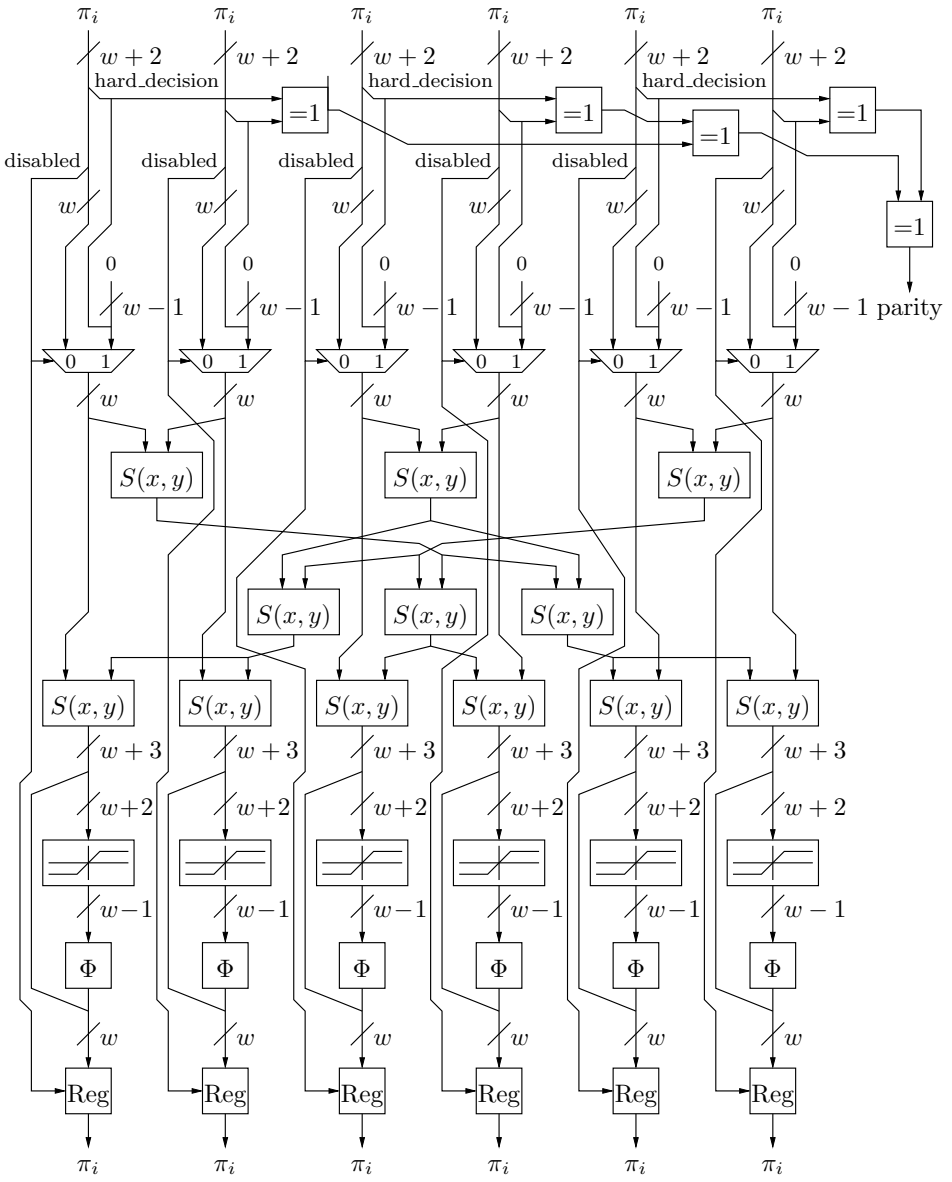


Figure 3.5 *Implementation of check node processing unit for the* $(3, k)$*-regular early-decision decoder.*

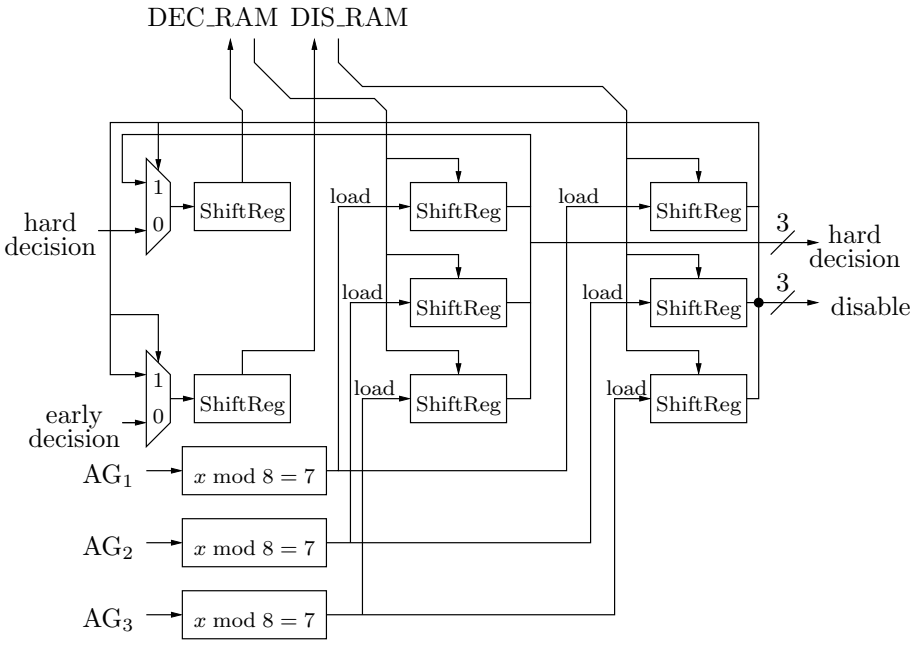


Figure 3.6 *Implementation of early decision logic for the* $(3, k)$*-regular early-decision decoder.*

3.3.4 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 straight-forwardly 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.

3.4 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. 3.5.3, 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.

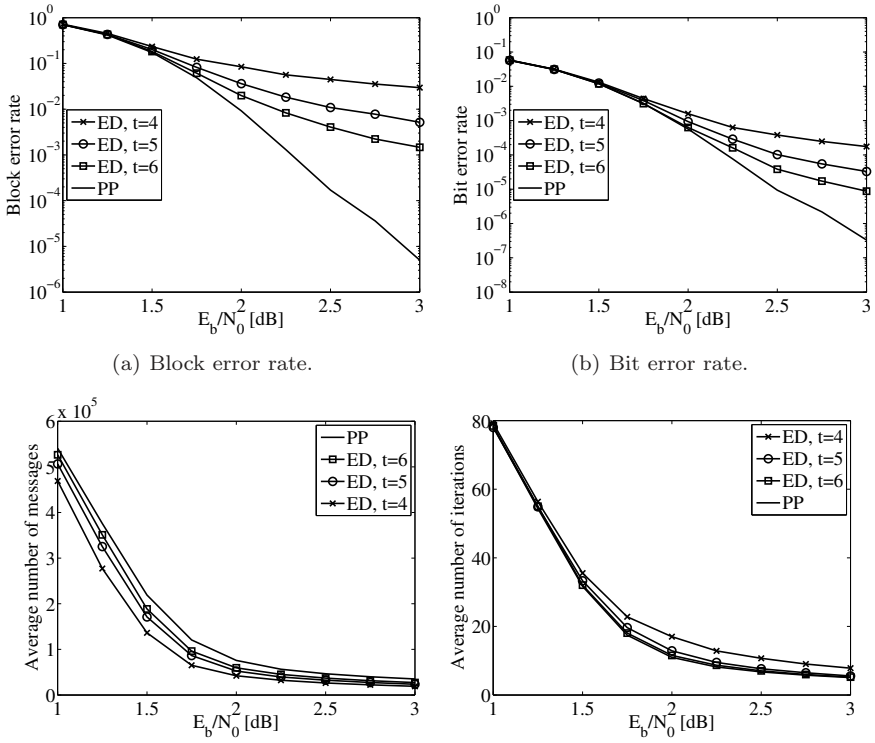
3.5 Simulation results

3.5.1 Choice of threshold

Figure 3.7 shows 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 the complications resulting from the early decision modification.

It is obvious that early decision impacts the block error rate, shown in Fig. 3.7(a), significantly more than the bit error rate, shown in Fig. 3.7(b). 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.



(a) Block error rate.

(b) Bit error rate.

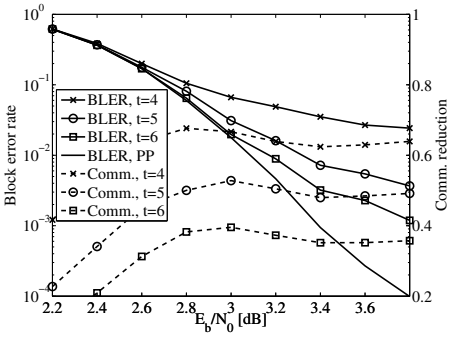
(c) Average number of internal messages.

(d) Average number of iterations.

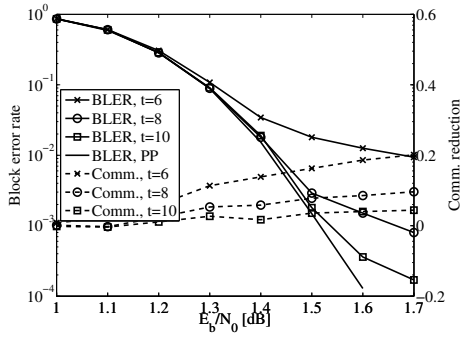
Figure 3.7 *Early decision decoding results of rate-1/2 regular code with* $(N, j, k) = (1152, 3, 6)$

The average number of internal messages sent per block is shown in Fig. 3.7(c), and the average number of iterations is shown in Fig. 3.7(d). 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 comm.red. $= 1 - C_{mod}/C_{ref}$, where C_{mod} is the number of internal messages using the modified algorithm, and C_{ref} is the number of internal messages using the reference algorithm. Results of reducing the number of iterations using enforcing check constraints is shown in Sec. 3.1.4.

The performance of the early decision algorithm for a rate-3/4 $(N, j, k) = (1152, 3, 12)$ code is shown in Fig. 3.8(a), and for a rate-1/2 $(N, j, k) = (9216, 3, 6)$ code in Fig. 3.8(b). The performance for the higher rate code is significantly better than for the rate-1/2 code of the same block size. Similarly, the performance of the longer code is worse than for the shorter. It is intuitive that codes with higher

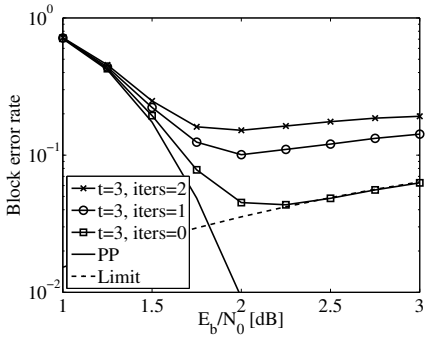


(a) Rate-3/4 code, $(N, j, k) = (1152, 3, 12)$.

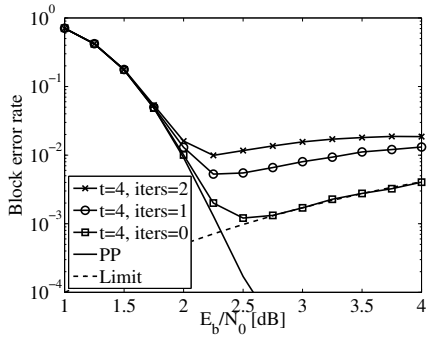


(b) Rate-1/2 code, $(N, j, k) = (9216, 3, 6)$.

Figure 3.8 *Performance of early decision decoding algorithm on codes with high rates and codes with long block lengths.*



(a) $t = 3$.

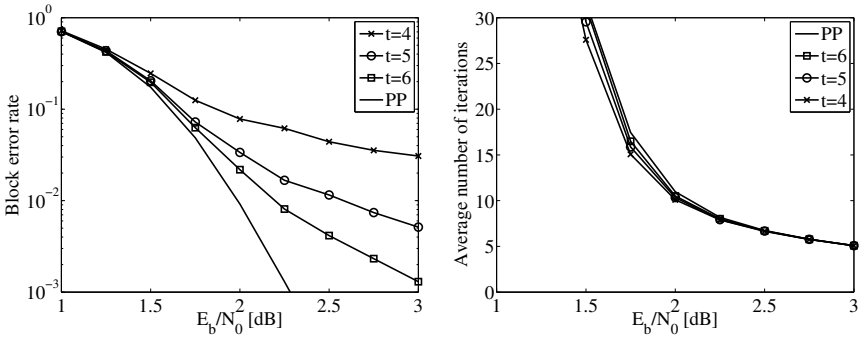


(b) $t = 4$.

Figure 3.9 *Error floor resulting from early decision decoding and making decisions only in the initial iterations.*

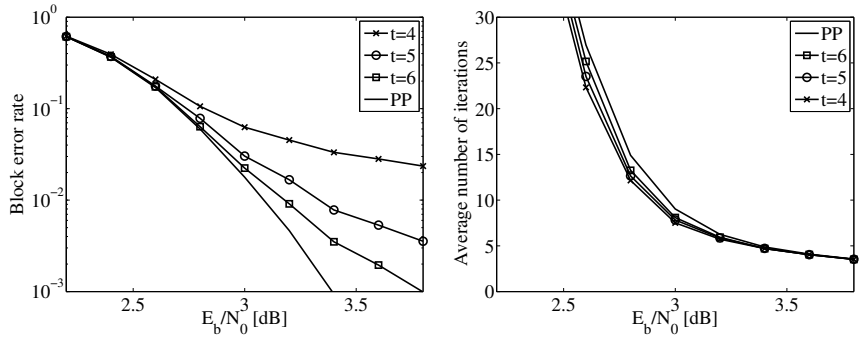
error correction capabilities (such as lower rate and larger block size) allow noisier signals, which increases the chance of making wrong decisions and thereby reducing performance.

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. 3.9(a) and for $t = 4$ in Fig. 3.9(b). 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



(a) Block error rate. (b) Average number of iterations.

Figure 3.10 *Early decision decoding of* $(N, j, k) = (1152, 3, 6)$ *code using enforcing checks.*



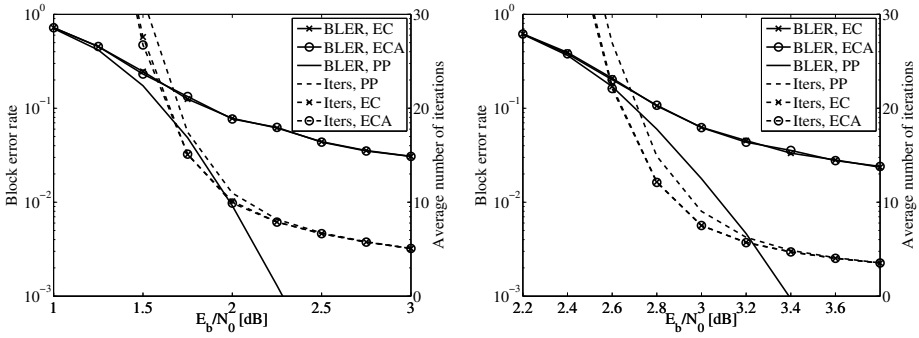
(a) Block error rate. (b) Average number of iterations.

Figure 3.11 *Early decision decoding of* $(N, j, k) = (1152, 3, 12)$ *code using enforcing checks.*

of the messages caused by cycles in the graph causes the reliabilities of the bits to escalate.

3.5.2 Enforcing check constraints

Enforcing check constraints were introduced in Sec. 3.1.4 as a way of reducing 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. 3.10 and for the $(N, j, k) = (1152, 3, 12)$ code in Fig. 3.11. By comparing with Fig. 3.7(a) and Fig. 3.8(a) it can be seen that the error correction capability of the decoder is not additionally worsened by the enforcing check constraints



(a) $(N, j, k) = (1152, 3, 6)$ (b) $(N, j, k) = (1152, 3, 12)$

Figure 3.12 *Early decision decoding of* $(N, j, k) = (1152, 3, 6)$ *and* $(N, j, k) = (1152, 3, 12)$ *codes using enforcing check approximation.* $t = 4$.

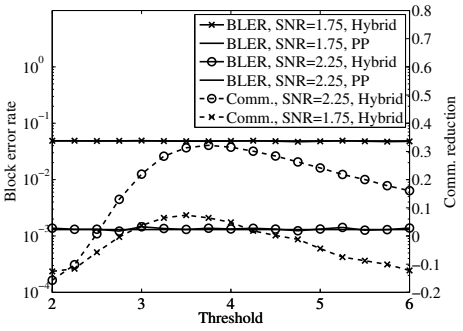
modification. However, the average number of iterations is significantly reduced, and for low SNRs the early decision algorithm requires an even lower number of iterations than the standard algorithm as graph inconsistencies are discovered for blocks that the standard algorithm will not be able to successfully decode.

The enforcing check approximation is investigated in Figs. 3.12(a) and 3.12(b) 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.

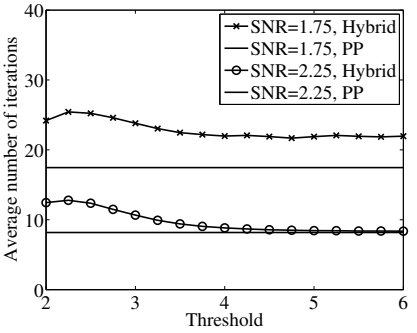
3.5.3 Hybrid decoding

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

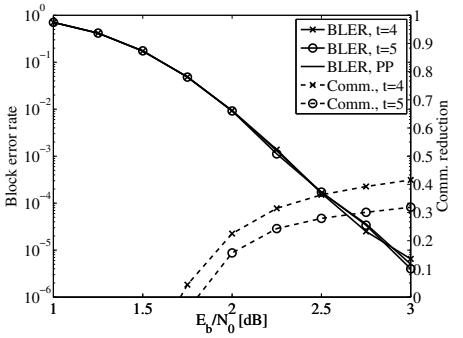


(a) Block error rate, communication reduction.

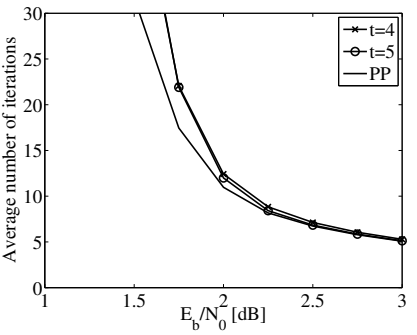


(b) Average number of iterations.

Figure 3.13 *Decoding of* $(N, j, k) = (1152, 3, 6)$ *code using hybrid algorithm showing performance as function of threshold.*



(a) Block error rate, communication reduction.

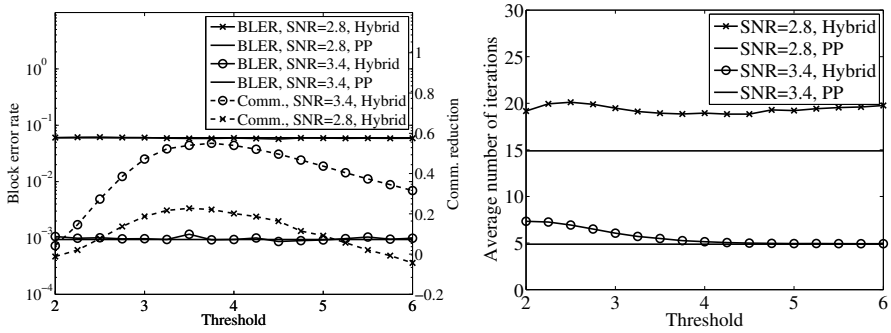


(b) Average number of iterations.

Figure 3.14 *Decoding of* $(N, j, k) = (1152, 3, 6)$ *code using hybrid algorithm showing performance as function of SNR for threshold 4 (optimal) and 5.*

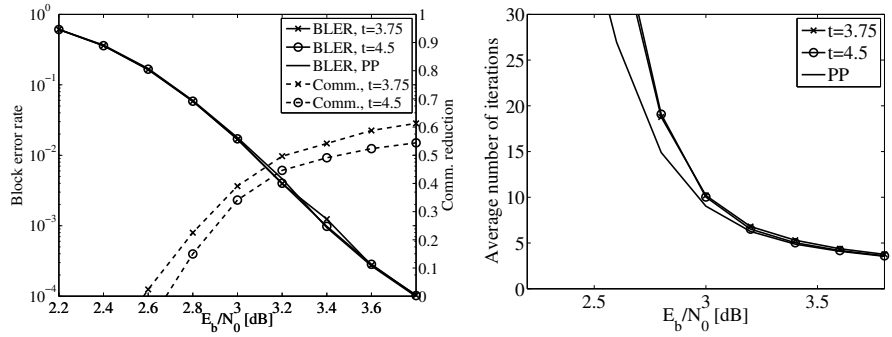
of the hybrid algorithm is indistinguishable from that of the standard algorithm (in the SNR region simulated), whereas the internal communication of the decoder is much lower. In fact, for the rate-1/2 codes, no decoder errors have occurred, and therefore all errors introduced by wrong decisions of 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. 3.13 and 3.14, for the $(N, j, k) = (1152, 3, 12)$ code in Figs. 3.15 and 3.16, and for the $(N, j, k) = (9216, 3, 6)$ code in Figs. 3.17 and 3.18. In the plots, the average



(a) Block error rate, communication reduction. (b) Average number of iterations.

Figure 3.15 *Decoding of* $(N, j, k) = (1152, 3, 12)$ *code using hybrid algorithm showing performance as function of threshold.*



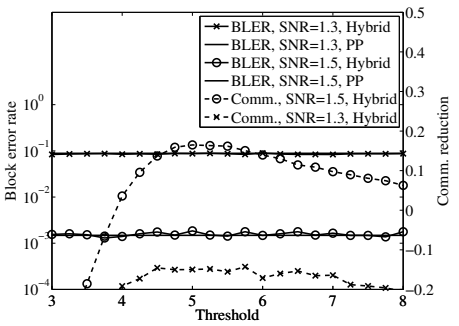
(a) Block error rate, communication reduction. (b) Average number of iterations.

Figure 3.16 *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.*

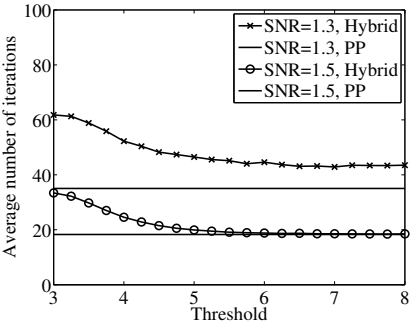
number of iterations denote the sum of the number of iterations required for the early decision pass and the standard algorithm pass.

3.5.4 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 were defined in Sec. 2.7, and are the initial values $C^2_{x,y}$ and $C^3_{x,y}$ of the

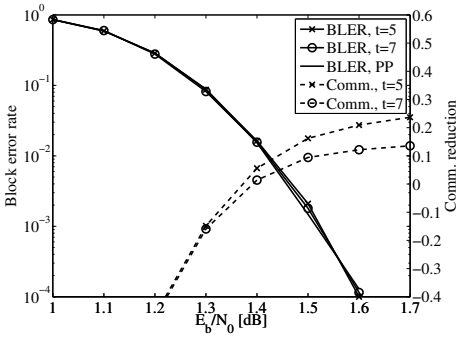


(a) Block error rate, communication reduction.

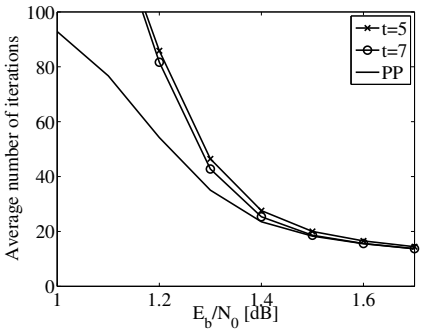


(b) Average number of iterations.

Figure 3.17 *Decoding of* $(N, j, k) = (9216, 3, 6)$ *code using hybrid algorithm showing performance as function of threshold.*



(a) Block error rate, communication reduction.

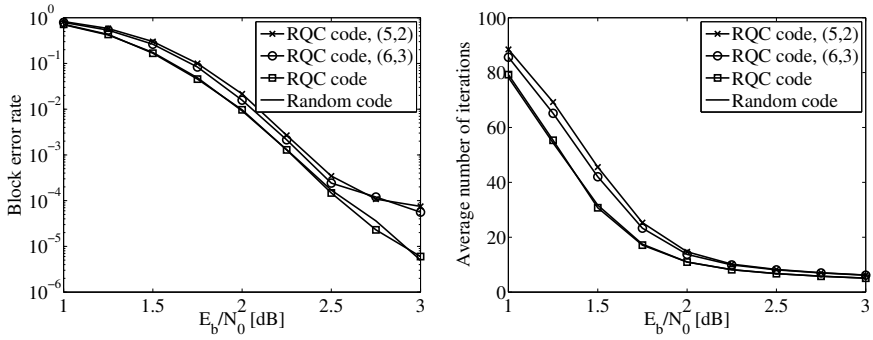


(b) Average number of iterations.

Figure 3.18 *Decoding of* $(N, j, k) = (9216, 3, 6)$ *code using hybrid algorithm showing performance as function of SNR for threshold 5 (optimal) and 7.*

address generators $\mathrm{AG}^2_{x,y}$ and $\mathrm{AG}^3_{x,y}$, the permutation functions $\Psi_{l,n}$ and $\Omega_{l,m}$, and the contents of the row and column permutation ROMs for π_3, $\psi_{l,n}$ and $\omega_{l,m}$. Furthermore, the initial values of the address generators were chosen to satisfy (2.25) to ensure high girth, as well as (3.8) to ensure that an early decision decoder implementation is possible.

In Fig. 3.19, 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



(a) Block error rate.

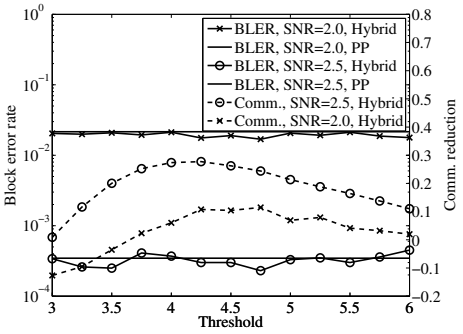
(b) Average number of iterations.

Figure 3.19 *Decoding of $(N, j, k) = (1152, 3, 6)$ randomized QC-LDPC code, using the floating-point sum-product algorithm and fixed-point precision. (w, w_f) denotes quantization to w bits where w_f bits are fraction bits.*

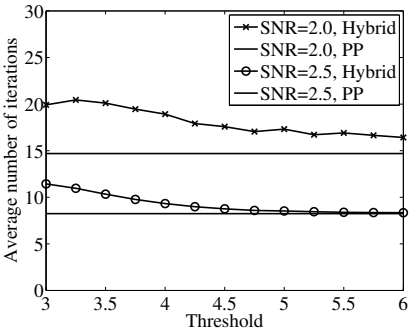
increasing the number of integer bits did not increase the performance. Down to a block error rate of 10^{-4}, both fixed-point representations show a performance hit of less than 0.1 dB. The performance difference between wordlengths of 5 and 6 bits is rather small, which is consistent with previous results [111], 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. 3.20, 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, fox 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. 3.21, 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 the sum-product decoder would fail. Regarding the reduction of internal communication, results close to those obtained using floating-point data representation are achieved.

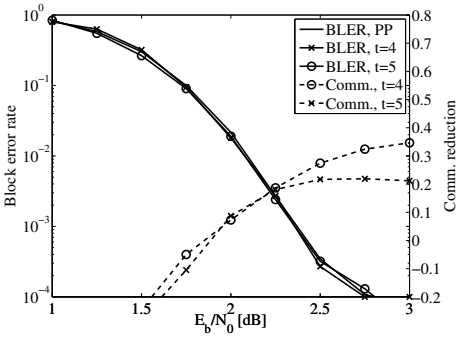


(a) Block error rate, communication reduction.

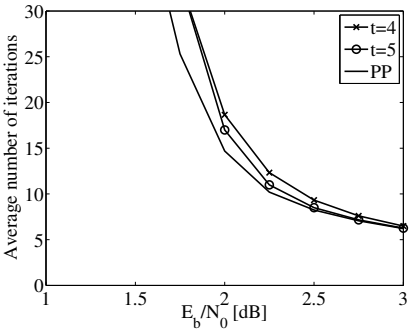


(b) Average number of iterations.

Figure 3.20 *Decoding of* $(N, j, k) = (1152, 3, 6)$ *randomized QC-LDPC code using* $(w, w_f) = (5, 2)$ *precision and the hybrid algorithm.*



(a) Block error rate, communication reduction.



(b) Average number of iterations.

Figure 3.21 *Decoding of* $(N, j, k) = (1152, 3, 6)$ *randomized QC-LDPC code using* $(w, w_f) = (5, 2)$ *precision and the hybrid algorithm.*

3.6 Synthesis results

The reference sum-product decoder and the early decision decoder as described in Sec. 2.7 and Sec. 3.3, respectively, 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 3.1, 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

	Reference decoder	Early-decision decoder
Slice utilization [%]	14.5	21.0
Number of block RAMs	54	54
Maximum frequency [MHz]	302	220
Logic energy [pJ/iteration/bit]	499	291
Clock energy [pJ/iteration/bit]	141	190

Table 3.1 *Synthesis results for the reference sum-product decoder and the early-decision decoder in a Xilinx Virtex 5 FPGA.*

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. 3.6). 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.

4

Rate-compatible LDPC codes

In this chapter, new results related to rate-compatible LDPC codes are presented. In Sec. 4.1, an ILP optimization approach to the design of puncturing patterns for QC-LDPC codes is proposed. The algorithm generates puncturing sequences that are optimal in terms of recovery time and eliminates the element of randomness that is inherent in some of the greedy algorithms [46, 47]. In Sec. 4.2, an improved algorithm for the decoding of rate-compatible LDPC codes is presented. The algorithm systematically creates a new parity-check matrix used by the decoding algorithm. In the new matrix, the punctured nodes are not present, and thereby the convergence speed is significantly improved. In Sec. 4.3, a check-serial architecture for the decoding of rate-compatible LDPC codes using the proposed check-merging algorithm is presented. The proposed architecture is an extension of the work done in [66]. Finally, Sec. 4.4 and Sec. 4.5 contain simulation results of the check-merging decoding algorithm and synthesis results of the decoder architecture, respectively.

Previous work in the area is presented in [43] and [60]. In [43], a method of designing rate-less codes through check node splitting is proposed. Using the method, high-degree check nodes are split in two with equal weight to add additional parity bits to the code. In [60], a decoding algorithm using check node merging for dual-diagonal LDPC codes is suggested and the algorithm is analyzed using density evolution.

Section 4.1 has been published in [15], and Sec. 4.3 has been published in [10].

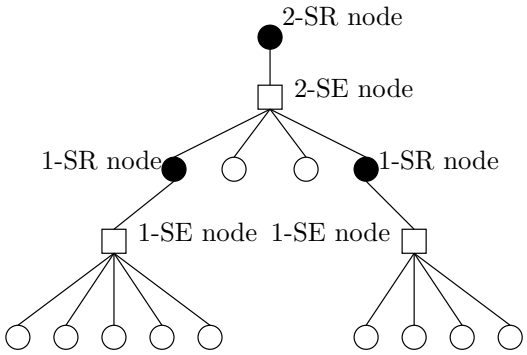


Figure 4.1 *Recovery tree of a 2-SR node. The filled nodes denote punctured bits in the codeword. Bits that are recovered after k iterations are denoted k-SR. Recovering check nodes are denoted k-SE.*

4.1 Design of puncturing patterns

The concept of k-SR nodes was explained in Sec. 2.5.1. Here, an ILP-based optimization algorithm for the design of puncturing sequences is proposed. The algorithm uses the number of k-SR nodes as a cost measure. The primary goal of the algorithm is to optimize the puncturing pattern for fast recovery of the punctured nodes. Given a parity check matrix, it maximizes the number of 1-SR nodes. Following that, it maximizes the number of punctured nodes for increasing values of K, given the puncturing patterns obtained in earlier iterations and the constraint that only k-SR nodes with $k \leq K$ are allowed.

In Sec. 4.1.1, needed notations are introduced. In Sec. 4.1.2, the optimization problem with variable definitions, goal function and constraints is described. Also, design cuts to decrease the optimization time are discussed. Then, the proposed algorithm for designing the puncturing patterns using the ILP optimization problem is presented in Sec. 4.1.3.

4.1.1 Preliminaries

Using a message passing algorithm to decode a punctured block, the check nodes adjacent to a punctured bit node will propagate null information until the punctured variable node has received a non-zero message from another check node. Thus, the concept of recoverable nodes (k-SR nodes) was introduced in Sec. 2.5.1. Here, the concept is extended for check nodes in order to enable the optimization problem formulation.

As shown in Fig. 4.1, a check node is denoted 1-step enabled (1-SE) if it involves a singular punctured node and is used to recover that node in the first iteration. Furthermore, a punctured variable node is denoted 1-SR if it is connected to a

1-SE check node. Generally, a check node is denoted k-SE if it begins to communicate non-null information in the kth iteration and has been chosen to recover a punctured node. A punctured variable node is denoted k-SR if it is connected to a k-SE check node but to no lower k-SE check node.

It is possible to create puncturing patterns that contain nodes that will never be recovered. Such nodes thus never change value and severely impact the decoding performance. In [46], it was shown that decreasing the recovery time increases the reliability of the recovery information, and it was then conjectured that increasing the number of nodes with fast recovery enhances the performance of the puncturing pattern.

4.1.2 Optimization problem

The optimization problem is formulated to maximize the number of k-SR variable nodes, where $k < K$ for a given value of K. As secondary optimization criteria, the algorithm chooses k-SR variable nodes and k-SE check nodes such that first the sum of the degrees of the k-SR variable nodes is minimized, and then the sum of the degrees of the k-SE check nodes is minimized, as low-degree nodes were reported to give good results in [83].

The following constants are defined:

$\mathbf{H}_b$: the base matrix of the code, as defined in Sec. 2.3.2
M: the number of rows in $\mathbf{H}_b$
N: the number of columns in $\mathbf{H}_b$
$\mathbf{H}_a$: the $z = 1$ expansion of $\mathbf{H}_b$ (each -1 element replaced by 0 and each non-negative element replaced by 1)
C_p: the cost for each non-punctured node
C_r: the recoverability cost
C_v: the variable node degree cost
C_c: the check node degree cost
K: the maximum allowed recovery delay (maximum k-SR node)
F: set of nodes for which puncturing is forced

Furthermore, the set Q_m of involved variable nodes in check node m and the set R_n of involved check nodes in variable node n are defined as

$$Q_m = \{n : \mathbf{H}_a(m, n) = 1\} \tag{4.1}$$
$$R_n = \{m : \mathbf{H}_a(m, n) = 1\} \tag{4.2}$$

The following binary variables are defined:

p_n: 1 if variable node n is punctured, 0 otherwise
$s_{k,n}$: 1 if variable node n is a k-SR node, 0 otherwise, $k = 1, \ldots, K$
$c_{k,m}$: 1 if check node m is a k-SE node, 0 otherwise, $k = 1, \ldots, K$
$r_{k,n}$: 1 if variable node n is still unrecovered at level k (after k iterations), 0 otherwise, $k = 0, \ldots, K$

The goal function G is to be minimized and is written

$$G = C_p G_p + C_r G_r + C_v G_v + C_c G_c, \tag{4.3}$$

where

$$G_p = \sum_{n=0}^{N-1} (1 - p_n) \tag{4.4}$$

$$G_r = \sum_{k=1}^{K} \sum_{n=0}^{N-1} k s_{k,n} \tag{4.5}$$

$$G_v = \sum_{n=0}^{N-1} |R_n| p_n \tag{4.6}$$

$$G_c = \sum_{k=1}^{K} \sum_{m=0}^{M-1} |Q_m| c_{k,m} \tag{4.7}$$

G_p maximizes the number of punctured nodes, whereas G_r ensures as fast recovery as possible. Secondary objectives are achieved by G_v and G_c that minimize the degrees of the punctured variable nodes and the check nodes used for recovery, respectively. By setting $C_p \gg C_r \gg C_v \gg C_c$ it can be ensured that the partial objectives are achieved with falling priorities.

The goal function is to be minimized subject to the following constraints:

Initialize unrecovered nodes: For $n = 0, \ldots, N-1$, let

$$r_{0,n} = p_n. \tag{4.8}$$

At level 0, only the non-punctured nodes (0-SR nodes) are recovered.

Recover nodes: For $k = 0, \ldots, K-1$ and $n = 0, \ldots, N-1$,

$$r_{k+1,n} \geq r_{k,n} - s_{k,n}. \tag{4.9}$$

This constraint makes sure that a punctured node will be unrecovered until the level where it is marked as a k-SR node.

Require recovered neighbours: For $m = 0, \ldots, M-1$, $k = 1, \ldots, K$ and $n = 0, \ldots, N-1$,

$$r_{k,n} \geq -|Q_m|(2 - s_{k,n} - c_{k,m}) + \sum_{n' \in Q_m \setminus n} r_{k-1,n'}. \tag{4.10}$$

Thus, if check node m is used to recover variable node n at level k, the first term of the right hand side will be zero and the variable node is allowed to be recovered ($r_{k,n} = 0$) only if the other check node's neighbours are already recovered (the sum is zero). If check node m is not k-SE or variable node n is not k-SR, the condition imposes no additional constraints.

Require k-SE neighbour: For $k = 1, \ldots, K$ and $n = 0, \ldots, N-1$,

$$s_{k,n} \leq \sum_{m \in R_n} c_{k,m}. \tag{4.11}$$

The constraint states that, in order for a variable node n to be k-SR, one of its check node neighbours must be k-SE.

Require recovering of all nodes: For $n = 0, \ldots, N-1$,

$$r_{K,n} = 0, \tag{4.12}$$

which states that all variable nodes must be recovered at level K.

Forced punctures: For $n \in F$, $p_n = 1$. Thus, for the nodes in the set F, puncturing is forced. However, the recovery level is not constrained.

In addition to the optimization constraints, two cuts can be used to decrease the optimization time without affecting optimality of the results.

Cut 1: For $m = 0, \ldots, M-1$, $k = 0, \ldots, K-1$ and $n \in Q_m$,

$$c_{k+1,m} \geq r_{k,n} - \sum_{n' \in Q_m \setminus n} r_{k,n'}. \tag{4.13}$$

The cut implies that, whenever a variable node is the only unrecovered node in a check m at level k, check node m must be $k+1$-SE.

Cut 2: For $m = 0, \ldots, M-1$, $k = 0, \ldots, K-1$ and $n \in Q_m$,

$$s_{k+1,n} \geq r_{k,n} - \sum_{n' \in Q_m \setminus n} r_{k,n'}. \tag{4.14}$$

The cut implies that, whenever a variable n is the only unrecovered node in a check at level k, variable node n must be $k+1$-SR.

4.1.3 Puncturing pattern design

The following algorithm is proposed for the design of puncturing sequences for a class of codes based on a base matrix $\mathbf{H}_b$ with different expansion factors z.

1. Initialize the puncturing sequence $p = ()$ and set the maximum recovery delay $K = 1$. Set the costs $C_p \gg C_r \gg C_v \gg C_c$.

2. Set the forced puncturing pattern $F = p$ and solve the optimization problem on $\mathbf{H}_b$.

3. Let q denote the puncturing pattern solution. Order $q \setminus p$ first by increasing recovery delay and then by increasing node degree, and append the sequence to p.

4. If a pre-defined maximum recovery delay K has been reached, quit. Otherwise, increase K and go to Step 2.

The result of the above procedure is an ordered sequence p defining a puncturing order for the blocks of the base matrix $\mathbf{H}_b$. For the parity-check matrix $\mathbf{H}$ with expansion factor z, the puncturing sequence p_z is defined by replacing each element a in p with the sequence $za, \ldots, z(a+1)-1$.

Simulation results of the proposed puncturing pattern design algorithm are in Sec. 4.4.1.

4.2 Check-merging decoding algorithm

Assume that a code defined by a parity-check matrix $\mathbf{H}$ of size $M \times N$ is used for a communications system. Also assume that signal conditions allow for a puncturing pattern $\mathbf{p} = (p_0, \ldots, p_{L-1})$ to be used. The sum-product decoding algorithm is defined on $\mathbf{H}$ and initializes the prior log-likelihood ratios of the punctured nodes $\gamma_{p_0}, \ldots, \gamma_{p_{L-1}}$ to zero. However, by merging the check nodes involving each punctured bit and purging the rows and columns associated with them, a modified parity-check matrix $\mathbf{H}_P$ of size $(M-L) \times (N-L)$ can be defined. Decoding using $\mathbf{H}_P$ instead of $\mathbf{H}$ can in many circumstances reduce the convergence time for the decoding algorithm significantly, while the computational complexity of each iteration is reduced.

The main motivation for the proposed algorithm is systems with varying signal conditions, i.e., most wireless systems. In such systems, rate-compatible codes can be used to adapt the protection better to the current channel. The result is that most of the transmitted blocks use some amount of puncturing, and then increasing the throughput of the decoder for punctured blocks is motivated.

4.2.1 Defining $\mathbf{H}_P$

In the following, let all matrix operations be defined using modulo-2 arithmetic. $\mathbf{H}_P$ can be defined if and only if the columns of $\mathbf{H}$ corresponding to the punctured bits $\mathbf{p}$ are linearly independent. The implications of the requirement and how to construct puncturing patterns that satisfy it are discussed in Sec. 4.2.3.

Consider a sub-matrix $\mathbf{B}$ of size $M \times L$ formed by taking the columns of $\mathbf{H}$ corresponding to the punctured bits $\mathbf{p}$. By assumption, the columns of $\mathbf{B}$ are linearly independent. Thus, $\mathbf{B}$ has a set of L rows that are linearly independent. Let $\mathbf{c} = (c_0, \ldots, c_{L-1})$ denote such a set. $\mathbf{c}$ is thus the set of check nodes to purge along with the punctured variable nodes.

Given a set of punctured variable nodes $\mathbf{p} = (p_0, \ldots, p_{L-1})$ and purged check nodes $\mathbf{c} = (c_0, \ldots, c_{L-1})$, perform column and row reordering of $\mathbf{H}$ such that the punctured variable nodes $\mathbf{p}$ and the purged check nodes $\mathbf{c}$ end up at the right and bottom, respectively. Denote the resulting matrix $\tilde{\mathbf{H}}$. $\tilde{\mathbf{H}}$ thus has the following structure:

$$\tilde{\mathbf{H}} = \begin{pmatrix} \mathbf{H}_A & \mathbf{B}_A \\ \mathbf{H}_B & \mathbf{B}_B \end{pmatrix}, \tag{4.15}$$

where $\mathbf{B}_A$ and $\mathbf{B}_B$ have sizes of $(M-L)\times L$ and $L\times L$, respectively. Further, $\mathbf{B}_B$ is non-singular as its rows are linearly independent.

Define the matrix $\mathbf{A}$ of size $M\times M$ as

$$\mathbf{A}=\begin{pmatrix}\mathbf{I} & \mathbf{B}_A\\ \mathbf{0} & \mathbf{B}_B\end{pmatrix}, \tag{4.16}$$

where $\mathbf{I}$ is the identity matrix of size $(M-L)\times(M-L)$. Since $\mathbf{B}_B$ is a non-singular matrix, $\mathbf{A}$ is also non-singular and its inverse is

$$\mathbf{A}^{-1}=\begin{pmatrix}\mathbf{I} & \mathbf{B}_A\mathbf{B}_B^{-1}\\ \mathbf{0} & \mathbf{B}_B^{-1}\end{pmatrix}. \tag{4.17}$$

In order to define $\mathbf{H}_P$, first define the matrix $\tilde{\mathbf{H}}_R$ of size $M\times N$ as

$$\tilde{\mathbf{H}}_R=\mathbf{A}^{-1}\tilde{\mathbf{H}}=\begin{pmatrix}\mathbf{H}_A+\mathbf{B}_A\mathbf{B}_B^{-1}\mathbf{H}_B & \mathbf{0}\\ \mathbf{B}_B^{-1}\mathbf{H}_B & \mathbf{I}\end{pmatrix}. \tag{4.18}$$

As $\tilde{\mathbf{H}}_R$ is obtained by reversible row operations on $\mathbf{H}$, the matrices have the same null space and thus define the same code. However, considering the behavior of sum-product decoding with punctured bits over $\tilde{\mathbf{H}}_R$, two observations can be made. First, consider the check-to-variable message $\beta_{c_l n_0}$ from the purged check node c_l to one of its non-punctured neighbors $n_0\in\mathcal{N}(c_l)\setminus p_l$, as depicted in Fig. 4.2(a).

$$\begin{aligned}\beta_{c_l n_0}&=\left(\prod_{n'\in\mathcal{N}(c_l)\setminus n_0}\operatorname{sign}\alpha_{n'c_l}\right)\cdot\Phi\left(\sum_{n'\in\mathcal{N}(c_l)\setminus n_0}\Phi\left(|\alpha_{n'c_l}|\right)\right)=\\&=\left(\prod_{n'\in\mathcal{N}(c_l)\setminus n_0}\operatorname{sign}\alpha_{n'c_l}\right)\cdot\Phi\left(\Phi\left(|\alpha_{p_l c_l}|\right)+\sum_{n'\in\mathcal{N}(c_l)\setminus\{n_0,p_l\}}\Phi\left(|\alpha_{n'c_l}|\right)\right)=0,\end{aligned} \tag{4.19}$$

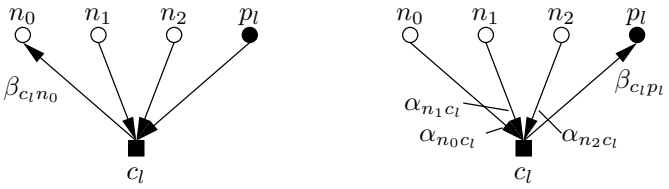
since $\Phi\left(|\alpha_{p_l c_l}|\right)=\Phi\left(\gamma_{p_l}\right)=\Phi(0)=\infty$. Thus, the purged check nodes $\mathbf{c}$ affect neither the computations of the variable-to-check messages nor the pseudo-posterior likelihoods at their neighboring variable nodes.

Second, consider the variable nodes involved in check node c_l, as depicted in Fig. 4.2(b). The hard-decision of p_l can be written

$$\operatorname{sign}\lambda_{p_l}=\operatorname{sign}\beta_{c_l p_l}=\prod_{n\in\mathcal{N}(c_l)\setminus p_l}\operatorname{sign}\alpha_{nc_l}=\prod_{n\in\mathcal{N}(c_l)\setminus p_l}\operatorname{sign}\lambda_n. \tag{4.20}$$

Thus, in a converged state, where all non-purged check nodes are fulfilled and the hard-decision values of the variable nodes do not change, c_l will eventually also be fulfilled. It follows from these two observations that the purged check nodes do not affect the behavior of the decoding algorithm, apart from possibly delaying convergence. $\mathbf{H}_P$ is thus defined as the matrix $\tilde{\mathbf{H}}_R$ with the purged check nodes and punctured variable nodes removed:

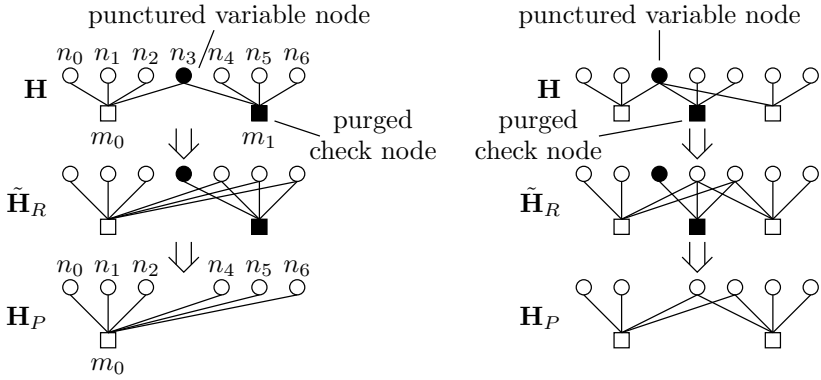
$$\mathbf{H}_P=\mathbf{H}_A+\mathbf{B}_A\mathbf{B}_B^{-1}\mathbf{H}_B. \tag{4.21}$$



(a) Message from purged check node c_l to non-punctured node n_0.

(b) Message from purged check node c_l to punctured node p_l.

Figure 4.2 *Check-to-variable messages from purged check nodes.*



(a) Degree-2 variable node.

(b) Degree-3 variable node.

Figure 4.3 *Merging of check nodes for punctured variable nodes.*

4.2.2 Algorithmic properties of decoding with $\mathbf{H}_P$

In the case when all punctured variable nodes are of degree 2, decoding over $\mathbf{H}_P$ has some interesting properties. Consider first the case when the punctured variable nodes do not share any check node and $\mathbf{H}$ is free of length-4 cycles. Given these assumptions, $\mathbf{B}_B$ is a permuted identity matrix and $\mathbf{B}_A$ is a permuted identity matrix with possible additional zero-filled rows. In this case, $\mathbf{H}_P = \mathbf{H}_A + \mathbf{B}_A\mathbf{B}_B^T\mathbf{H}_B$, and each purged check node c_l is merged with the other neighbor of its corresponding punctured variable p_l, as shown in Fig. 4.3(a).

Denoting the variable and check node operations of node k by V_k and C_k, respectively, consider the messages $\beta^i_{m_0 n_0}$ in iteration i in the top and bottom

graphs in Fig. 4.3(a). In the top graph using $\mathbf{H}$,

$$\begin{aligned}\beta^i_{m_0n_0} &= \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \alpha^{i-1}_{n_3m_0}) = \\ &= \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \mathrm{V}_3(\beta^{i-1}_{m_1n_3})) = \\ &= \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \beta^{i-1}_{m_1n_3}) = \\ &= \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \mathrm{C}_1(\alpha^{i-2}_{n_4m_1}, \alpha^{i-2}_{n_5m_1}, \alpha^{i-2}_{n_6m_1})) = \\ &= \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \alpha^{i-2}_{n_4m_1}, \alpha^{i-2}_{n_5m_1}, \alpha^{i-2}_{n_6m_1}),\end{aligned} \tag{4.22}$$

whereas in the bottom graph using $\mathbf{H}_P$,

$$\beta^i_{m_0n_0} = \mathrm{C}_0(\alpha^{i-1}_{n_1m_0}, \alpha^{i-1}_{n_2m_0}, \alpha^{i-1}_{n_4m_0}, \alpha^{i-1}_{n_5m_0}, \alpha^{i-1}_{n_6m_0}). \tag{4.23}$$

The operations performed when decoding using $\mathbf{H}_P$ are thus the same as when decoding using $\mathbf{H}$, and differ only in the speed of the propagation of the messages. It can thus be expected that decoding using $\mathbf{H}_P$ achieves convergence in less iterations than using $\mathbf{H}$. As seen in Sec. 4.4, this is also the case.

When the punctured variable nodes share check nodes, (4.23) can be shown to still hold by repeated application of (4.21) for independent sets of nodes. However, as (4.21) may introduce length-4 cycles, (4.22) may possibly contain duplicate messages which are not present in (4.23). Although the presence of duplicate messages results in a computational change in the algorithm, the defined code is the same.

The necessary conditions of punctured degree-2 variable nodes are relatively strict, but are often met in practice due to the simplified encoder designs for these types of codes [69]. Puncturing of higher-degree nodes is possible and an example of a degree-3 node is shown in Fig. 4.3(b). However, for higher-degree nodes (4.18) normally introduces length-4 cycles which reduce the performance of the decoding algorithm.

4.2.3 Choosing the puncturing sequence p

The requirement that the columns in $\mathbf{H}$ corresponding to the punctured nodes must be linearly independent is related to the concepts of k-SR variable nodes (defined in Sec. 2.5.1) and k-SE check nodes (defined in Sec. 4.1.1). However, the requirement of linear independence is less strict. First, it is shown that the columns in $\mathbf{H}$ corresponding to a set of k-SR nodes are linearly independent. Then the implications of linear independence in terms of recoverability is discussed.

Assume that the parity-check matrix $\mathbf{H}$ is given, and let $\mathbf{p}$ denote a set of recoverable punctured nodes. Further, let $\mathbf{c}$ denote the set of check nodes used for the recovery of the punctured nodes. Denote the k-SR nodes by $\mathbf{p}_k$ and the corresponding k-SE check nodes by $\mathbf{c}_k$. Now, define $\tilde{\mathbf{H}}$ as in Sec. 4.2.1 and consider the $\mathbf{B}_B$ sub-matrix. Obviously, the rows $\mathbf{c}_1$ and columns $\mathbf{p}_1$ corresponding to the 1-SR nodes form the identity matrix. Generally, the checks $\mathbf{c}_k$ can only contain l-SR nodes where $l < k$, in addition to the k-SR node it is recovering. It follows that $\mathbf{B}_B$ is upper triangular with 1s along the diagonal, and is therefore non-singular.

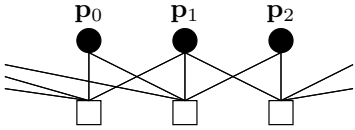


Figure 4.4 *Example of non-recoverable punctured nodes.*

Thus, as a set of recoverable nodes implies that $\mathbf{H}_P$ can be defined, any of the available algorithms to determine k-SR puncturing sequences can be used also to determine sets of punctured nodes used for check-merged decoding.

The converse, however, is not true. Consider for example the punctured variable nodes shown in Fig. 4.4. None of the nodes are recoverable, as all their check node neighbors require at least one other of the punctured variable nodes to be recovered in order to produce non-zero messages. However, the parity-check matrix corresponding to the graph is

$$\mathbf{H} = \begin{pmatrix} \cdots & 1 & 1 & 0 \\ \cdots & 1 & 1 & 1 \\ \cdots & 0 & 1 & 1 \\ \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & 0 & 0 \end{pmatrix}, \tag{4.24}$$

where the three rightmost columns corresponding to the punctured variable nodes are linearly independent.

4.3 Rate-compatible QC-LDPC code decoder

In this section, a check-merging decoder architecture for QC-LDPC codes is presented. The architecture is based on the reference decoder in Sec. 2.8 (also in [66]), and includes modifications to enable check-merging. The modifications require some logic overhead, but the benefit is a significantly increased throughput of the decoder on punctured blocks. The increased throughput comes from three sources:

- faster propagation of messages due to merged check nodes, leading to a decrease in the average number of iterations,
- a reduction in the code complexity due to removed nodes, reducing the number of clock cycles for each iteration, and
- the elimination of idle waiting cycles between layers.

4.3.1 Decoder schedule

The schedule of the decoder and relevant definitions are described in Sec. 2.8.1.

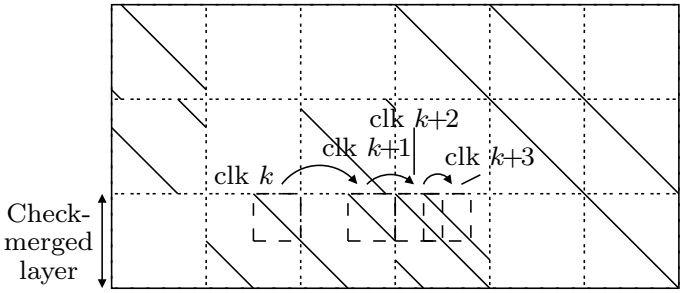


Figure 4.5 *Schedule of decoder on check node merged parity-check matrix.*

Figure 4.5 shows the resulting parity-check matrix when the bits in the right-most sub-matrix in Fig. 2.25 have been punctured and the last two layers merged. As can be seen, the merged layer consists of the sums of the corresponding rows in the original parity-check matrix. The result is a parity-check matrix that may contain sub-matrices that have column weights larger than one. The bit-sum blocks in a check-node group may then overlap, which the architecture has to be aware of. Whereas the original architecture computes updated bit-sums directly, the proposed architecture computes bit-sum differences which are then added to the old bit-sums to produce the updated ones. The proposed architecture thus efficiently handles the case where a variable node is involved simultaneously in several check nodes.

As the check nodes in a check node group are processed in parallel, check node merging can only be used if all the check nodes in the check node group are purged. This is not a major limitation, however, as puncturing sequences are normally defined on the base matrix of the code and then expanded to the used block length. Thus, at most one check node group needs to have both purged and non-purged check nodes, and this check node group is then processed by initializing the punctured variable nodes to zero.

4.3.2 Architecture overview

An overview of the check node merging decoder architecture is shown in Fig. 4.6. At the start of each block, the contents of the bit-sum memories are initialized with the log-likelihood ratios received from the channel. Then the contents of the bit-sum memories are updated during a number of iterations. In each iteration, the hard-decision values are also written to the decision memory, from where they can be read when decoding of the block is finished.

In each iteration, bit-sums are read from the bit-sum memories, routed by a cyclic shifter to the correct check function unit (CFU), and then routed back. The output of the CFUs are bit-sum differences which are accumulated for each read bit-sum value in the bit-sum update block. The updated bit-sums are then

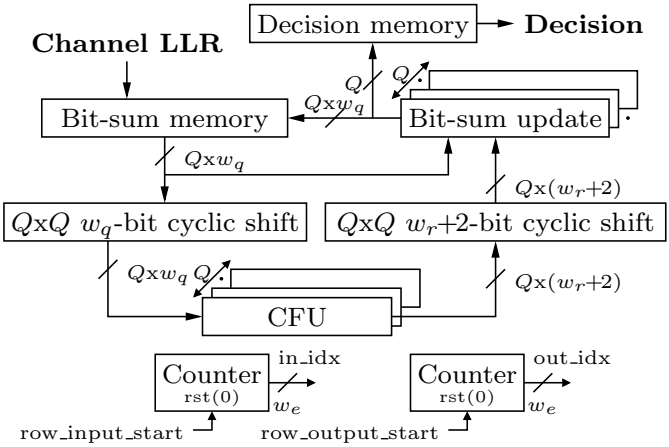


Figure 4.6 *Architecture of decoder utilizing check merging.*

rewritten to the bit-sum memories. The two counters count the current input and output indices of the bit-sum blocks of the currently processed check node group.

The following parameters are defined for the architecture:

- Z is the maximum expansion factor
- Q is the parallelization factor
- C is the maximum check node degree
- w_q is the bit-sum wordlength
- w_r is the wordlength of the min-sum magnitudes
- w_e is the edge index wordlength

4.3.3 Cyclic shifters

As in the original architecture [66], the shifters are implemented as barrel shifters with programmable wrap-around. However, unlike the original architecture the return path needs a cyclic shift as a bit-sum may need updates produced in several different CFUs.

4.3.4 Check function unit

In the CFU, shown in Fig. 4.7, the old check-to-variable messages are stored for each of the check nodes' neighbors. As the min-sum algorithm is used, only two values are needed. These are stored as w_r-bit magnitudes along with a w_e-bit index of the node with the minimum message and one bit containing the check node parity. These bits are grouped together and stored in the min-sum memory.

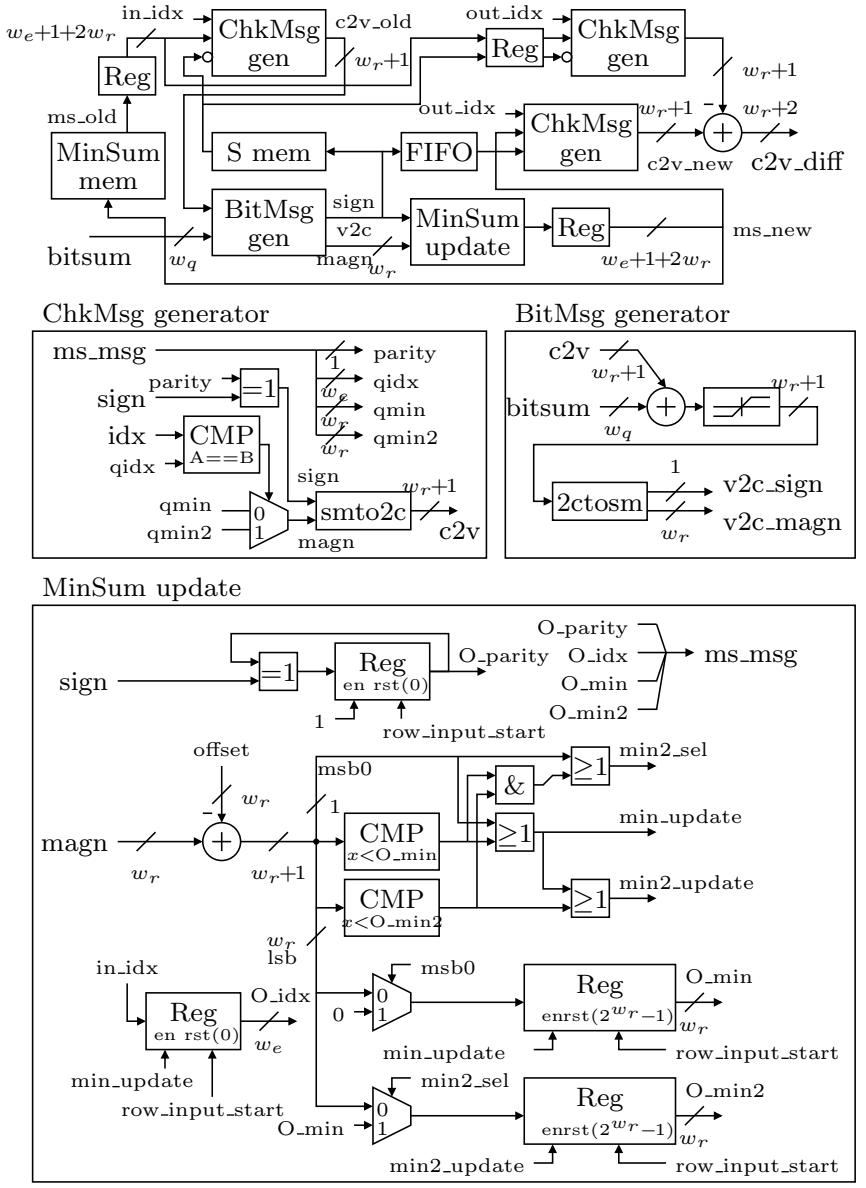


Figure 4.7 *Check function unit.*

Along with the signs of each check-to-variable message stored in the S memory, the check message generator generates the old check-to-variable message in two's complement $w_r + 1$-bit representation for each variable node neighbor. The old check-to-variable messages are subtracted from the bit-sum input to generate the

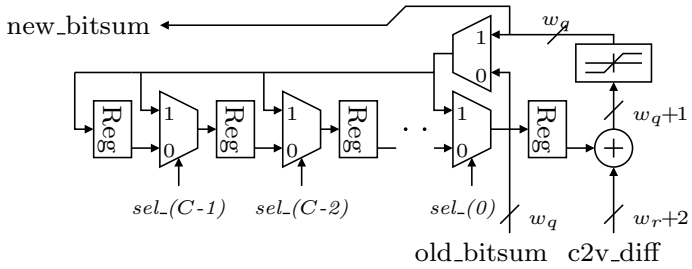


Figure 4.8 *Bit-sum update unit.*

variable-to-check messages in the bit message generator. These are used by the min-sum update unit to compute the new min-sum values, which are subsequently stored in the min-sum memory. They are also sent to the second check message generator, which generates the new check-to-variable messages. The difference between the old and new check-to-variable message is computed, and forms the output of the CFU.

The check message generator consists of a mux to choose the correct check-to-variable message and a converter from signed magnitude to two's complement. The bit message generator computes the variable-to-check message by adding the inverted old check-to-variable message to the bit-sum and then converting the result back to signed magnitude representation. The min-sum update unit is unchanged from the original in [66], and performs a basic search for the two minimum magnitudes in its input. A programmable offset value is used to correct for the overly optimistic estimates of the min-sum algorithm. Also, the parity of the check node is computed.

4.3.5 Bit-sum update unit

In Fig. 4.8, the bit-sum update unit is shown. When processing of a check node group starts, the bit-sums are loaded in the shift registers with a delay equal to the current node degree, and thus equal to the delay of the CFUs. Then, when the bit-sum differences from the CFUs start to arrive, they are added to the old bit-sums. The sum is truncated and sent to the output to be written back to the bit-sum memories. When several CFUs have updates to the same bit-sum, the updated bit-sum is reloaded in the shift register to replace the obsolete value.

4.3.6 Memories

The bit-sums are organized into Q memory banks with data wordlengths of w_q bits. Each memory l contains the bit-sums of the variable nodes v_k where $k = l \bmod Q$. The min-sums are organized into Q memory banks with data wordlengths of $w_e + 1 + 2w_r$ bits, which holds a w_e-bit edge index, 1-bit parity, and two w_r-bit

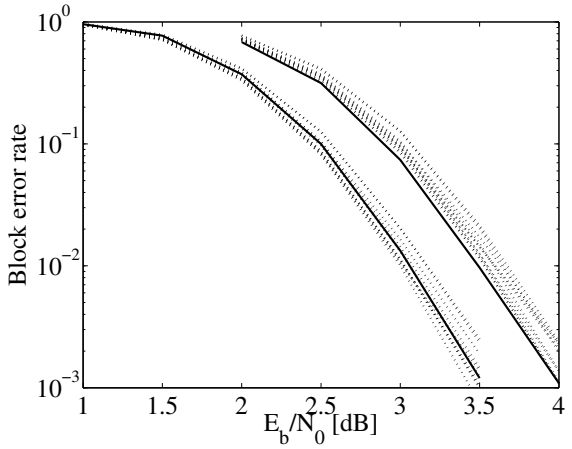


Figure 4.9 *Comparison of puncturing sequences for the WiMax rate-1/2 code punctured to rate 2/3 and 3/4. The straight lines are the optimized sequences and the dotted lines are obtained using the heuristic in [46].*

magnitudes. Each memory l contains the min-sums of the check nodes c_k where $k = l \bmod Q$. [1]

4.4 Simulation results

4.4.1 Design of puncturing sequences

The optimization algorithm in Sec. 4.1 has been applied to the WiMax rate-1/2 base matrix ($(N, M) = (24, 12)$) to create a base puncturing pattern of length 11, with which code rates up to 0.92 can be obtained. The base puncturing pattern was expanded with factors of 24, 36 and 48 to create puncturing patterns for block lengths of 576, 864 and 1152. To solve the optimization problem, the SCIP solver [2] has been used. Solving the problem for the WiMax rate-1/2 base matrix was done in seconds on a desktop computer, and resulted in the puncturing sequence

$$p = (13, 16, 19, 22, 14, 17, 20, 23, 6, 15, 8)\,. \tag{4.25}$$

The performance of the codes was evaluated using the sum-product decoding algorithm with a maximum of 100 iterations. For all of the measurements a minimum of 100 failed decoding attempts were observed.

In Fig. 4.9, puncturing sequences for the $(N, K) = (576, 288)$ code punctured to rate 2/3 and 3/4 have been compared. The figure shows the ILP-optimized

[1] For standard codes and parallelization degrees, these memory arrangements may result in sparingly used memories. To circumvent this, the same techniques as in [66] may be used.

1-SR	4	5	6	7
Number	130	372	385	113

Table 4.1 *Number of 1-SR nodes for 1000 runs of the algorithm [46] applied to the WiMax rate-1/2 base matrix. The proposed algorithm achieves 8 1-SR nodes.*

Rate	0.75	0.8	0.86	0.92
[46]	2	2–3	2–4	6
Proposed	1	2	2	4

Table 4.2 *Recovery delay (maximum k-SR node) for different punctured rates of the WiMax rate-1/2 base matrix.*

sequence together with an ensemble of ten random sequences obtained using the greedy algorithm in [46]. In order to provide a fair comparison with scalable block sizes, the greedy algorithm was applied on the base matrix and the obtained base puncturing patterns were expanded with the matrix expansion factor. For rate 2/3, the performance of the optimized sequence is comparable to those obtained with the heuristics, whereas for rate 3/4, the performance of the optimized sequence is superior to those obtained with the heuristics. There is a small trade-off between performance at different rates, which partly explains why the optimized sequence is not superior for all rates. However, it is also the case that the algorithm optimizes recovery time, which may not necessarily result in optimal performance.

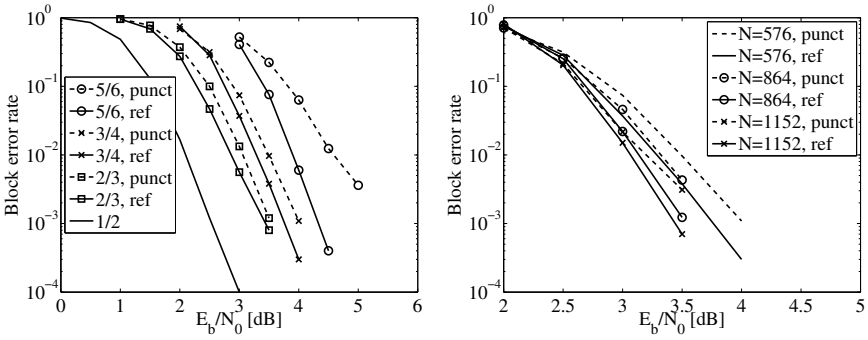
In Table 4.1, the number of 1-SR nodes obtained with the algorithm in [46] is shown to vary between 4 and 7. In contrast, the optimization achieves 8 1-SR nodes. In addition, as seen in Table 4.2 the recovery delay is commonly higher for the greedy algorithm, leading to shorter decoding times for the optimized puncturing sequences.

In Fig. 4.10(a), the rate-1/2 length-576 WiMax code punctured to rates 2/3, 3/4, and 5/6 have been compared to the dedicated codes of the standard. It is seen that the performance difference is small for the lower rates.

In Fig. 4.10(b), the performance loss of puncturing the rate-1/2 WiMax code to rate 3/4 is shown for block sizes of 576, 864 and 1152. The puncturing patterns are obtained from the same base puncturing pattern, showing the possibility of using the proposed scheme with scalable block sizes.

4.4.2 Check-merging decoding algorithm

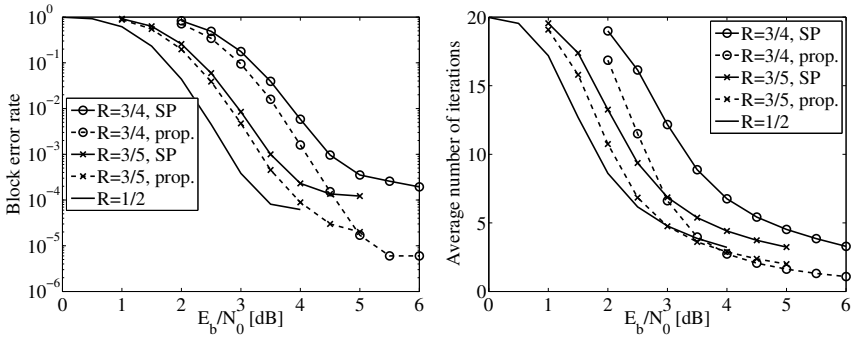
The proposed check-merging decoding algorithm in Sec. 4.2 has been applied to the WiMAX rate-1/2 code punctured to rates 3/5 and 3/4. The resulting codes have been simulated over an AWGN channel and decoded with the sum-product algorithm on both the rate-1/2 graph and the check-merged graph. The block error



(a) Comparison of the punctured rate-1/2 length-576 WiMax code with dedicated rates of the standard.

(b) Performance of the rate-1/2 WiMax code punctured to rate 3/4 for block sizes of 576, 864 and 1152.

Figure 4.10 *Performance of ILP-based puncturing scheme.*



(a) Block error rate of WiMAX rate-1/2 code punctured to rates 3/5 and 3/4.

(b) Average number of iterations for WiMAX rate-1/2 code punctured to rates 3/5 and 3/4.

Figure 4.11 *Simulation results of check-merging decoding algorithm.*

rate and average number of iterations have been computed. Simulations using a maximum of 20 iterations are shown in Figs. 4.11(a) and 4.11(b), respectively. The main benefit is the significant reduction of the average number of iterations, allowing an increase in the throughput.

4.5 Synthesis results of check-merging decoder

The proposed check-merging architecture in Sec. 4.3 has been synthesized for the LDPC codes used in IEEE 802.16e and IEEE 802.11n in order to determine the logic overhead required to support check merged decoding. For IEEE 802.16e, a parallelization factor of $Q = 24$ has been used. For IEEE 802.11n, the maximum

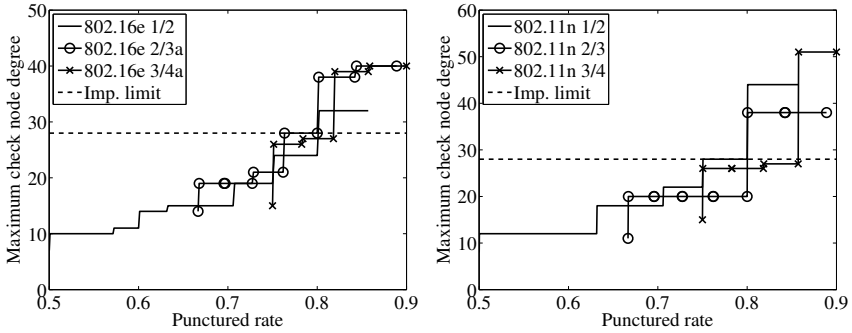


Figure 4.12 *Maximum check node degrees of check node merged matrices for IEEE 802.16e codes and IEEE 802.11n codes.*

length codes of $N = 1944$ have been assumed, and in order to more accurately conform to the data rate requirements a larger parallelization factor of $Q = 81$ has been used. For all implementations, $w_q = 8$, $w_r = 5$ and $w_e = 5$.

4.5.1 Maximum check node degrees

The maximum check node degree supported by the decoder is determined by the length of the register chain in the bit-sum update unit. Therefore a limit must be set based on the degree distributions of the check node merged matrices. Figure 4.12 shows the maximum check node degrees as a function of the punctured rate for the considered codes of the IEEE 802.16e and IEEE 802.11n standards. In both cases, $C = 28$ has been considered a suitable trade-off between decoder complexity and achievable rates.

4.5.2 Decoding throughput

As the complexity of the code is reduced when checks are merged, the attainable throughput increases as the punctured rate increases. This gain is in addition to the faster convergence rate of the check merging decoder. Assuming a fixed number of 10 iterations, the minimum achievable throughputs for IEEE 802.16e and IEEE 802.11n are shown in Fig. 4.13.

4.5.3 FPGA synthesis

The reference decoder (Sec. 2.8) and the proposed check merging decoder have been synthesized for an Altera Cyclone II EP2C70 FPGA with speedgrade -6. Mentor Precision was used for the synthesis, and the results are shown in Tables 4.3 and 4.4 for IEEE 802.16e and IEEE 802.11n, respectively. The C values for the reference decoders were set to the maximum check node degrees of the fixed rate codes in the

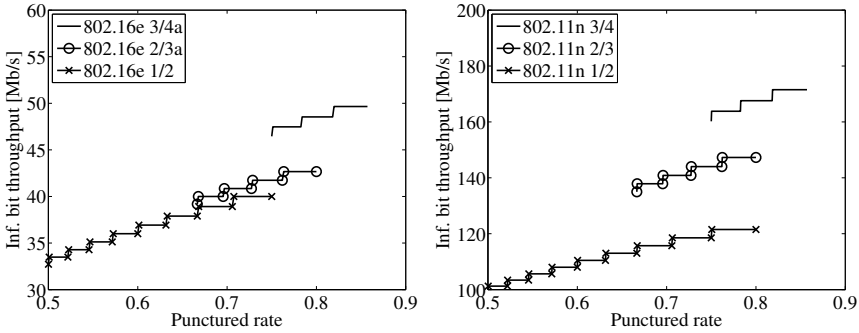


Figure 4.13 *Information bit throughput of decoder on IEEE 802.16e codes and IEEE 802.11n codes.*

Table 4.3 *Synthesis results of decoders for IEEE 802.16e codes.*

	Q	C	LUTs	Regs	Mem. bits
Fixed rate (Sec. 2.8	24	20	4750	6303	20640
Rate-compatible (Sec. 4.3)	24	28	5988	8015	20480

Table 4.4 *Synthesis results of decoders for IEEE 802.11n codes.*

	Q	C	LUTs	Regs	Mem. bits
Fixed rate (Sec. 2.8)	81	23	16396	21865	66560
Rate-compatible (Sec. 4.3)	81	28	22884	28635	66336

respective standards. For 802.16e, the overhead of the check merging decoder is 26% and 27% for the LUTs and registers, respectively. For 802.11n, the overhead is bigger at 40% and 31% for the LUTs and registers, respectively. A significant number of the LUTs is consumed in the barrel shifters.

5

Data representations

In this chapter, the representation of the data in a fixed-point decoder implementation is discussed. It is shown 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.

This chapter has been published in [7].

5.1 Fixed wordlength

The sum-product algorithm, as formulated in the log-likelihood domain in (2.13)–(2.15), lends itself well towards fixed wordlength implementations. Usually, decent performance is achieved using wordlengths as short as 4–6 bits [111]. 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.

5.2 Data compression

The function $\Phi(x)$ is shown in Fig. 5.1. In a hardware implementation, usually the output of this function is stored in memories between the phases, and it is therefore of interest to represent the information efficiently. However, as the function is non-

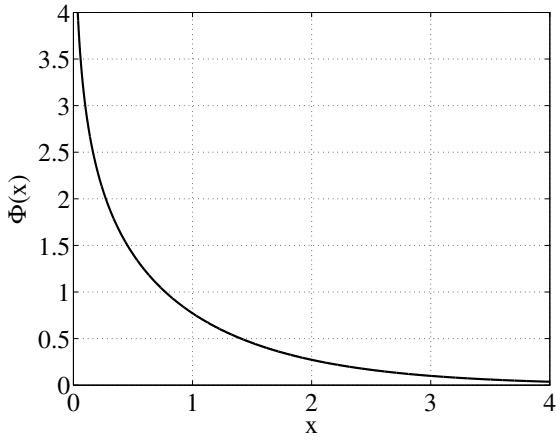
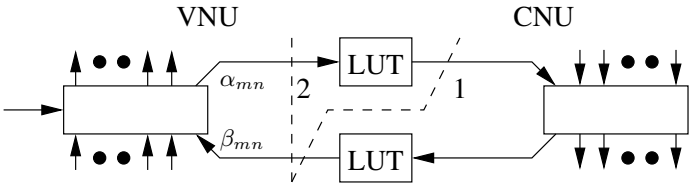
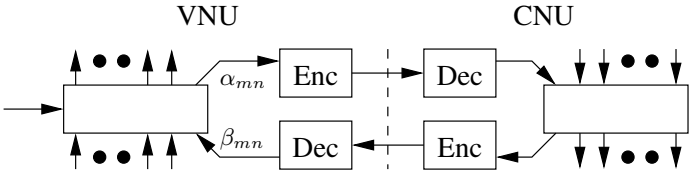


Figure 5.1 *The domain transfer function $\Phi(x)$.*



(a) Conventional data-path.

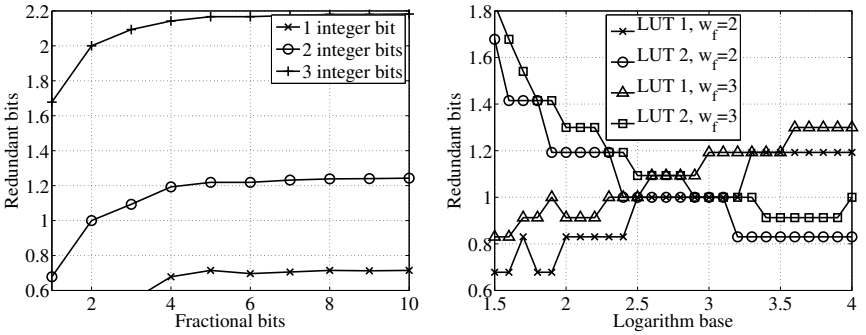


(b) Data-path utilizing compression.

Figure 5.2 *Data-paths in a sum-product decoder.*

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. 5.2(a). The VNU block performs the computation of the α_{nm} messages in (2.13), whereas the CNU block performs the additions for the β_{mn} messages in (2.14). $\Phi(x)$ 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, e.g., in [109] and [68], respectively. Instead of these partitions, the implementation in Fig. 5.2(b) is suggested, where an encoder is used to convert the messages to a more compact representation which is used to communicate the



(a) Redundant bits $B(w_i, w_f)$ of LUT value entries for different data representations.

(b) Redundant bits of LUT value entries for different logarithm bases using data representations with $w_i = 2$, $w_f = 2$, and $w_i = 2$, $w_f = 3$.

Figure 5.3 *Redundant bits of discretized domain transfer function* $\Phi(x)$.

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 output of the LUT blocks can be written

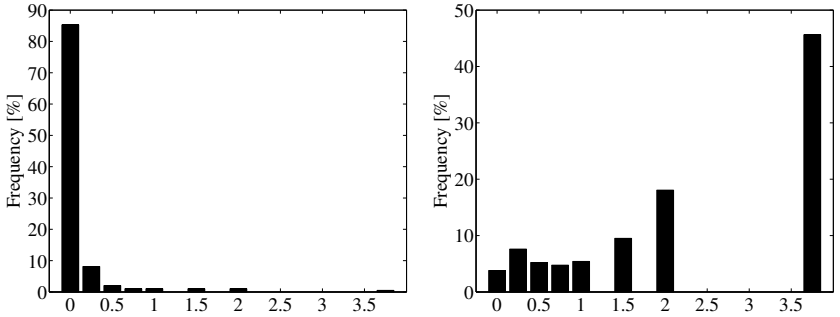
$$B(w_i, w_f) = \log_2 \left(2^{w_i + w_f} - N_o\right), \tag{5.1}$$

where N_o is the number of unique function values of the discretized function of $\Phi(x)$.

5.3 Results

$B(w_i, w_f)$ is plotted for different parameters in Fig. 5.3(a). Obviously $B(w_i, w_f)$ also depends on the base of the logarithm in $\Phi(x)$ described in Sec. 2.6.5. 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 [111] 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. 5.3(b). 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.

Assuming that the natural logarithm base and a data representation with $w_i = 2$ and $w_f = 2$ is used, messages consist of 6 bits (including sign bit and hard decision/parity-check bit), and compression thus results in a wordlength reduction of 16.7%. The area overhead associated by the separation of the look-up



(a) Distribution of input values to CNU. (b) Distribution of input values to VNU.

Figure 5.4 *Distributions of arguments to discretized domain transfer function* $\Phi(x)$.

table has been estimated by synthesis to standard cells in a $0.35\mu m$ CMOS process. The synthesis was performed using Synopsys design compiler. The synthesis of the original look-up table required 12 cells utilizing a total area of $726\mu m^2$. Realized as separate blocks using a straightforward mapping of data to the intermediate compressed format, the encoder and decoder parts utilized 8 cells occupying $528\mu m^2$ and 6 cells occupying $309\mu m^2$, respectively. As comparison, a 6-input CNU and a 3-input VNU occupy roughly $39000\mu m^2$ and $31000\mu m^2$ respectively, when synthesized using the same methods. Thus the area overhead of $111\mu m^2$ per look-up table amounts to a 1–1.5% area increase for the process elements. In contrast, problems with large routing overheads are commonly reported in implementations, with overheads as large as 100% in the fully parallel $(N, K) = (1024, 512)$ code decoder in [18].

Additionally to reducing the wordlength of messages, the separation of the look-up table also allows a possibility to choose a representation suitable for energy-efficient communication, and, as shown above, the separation is essentially free. Consider the distribution of look-up table values for CNU input (Fig. 5.4(a)) and VNU input (Fig. 5.4(b)) obtained using an $(N, K) = (1152, 576)$ code. It is obvious that the energy dissipation for communication of messages will depend on the encoding chosen. For example, in a parallel architecture the most common values can be assigned representations with a small mutual distance, whereas in a partly parallel or serial architecture an asymmetric memory [22] might be efficiently used if low-weight representations are chosen for the most common values.

6

Conclusions and future work

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

6.1 Conclusions

In part I of this thesis, two improved algorithms for the decoding of LDPC codes were proposed. Chapter 3 concerns the early decision algorithm, which reduces the computational complexity by deciding certain parts of the codeword early during decoding. 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. 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}. In general, larger reductions are obtainable for codes with higher rates and smaller block sizes.

In Chapter 4, the check merging decoding algorithm for rate-compatible LDPC codes was suggested. The algorithm proposes a technique of merging check nodes of punctured LDPC codes, thereby reducing the decoding complexity. For dual-diagonal LDPC codes, the algorithm offers a reduced decoding complexity due to two factors: the complexity of each iteration is reduced due to the removal of nodes in the code's Tanner graph, and the number of iterations required for convergence

is reduced due to faster propagation of messages in the graph. For the punctured rate-1/2 length-576 IEEE 802.16e code, punctured to rate-3/4, the check merging algorithm offers a more than 60% reduction in the average number of iterations at a block error rate of 10^{-3} over an AWGN channel.

The early decision and check merging algorithm have both been implemented on FPGAs. In both cases, the proposed algorithms come with a logic overhead compared with their respective reference implementations. The early decision algorithm was implemented for a $(3, 6)$-regular code class, whereas the check merging algorithm was implemented for the QC-LDPC codes with dual-diagonal structure used in the IEEE 802.16e and IEEE 802.11n standards. The logic overhead of the early decision algorithm was relatively big at 45%, whereas the logic overhead of the check merging algorithm was 26% and 40% for the IEEE 802.16e and IEEE 802.11n implementations, respectively.

6.2 Future work

The following ideas are identified as possibilities for future work:

- 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.
- The behavior of the early decision algorithm using irregular codes and efficient choices of thresholds should be analyzed.
- The check merging algorithm works well when the punctured nodes are of degree two, but investing the possibilities of adapting the algorithm to work with punctured nodes of higher degrees is of interest.
- For both of the proposed algorithms, their performance over fading channels is of interest.

Part II

High-speed analog-to-digital conversion

7

Introduction

7.1 Background

The analog-to-digital converter (ADC) is the interface between the analog signal domain and the digital processing domain, and is used in all digital communications systems. Normally, an ADC consists of a sample-and-hold stage and a quantization stage, where the sample-and-hold performs discretization in time and the quantization stage performs discretization of the signal level. There are many different types of ADCs, and some of the more common ones are:

- the *flash ADC*, which consists of a number of comparators with different reference signal levels, followed by a thermometer-to-binary decoder. The flash ADC is fast, but achieves low resolution as the number of comparators grows exponentially with the required number of bits.
- the *successive approximation ADC*, which uses an internal digital-to-analog converter (DAC) fed on a register containing the current approximation of the analog input. In each clock cycle, a comparator compares the input to the current approximation, deciding one bit of the output. The successive approximation ADC is slower than the flash ADC, but can achieve medium resolutions.
- the *pipeline ADC*, which consists of several cascaded flash ADCs. In each

stage, the signal is discretized by a course flash ADC, and the approximation is then subtracted from the signal and the difference is passed on to the following stages. The pipeline ADC thus combines the speed of the flash ADC with the resolution of the successive approximation ADC.

- the *sigma-delta ADC* ($\Sigma\Delta$-ADC), which is essentially a low-resolution flash ADC with a negative feed-back loop for the quantization error. The feedback allows the quantization noise to be frequency shaped, which is combined with oversampling to place the majority of the quantization noise in a frequency band unoccupied by the signal. This requires the ADC to be followed by a filter to remove the quantization noise and a downsampling stage to reduce the sampling frequency to a useful rate. Due to the oversampling, the achievable speed of a $\Sigma\Delta$-ADC is limited, but the architecture is normally resistant to analog errors, allowing high resolutions to be obtained.

The resistance to analog errors of the $\Sigma\Delta$-ADC is attractive from a system-on-chip point of view, as digital circuits are almost exclusively made in cost-efficient CMOS processes with small feature sizes, in which linear analog devices are difficult to design. $\Sigma\Delta$-ADCs allow a relaxation on the requirements of the analog devices, and instead compensating for errors with digital signal processing for which the CMOS processes are well suited. Therefore, attemps have recently been made to build high-speed $\Sigma\Delta$-ADCs [19, 82, 103] for general communications standards.

The work in this part of the thesis considers the design of high-speed ADCs through parallel $\Sigma\Delta$-modulators, and the design of high-speed FIR filters with short wordlengths which are usable as decimation filters for high-speed ADCs. The proposed parallel $\Sigma\Delta$-modulator designs use modulation of the input signal to reduce matching requirements of the analog devices, and a general method to analyze the matching requirements of such systems is presented. The targeted data converters have bandwidths of tens of MHz and sampling frequencies in the GHz range, making both the linearity of analog devices and the implementation of digital decimation filters difficult. The proposed design of high-speed FIR filters uses bit-level optimization on the placements of full and half adders in the implementation.

7.2 Applications

There are many applications that could benefit from system-on-chip integration of transceiver frontends, but the main target is area- and power-constrained devices such as handhelds and laptops. Such devices support an increasing number of communication standards such as GSM and UMTS for voice and data communications, wireless LAN for high-speed internet access, bluetooth for peripheral connections and GPS for positioning services. These standards have wildly differing requirements on bandwidth and resolution, making the $\Sigma\Delta$-ADC attractive as these measures can be traded by adjusting the width of the digital decimation filter.

7.3 Scientific contributions

There are two main contributions in this part of the thesis. In Chapter 10, a method of analyzing general parallel ADCs for matching requirements is presented. This is done by a multi-rate formulation of the parallel system, and the stability of subsets of the channels can then be determined by observing the transfer function matrix for the sub-system. Time-interleaved, Hadamard-modulated and frequency-band decomposed parallel ADCs are then described as special cases of the general system, and their resistances to analog imperfections are analyzed. The multi-rate formulation is then used as a base to find insensitive systems.

The second main contribution is in Chapter 12 and considers the design of high-speed FIR filters that may be used as decimation filters for high-speed $\Sigma\Delta$-ADCs. The problem of designing an efficient filter is decomposed in two parts, where the first part considers generation of a set of weighted bit-products and the second part considers the construction of a pipelined summation tree. In the first part, the filter is first formulated as a multi-rate system. Then, the branch-filters are implemented using one of several structures, including direct form FIR and transposed direct form FIR, and bit-products are generated for each of the coefficients. Finally, bit-products with the same bit weight are merged. In the second part, the summation of the generated bit-products is considered. Traditional methods including Wallace trees and Dadda trees are used as references, and a bit-level optimization algorithm is proposed that minimizes a cost function based on the number of full adders, half adders and pipeline registers. Finally, the suitability of the different structures are evaluated for decimation and interpolation filters of varying wordlengths, rate change factors and pipeline depths.

8

FIR filters

In this chapter, the basics of finite impulse response (FIR) filters are discussed. The definition of a filter from an impulse response is in Sec. 8.1, and the z-transform is introduced as a tool to analyze the behavior of a filter. In Sec. 8.2, some design methods for FIR filters are touched briefly. In Sec. 8.3, the basics of sampling rate conversion and multirate signal processing are discussed, and in Sec. 8.4, the main architectures for the realization of FIR filters are introduced. Also, a bit-level realization of an FIR filter for decimation is shown, and the realization uses multirate theory to reduce the arithmetic complexity. More in-depth introductions to digital filters are available in [78], and multirate theory is extensively discussed in [94].

8.1 FIR filter basics

A digital filter is a discrete linear time-invariant system that is typically used to amplify or suppress certain frequencies of a signal. Digital filters can be partitioned into two main classes: finite impulse response (FIR) filters and infinite impulse response (IIR) filters. FIR filters often result in more complex implementations than IIR filters, for a given filter specification. However, FIR filters have other advantages such as better phase characteristics, better stability, fewer finite-wordlength considerations and better pipeline abilities. The work in this thesis considers FIR

filters only.

8.1.1 FIR filter definition

An FIR filter [78] can be characterized by its impulse response $h(k)$, a scalar discrete function of k. For a causal FIR filter of order N, $h(k) = 0$ when $k < 0$ or $k > N$, and the impulse response thus has at most $N+1$ nonzero values. The impulse response defines factors with which different delays of the input should be multiplied, and the output is the sum of these. Thus, denoting the input and output by $x(n)$ and $y(n)$, respectively, the behavior of an Nth order filter with impulse response $h(k)$ is defined by

$$y(n) = \sum_{k=0}^{N} x(n-k)h(k). \tag{8.1}$$

It can be seen that when the input is the Kronecker delta function ($x(n) = \delta(n)$), the output is the impulse response ($y(n) = h(n)$).

8.1.2 z-transform

The impulse response describes the filter's behavior in the time domain, but does not say much about a filter's behavior in the frequency domain. Thus, in order to analyze a filter's frequency characteristics, the z-transform $H(z)$ of the impulse response $h(k)$ is defined as

$$H(z) = \sum_{k=-\infty}^{\infty} h(k)z^{-k}, \tag{8.2}$$

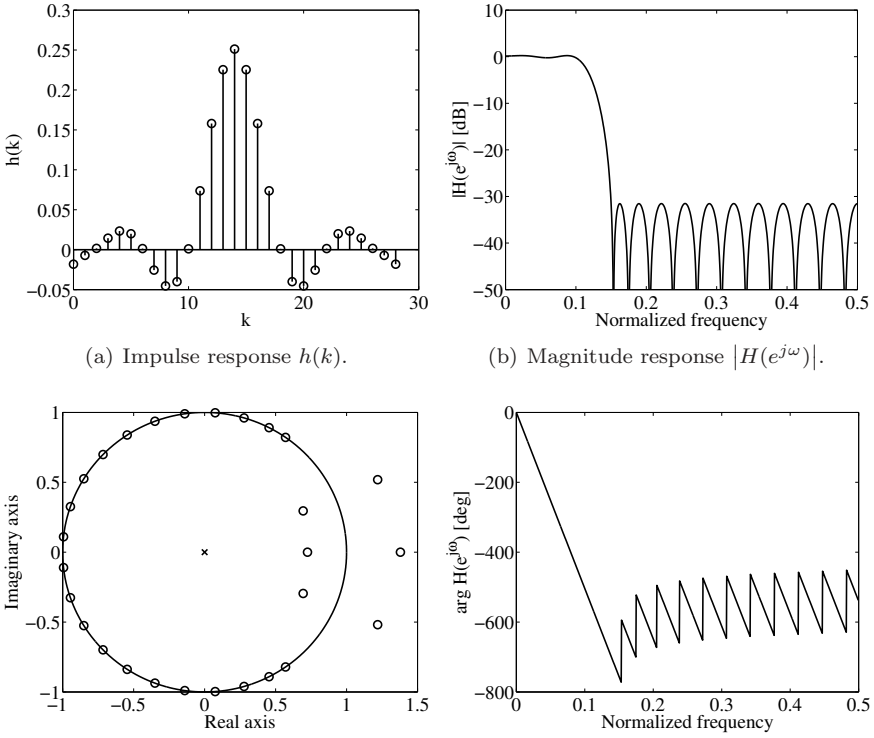
and the input and output are related by the equation

$$Y(z) = H(z)X(z). \tag{8.3}$$

The frequency response of the filter is then $H(e^{j\omega})$ which is common to visualize as the magnitude response $\left|H(e^{j\omega})\right|$ and the phase response $\arg H(e^{j\omega})$. It is also possible to visualize the filter's frequency characteristics by a pool-zero diagram. This is done by rewriting $H(z)$ as

$$H(z) = \sum_{k=0}^{N} h(k)z^{-k} = \frac{\sum_{k=0}^{N} h(k)z^{N-k}}{z^N}, \tag{8.4}$$

where the zeros and poles are the zeros of the numerator and the denominator, respectively. Since the denominator is a single power of z, all the poles are in the origin, making the filter unconditionally stable.



(a) Impulse response $h(k)$.

(b) Magnitude response $\left|H(e^{j\omega})\right|$.

(c) Pole-zero diagram.

(d) Phase response $\arg H(e^{j\omega})$.

Figure 8.1 *28th order linear phase FIR filter with impulse response, frequency response and pole-zero diagram.*

8.1.3 Linear phase filters

One of the advantages of FIR filters is the ability to design filters with linear phase. Linear phase filters ensure that the delay is the same for all frequencies, which is often desirable in data communications applications. A linear phase filter results when the impulse response coefficients are either symmetric or anti-symmetric around the middle, and are commonly designed using the MPR algorithm (see Sec. 8.2).

Consider the example design of a 28th order linear phase lowpass FIR filter shown in Fig. 8.1. The figure shows the impulse response $h(k)$, the magnitude response $\left|H(e^{j\omega})\right|$, the pole-zero diagram of $H(z)$ and the phase response $\arg H(e^{j\omega})$. The linear phase property can be seen in both the coefficient symmetry in the impulse response in Fig. 8.1(a) and in the phase response in Fig. 8.1(d).

(a) L-fold upsampler. (b) L-fold downsampler.

Figure 8.2 *Rate change elements.*

8.2 FIR filter design

There are many ways to design an FIR filter. In this section, some of the more commonly used when optimizing for a desired magnitude are given. A desired real non-negative magnitude response function $H_d(e^{j\omega})$ is given along with a weighting function $W(e^{j\omega})$. The optimization process then tries to design the filter $H(e^{j\omega})$, as defined by (8.2), to match $H_d(e^{j\omega})$ as closely as possible, with allowed relative errors for different frequencies defined by the weighting function $W(e^{j\omega})$. Using a mini-max goal function, the maximum weighted error over the frequency band is minimized for a given filter order, i.e.,

$$\text{minimize} \max_{\omega\in[0,2\pi]} W(e^{j\omega}) \left| \left|H(e^{j\omega})\right| - H_d(e^{j\omega})\right|. \tag{8.5}$$

In certain cases, it is more beneficial to minimize the error energy rather than the maximum error. This leads to a least-squares optimization goal, which can be defined as

$$\text{minimize} \int_0^{2\pi} \left(W(e^{j\omega}) \left| \left|H(e^{j\omega})\right| - H_d(e^{j\omega})\right|\right)^2. \tag{8.6}$$

In this thesis, the McClellan-Parks-Rabiner (MPR) algorithm [84, 85] has been used for the design of optimal mini-max FIR filters.

8.3 Multirate signal processing

8.3.1 Sampling rate conversion

In a system utilizing multiple sampling rates, rate changes are modelled by the upsampler and downsampler elements, shown in Fig. 8.2. The L-fold upsampler inserts zeros in the input stream, such that

$$y(n) = \begin{cases} x\left(\frac{n}{L}\right) & \text{for } n = 0, \pm L, \pm 2L, \dots \\ 0 & \text{otherwise.} \end{cases} \tag{8.7}$$

In the z-domain, (8.7) becomes

$$Y(z) = X(z^L), \tag{8.8}$$

and the spectrum of $y(n)$ thus consists of L copies of the spectrum of $x(n)$.

(a) L-fold interpolation. (b) L-fold decimation.

Figure 8.3 *Conventional interpolation and decimation of a signal.*

The L-fold downsampler retains every Lth sample of its input and discards the others, such that

$$y(m) = x(mL). \tag{8.9}$$

In the z-domain, (8.9) becomes

$$Y(z) = \frac{1}{L} \sum_{l=0}^{L-1} X\left(e^{-j\frac{2\pi l}{L}} z^{\frac{1}{L}}\right), \tag{8.10}$$

and the output spectrum thus consists of L stretched and shifted versions of the input spectrum.

It can be seen that both the upsampler and downsampler affects the spectrum of the input: the output of the upsampler contains replicas of the input spectrum, whereas the output of the downsampler contains aliasing. Thus, when the upsampler and downsampler are used to interpolate or decimate a signal, the signal must also be filtered to remove the replicas (for the upsampler) or to remove the aliasing (for the downsampler). Typically, the filter is a lowpass filter, but can also be a highpass filter or any bandpass filter with a bandwidth of at most $2\pi/L$. For the interpolator, filtering is performed on the output as shown in Fig. 8.3(a), whereas it is performed on the input for the decimator as shown in Fig. 8.3(b).

For both the interpolator and the decimator, it can be seen that the filtering is performed at the higher sampling rate. It can also be noted that the filtering is unnecessarily complex in both cases. For the interpolation filter it is known that $L-1$ of L input samples are zero and do not contribute to the output. For the decimation filter, $L-1$ of L output samples are computed, but then discarded by the downsampler. In Sec. 8.3.3, it is shown how this knowledge can be used to reduce the complexities of the filters.

8.3.2 Polyphase decomposition

For a filter $h(k)$ with z-transform $H(z)$, (8.2) can be rewritten

$$H(z) = \sum_{k=-\infty}^{\infty} h(k) z^{-k} = \sum_{l=0}^{L-1} z^{-l} \sum_{k=-\infty}^{\infty} h(Lk+l) z^{-Lk} = \sum_{l=0}^{L-1} z^{-l} H_l(z^L), \tag{8.11}$$

where $H_l(z)$ is the z-transform of the lth polyphase component

$$h_l(k) = h(Lk+l), \quad l = 0, 1, \ldots, L-1. \tag{8.12}$$

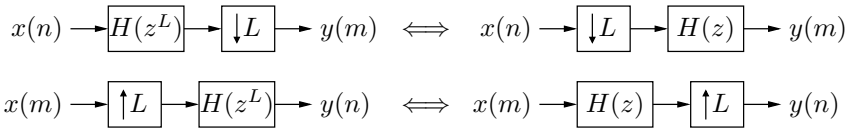


Figure 8.4 *Noble identities.*

This decomposition is usually called the Type-1 polyphase decomposition [94], and is often used to parallelize the implementations of FIR filters [78]. It is also normally used in the implementations of FIR rate change filters, as it allows the implementation to retain the complexity, but run at a lower sampling rate.

8.3.3 Multirate sampling rate conversion

The upsampler and downsampler are time-varying operations [94], and therefore operations can in general not be moved through them. However, the Noble identities [94] allow filters and rate change operations to exchange places in certain conditions. The Noble identities are shown in Fig. 8.4.

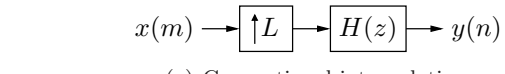
Using polyphase decomposition and the Noble identities, the complexities of the interpolator and decimator in Fig. 8.3 can be reduced significantly. The transformation of the interpolator is shown in Fig. 8.5, where the interpolation filter $H(z)$ is first polyphase decomposed with a factor of L. The resulting subfilters $H_l(z)$ have the same total complexity as the original filter $H(z)$. Then the upsampler is moved to after each subfilter, reducing the sampling frequencies of the filters by a factor of L. Finally, analyzing the structure at the output it is realized that only one branch is non-zero at a given time and can therefore be realized as a commutator between the branches. The transformation of the decimator is shown in Fig. 8.6 and is analogous.

8.4 FIR filter architectures

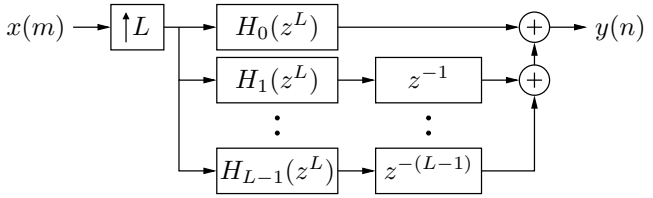
8.4.1 Conventional FIR filter architectures

The most straight-forward implementation of an FIR filter is the direct form architecture, which is a direct realization of the filter definition (8.1). The architecture is shown in Fig. 8.7, and has a delay line for the input signal $x(n)$, forming a number of delayed input signals $x(n-k)$. The delay line is tapped at a number of positions, where the delayed input is scaled by the appropriate impulse response coefficient, and all products are then summed together and form the output $y(n)$.

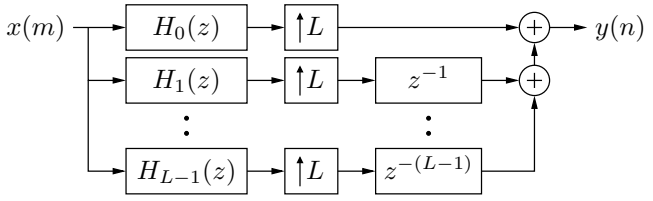
The transposed direct form structure can be obtained from the direct form structure using the transpose operation [78], which for systems with a single input and output results in an arithmetically equivalent system. The transposed direct form architecture is shown in Fig. 8.8. The transposed direct form may lend itself better to high-speed implementations, as the critical path of the adders at



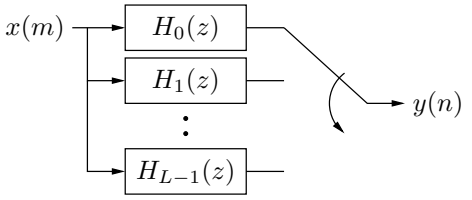
(a) Conventional interpolation.



(b) Polyphase decomposition of filter.



(c) Moving upsampler to after subfilters.



(d) Replacing output structure by commutator.

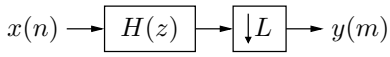
Figure 8.5 *Interpolator realized using polyphase decomposition.*

the output of the direct form structure may become significant and require extra pipelining registers.

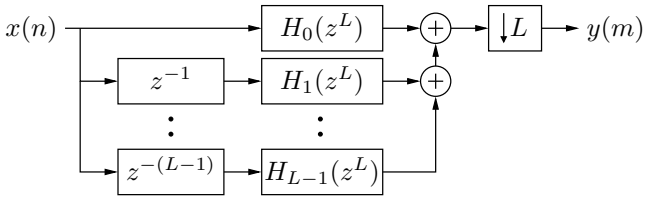
8.4.2 High-speed FIR filter architecture

In the case of implementations with high speed and short wordlengths, it may be more beneficial to consider realizations on the bit level rather than the word level. One such architecture is described here, and is also used as a base for the work in Chapter 12. First, the detailed derivation of a decimation filter on direct form is described, and then the resulting architecture for a transposed direct form realization is shown. Finally, the realizations of single rate and interpolation filters are discussed.

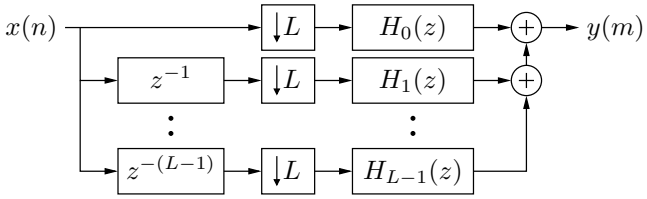
Consider the structure in Fig. 8.9(a), corresponding to the arithmetic part of the multirate decimation filter in Fig. 8.6. The system has L inputs $x_0(n), \ldots, x_{L-1}(n)$, L independent subfilters $H_0(z), \ldots, H_{L-1}(z)$, and one output $y(n)$. In Fig. 8.9(b),



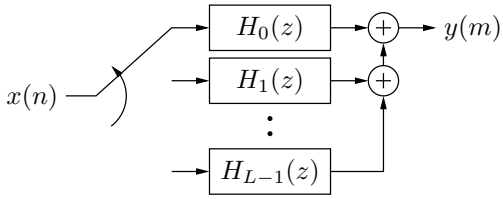
(a) Conventional decimation.



(b) Polyphase decomposition of filter.



(c) Moving downsampler to before subfilters.



(d) Replacing input structure by commutator.

Figure 8.6 *Decimator realized using polyphase decomposition.*

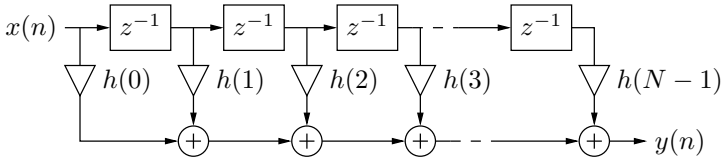


Figure 8.7 *Direct form FIR filter architecture.*

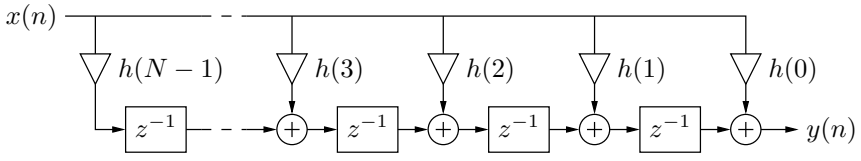
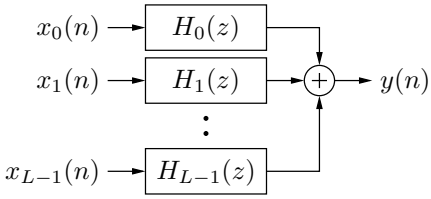
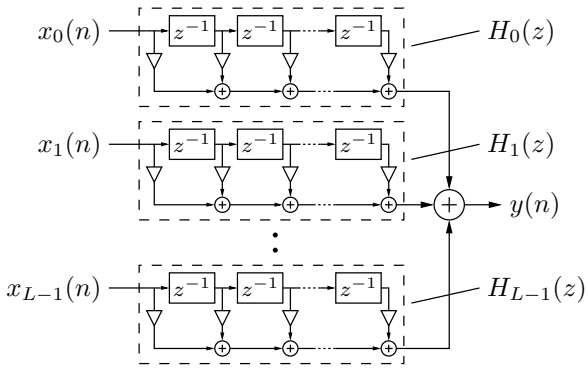


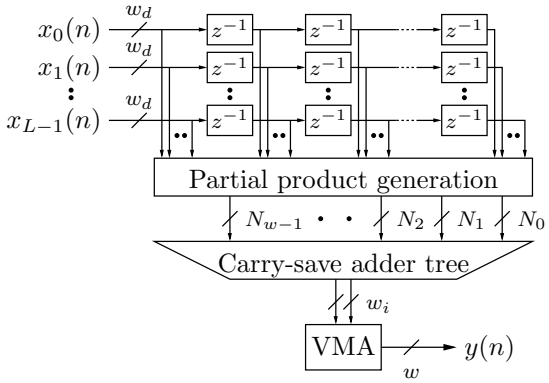
Figure 8.8 *Transposed direct form FIR filter architecture.*



(a) Multirate filter structure.



(b) Direct form realization of subfilters.



(c) Realization using partial product generation and carry-save adder tree.

Figure 8.9 *High-speed realization of FIR decimation filter on direct form using partial product generation and carry-save adders.*

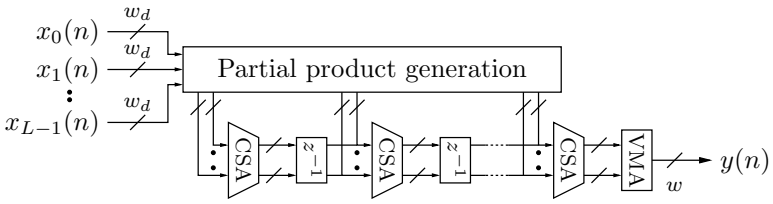


Figure 8.10 *High-speed architecture of FIR decimation filter on transposed direct form.*

the subfilters have been realized on direct form. Now, instead of directly computing all products of the inputs and coefficients, partial products corresponding to the non-zero bits of the coefficient are generated. The detailed generation of the partial products depends on the representation of the data and the coefficients, and is described in Sec. 12.1.2. However, the result is a number of partial products with different bit weights, represented by the partial product vectors in Fig. 8.9(c) where w_d is the input data wordlength, N_m is the number of partial products of bit weight m and w is the wordlength of $y(n)$. Typically, $N_m = 0$ for some m. The partial products are merged in a carry-save adder tree, which results in a still-redundant output of two vectors of length $w_i \leq w$. Finally, the vectors are merged by a vector merge adder to a non-redundant output $y(n)$.

A similar architecture on transposed direct form is shown in Fig. 8.10. In the transposed direct form architecture, the delay elements are at the output, and the partial product generation thus consists of several partitions of partial products corresponding to different delays. Each partition is reduced by a carry-save adder tree to a redundant form of two vectors in order to reduce the number of registers required.

The described architectures can be used also for single rate filters, simply by letting $L = 1$. It is also possible to realize an interpolation filter (Fig. 8.5) as L independent branches. However, for a direct form architecture, the delay elements of the input can typically be shared between the branches.

9

Sigma-delta data converters

In this chapter, a brief introduction to ΣΔ-ADCs is given. The basics of ΣΔ-modulators are given in Sec. 9.1. Signal and noise transfer functions of the modulator are defined, and expressions for the resulting quantization noise power and modulator SNR are obtained. In Sec. 9.2, some common structures of ΣΔ-modulators are shown. In Sec. 9.3, ADC systems using several ΣΔ-modulators in parallel are introduced. Finally, Sec. 9.4 gives some notes on implementations of decimation filters for ΣΔ-ADCs. More in-depth analyses of ΣΔ-ADCs are given in, e.g., [21, 81, 88].

9.1 Sigma-delta data conversion

9.1.1 Sigma-delta ADC overview

The ΣΔ-ADC is today often the preferred architecture for realizing low- to medium-speed analog-to-digital converters (ADCs) with effective resolution above 12 bits. Higher resolution than this is difficult to achieve for non-oversampled ADCs without laser trimming or digital error correction, since device matching-errors of semiconductor processes limit the accuracy of critical analog components [88]. The ΣΔ-ADC can overcome this problem by combining the speed advantage of analog circuits with the robustness and accuracy of digital circuits. Through oversampling

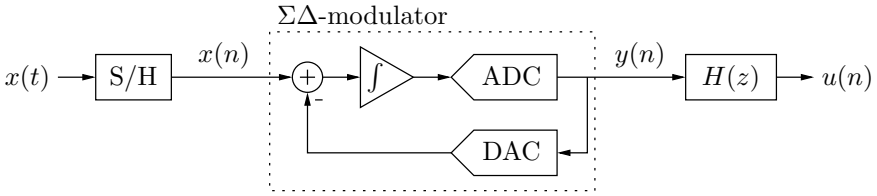


Figure 9.1 *Model of first order ΣΔ-ADC*

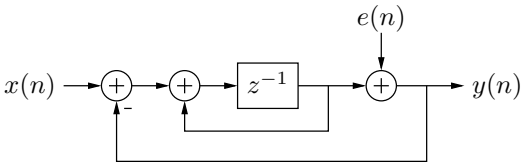


Figure 9.2 *Linear model of the first order ΣΔ-modulator in Fig. 9.1*

and noise shaping, the ΣΔ-modulator converts precise signal waveforms to an oversampled digital sequence where the information is localized in a narrow frequency band in which the quantization noise is heavily suppressed.

One of the prices to pay for these advantages is the required digital decimator operating on high sampling rate data. For Nyquist-rate CMOS ADCs, the power consumption increases approximately by a factor of four when increasing the resolution by one bit [93]. Hence, the power consumption of accurate Nyquist-rate ADCs tends to become very high. On the other hand, the analog circuitry of oversampled ΣΔ-modulators does not need to be accurate and power savings can therefore be made, in particular for continuous-time ΣΔ-modulators [53].

A model of a first order ΣΔ-ADC is shown in Fig. 9.1. The differentiator at the input subtracts the value of the previous conversion from the input, and the difference is integrated. Following the integration, the signal is sampled and quantized by a local ADC. The result is that the error of the previous conversion is subtracted from the next conversion, thus causing a suppression of the quantization error at low frequencies. Instead, the error appears at the higher frequencies. In the digital domain, a decimation filter can efficiently remove the noise, resulting in a signal that can achieve high resolution. However, in order to achieve high resolution, the sampling frequency has to be significantly larger than the signal bandwidth. This often causes the power dissipation of the digital decimation filter to constitute a major part of the power dissipation of the ADC.

9.1.2 Sigma-delta modulators

When analyzing the performance of a ΣΔ-modulator, the input signal and the quantization noise are assumed to be uncorrelated. With this assumption, the ΣΔ-modulator in Fig. 9.1 can be modelled by the structure in Fig. 9.2. Using

this model, a signal transfer function $F(z)$ and a noise transfer function $G(z)$ are defined such that the output $y(n)$ can be written

$$Y(z) = F(z)X(z) + G(z)E(z), \tag{9.1}$$

where $X(z)$, $E(z)$ and $Y(z)$ are the z-transforms of $x(n)$, $e(n)$ and $y(n)$, respectively. In the linear model of the first order $\Sigma\Delta$-modulator, the signal and noise transfer functions are given by

$$F(z) = z^{-1} \tag{9.2}$$

$$G(z) = \left(1 - z^{-1}\right). \tag{9.3}$$

Thus, the signal is passed unaffected by the modulator, whereas the quantization noise is high-pass shaped. It is also possible to construct higher order modulators, where the noise transfer function has a stronger high pass shape. Examples of the structures used in this thesis are included in Sec. 9.2. A straight-forward design of a kth order modulator yields signal and noise transfer functions given by

$$F(z) = z^{-k} \tag{9.4}$$

$$G(z) = \left(1 - z^{-1}\right)^k. \tag{9.5}$$

The noise transfer function $G(z)$ is shown in Fig. 9.3 for $\Sigma\Delta$-modulators of different orders. It is seen that, whereas the higher order modulators have a higher suppression of the noise at lower frequencies, the advantage comes at a cost of increased noise amplification at the higher frequencies. This increases the requirements of the digital decimation filter. Higher order modulators also have other considerations, including stability problems that reduce the allowed swing of the input signal, and analog matching problems. However, these are not further discussed in this thesis.

9.1.3 Quantization noise power

It is common to regard the quantization noise $e(n)$ as a white stochastic process with a uniform distribution in the interval $[-Q/2, Q/2]$, where Q is the quantization step. Assuming uniform quantization in the range $[-1, 1]$,

$$Q = \frac{2}{2^B - 1}, \tag{9.6}$$

where B is the number of bits used in the quantization. Let $R_{ee}(e^{j\omega T})$ denote the power spectral density (PSD) of the quantization noise. Then,

$$R_{ee}(e^{j\omega T}) = \frac{Q^2}{12} = \frac{1}{3(2^B - 1)^2}, \tag{9.7}$$

and the noise power at the output of the modulator can be written

$$R_{y_e y_e}(e^{j\omega T}) = R_{ee}(e^{j\omega T}) \left|G(e^{j\omega T})\right|^2 = \frac{4^K}{3(2^B - 1)^2} \sin^{2K}\left(\frac{\omega T}{2}\right) \tag{9.8}$$

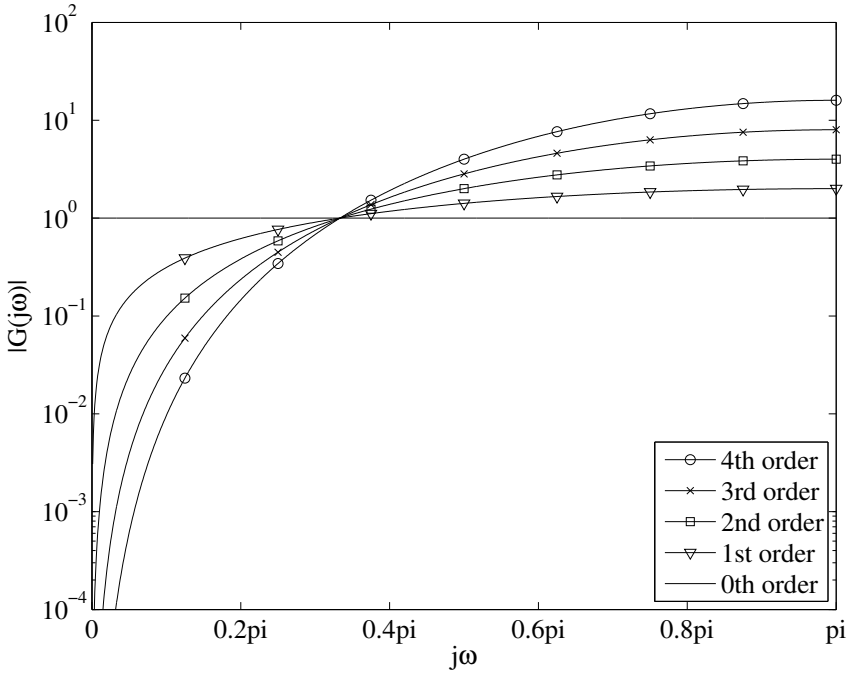


Figure 9.3 *Frequency response of noise transfer function $G(z)$ for ΣΔ-modulators of different orders*

The assumption of $e(n)$ to be independent of $x(n)$ is in reality not a well-motivated assumption. In contrast, $e(n)$ is highly dependent of $x(n)$ which causes the appearance of idle tones in the output spectrum. In addition, the assumption that the quantization noise is white with a uniform distribution is also not well motivated. In reality, the input is slow-changing due to being band-limited, which causes the quantization noise to be highly correlated in time. Both of these effects are well pronounced for single-bit ΣΔ-modulators, which are among the most commonly used in practice. In Chapter 11, an alternative analysis method is presented, with which the actual quantization noise for a specific input signal is used to evaluate the performance of different ΣΔ-modulators.

9.1.4 SNR estimation

Assume that the ΣΔ-modulator uses an oversampling ratio (OSR) of L, and thus that the input signal is bandlimited to the interval $\left[-\frac{\pi}{L}, \frac{\pi}{L}\right]$. The allowed input signal power depends on several things, including the modulator order, number of quantization bits, and the overloading behavior of the modulator, but assume for simplicity that a sine wave of unit amplitude is used. Then the input signal power

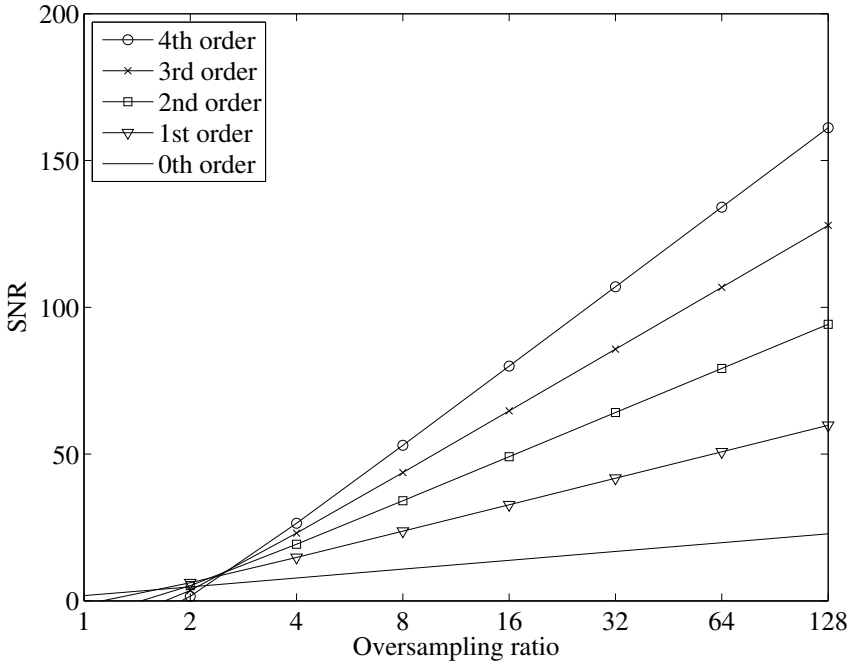


Figure 9.4 *SNR as function of OSR for single-bit ΣΔ-modulators of different orders.*

is $\sigma_x^2 = 1/2$. Since the signal transfer function is a pure delay, the output signal power is $\sigma_{yx}^2 = \sigma_x^2$. Now, the signal-to-noise ratio (SNR) at the output due to quantization noise can be written

$$\mathrm{SNR} = \frac{\sigma_{yx}^2}{\frac{1}{2\pi}\int_{-\frac{\pi}{L}}^{\frac{\pi}{L}} R_{y_e y_e}(e^{j\omega T})\mathrm{d}\omega}. \tag{9.9}$$

Figure 9.4 shows (9.9) evaluated for single-bit ΣΔ-modulators of different orders. The oversampling gain is $3 + 6K$ dB/octave for a ΣΔ-modulator of order K.

9.2 Modulator structures

In this thesis, three different ΣΔ-modulator structures are considered. These are presented in this section, and are the first order, single feedback ΣΔ-modulator shown in Fig. 9.5, the second order, double feedback ΣΔ-modulator shown in Fig. 9.6, and the fourth order multi-stage ΣΔ-modulator (MASH) shown in Fig. 9.7. The signal and noise transfer functions of the first and second order ΣΔ-modulators

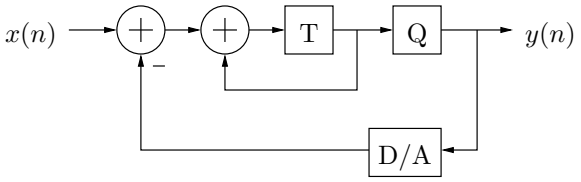


Figure 9.5 *A first-order, single-feedback ΣΔ-modulator.*

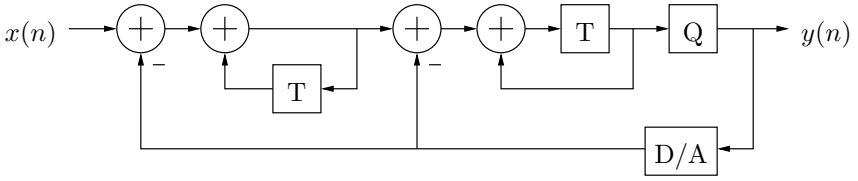


Figure 9.6 *A second-order, double-feedback ΣΔ-modulator.*

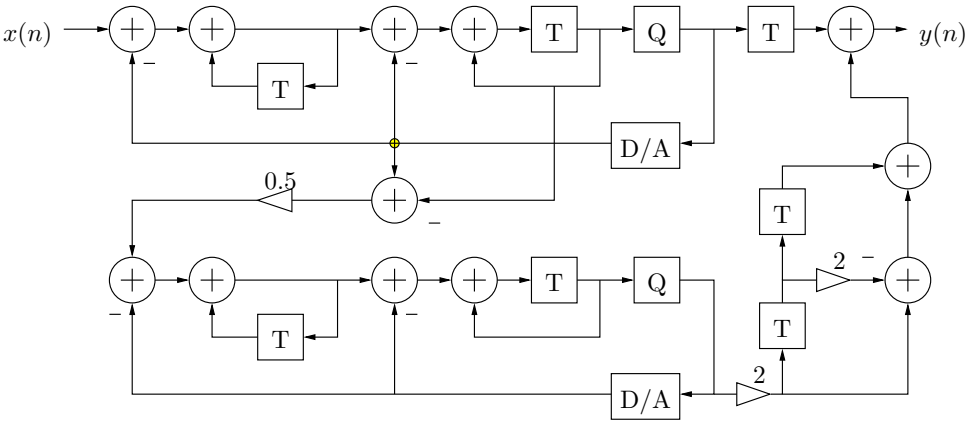


Figure 9.7 *A multistage noise-shaping (MASH) ΣΔ-modulator.*

are given by (9.4) and (9.5), respectively. For the MASH ΣΔ-modulator, the signal and noise transfer functions are given by

$$F_{\text{MASH}}(z) = z^{-2} \tag{9.10}$$

$$G_{\text{MASH}}(z) = 2(1 - z^{-1})^4. \tag{9.11}$$

9.3 Modulated parallel sigma-delta ADCs

Traditionally, ADCs and DACs based on ΣΔ-modulation have been used primarily for low bandwidth and high-resolution applications such as audio. The requirements make the architecture perfectly suited for this purpose. However,

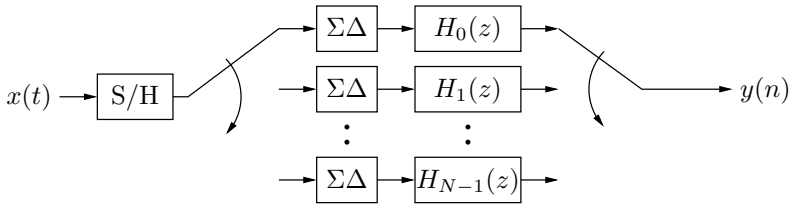


Figure 9.8 *Architecture of a parallel ΣΔ-ADC using time interleaving of the input signal.*

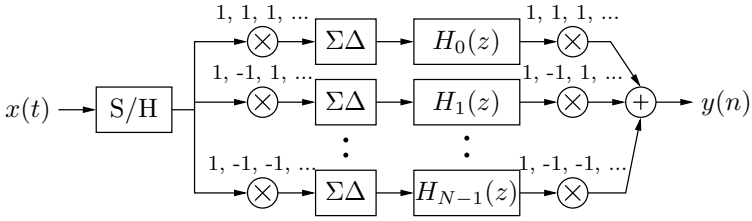


Figure 9.9 *Architecture of a parallel ΣΔ-ADC using Hadamard modulation of the input signal.*

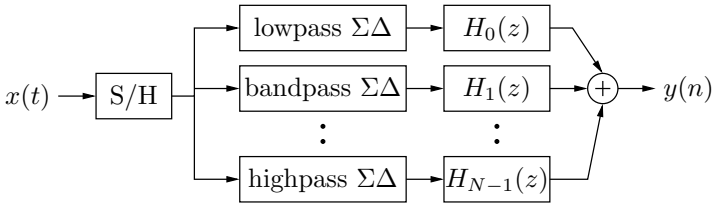


Figure 9.10 *Architecture of a parallel ΣΔ-ADC using frequency-band decomposition of the input signal.*

in later years, advancements in VLSI technology have allowed greatly increased clock frequencies, and ΣΔ-ADCs with bandwidths of tens of MHz have been reported [19, 103]. This makes it possible to use ΣΔ-ADCs in a wider context, for example in wireless communications. In order to reach such bandwidths with feasible resolutions, sampling frequencies in the GHz range are used. However, higher operating frequencies than this are hard to obtain with current technology. Even with reduced feature sizes, the increased performance with digital circuits comes from increased parallelism rather than increased operating frequencies.

One way to further increase the bandwidth of a ΣΔ-ADC is to use several modulators in parallel, where a part of the input signal is converted in each channel.

Several flavors of such ΣΔ-ADCs have been proposed, and these can essentially be divided into four categories: time-interleaved modulators [34, 35], Hadamard modulators [35, 40, 41, 61, 62], frequency-band decomposed modulators [29, 32, 35] and multirate modulators based on block-digital filtering [28, 56, 58, 64].

In the time-interleaved modulator, shown in Fig. 9.8, samples are interleaved in time between the channels. Each modulator is running at the input sampling rate, with its input grounded between consecutive samples. This is a simple scheme, but as interleaving causes aliasing of the spectrum, the channels have to be carefully matched in order to cancel aliasing in the deinterleaving at the output.

In a Hadamard modulator, shown in Fig. 9.9, the signal is modulated by a sequence constructed from the rows of a Hadamard matrix. One advantage over the time-interleaved modulator is an inherent coding gain, which increases the dynamic range of the ADC [35], whereas a disadvantage is that the number of channels is restricted to a number for which there exists a known Hadamard matrix. Another advantage, as is shown in Chapter 10, is the reduced sensitivity to mismatches in the analog circuitry.

The third category of parallel modulators is the frequency-band decomposed modulators, shown in Fig. 9.10, in which the signal is decomposed in frequency rather than time. This scheme is insensitive to analog mismatches, but has increased hardware complexity because it requires the use of bandpass modulators. The idea of the multirate modulators is different, based on a polyphase decomposition of the integrator in one channel, and is not further considered.

9.4 Data rate decimation

A significant part of the complexity, and power dissipation, of a ΣΔ-ADC is the digital decimation filter. This is especially true for the high-speed ADCs considered in this thesis, and for operation in the GHz range designing the decimation filter is a significant challenge. Normally, the decimation filter is designed in several stages to relax the requirements on each stage and reduce the overall complexity [94,100]. For the first stage, a common choice is the cascaded integrator comb (CIC) filter (also known as moving average filter). The impulse response of an N-tap (order $N-1$) CIC filter is

$$H(z) = \frac{1}{N}\sum_{i=0}^{N-1} z^{-i} = \frac{1}{N}\frac{1-z^{-N}}{1-z^{-1}} \tag{9.12}$$

If the number of taps, N, is selected as $N = M$, where M is the decimation rate, the CIC filter has zeros on the unit circle at $2i\pi/M$ rad for $i = 1, 2, \ldots, M-1$, i.e., the angles that are folded to 0 during the decimation. To increase the attenuation, several CIC filters are cascaded.

The main reason of the popularity of the CIC filter is the direct implementation of (9.12) as a cascade of an accumulator and a differentiator [20, 51], depicted in Fig. 9.11. This implementation has low arithmetic complexity, as for a cascade of L CIC filters only $2L$ adders are required. However, the wordlength of the

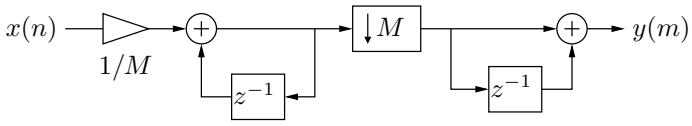


Figure 9.11 *Direct implementation of* (9.12) *as cascaded accumulator and differentiator.*

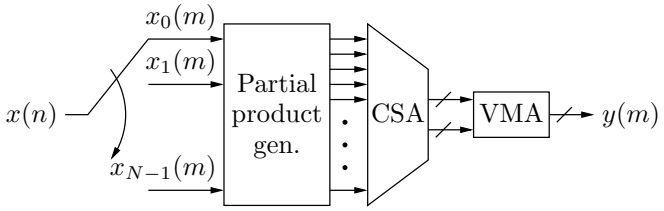


Figure 9.12 *General architecture of a decimation filter implemented using polyphase decomposition and partial product reduction with a carry-save adder tree.*

accumulators in a CIC filter is significantly longer than the input wordlength, which may lead to problems obtaining high throughput as the accumulators operate at the input sampling rate. This can be alleviated by the use of redundant arithmetic in the accumulators [76] or parallelizing the accumulators to operate at a lower sampling rate. However, this increases the complexity of the implementation which makes it less attractive for operations at high speed.

For decimation filters with short wordlength and high throughput requirements, it may be significantly more efficient to compute the impulse response of the cascaded filters and realize the resulting linear-phase FIR filter using polyphase decomposition [1, 42, 45, 67]. The realization using a direct form structure is described in Sec. 8.3, and a general architecture of a decimation filter implemented using polyphase decomposition is shown in Fig. 9.12. Another advantage of the architecture is the ability to use general FIR filters to allow an arbitrary set of filter coefficients, optimized for a suitable cost function. This is especially useful when using higher order $\Sigma\Delta$-modulators with arbitrary zero positioning for the noise transfer function.

10

Parallel sigma-delta ADCs

In this chapter, a multi-rate formulation of a system of parallel ΣΔ-ADCs is presented. The channels use modulation of the input sequence to achieve parallelism, and also allow the ΣΔ-modulators to differ between the channels. The formulation is used to analyze a system's sensitivity to mismatches between the channels, including channel gain, general ΣΔ-modulator mismatches and modulation sequence errors. It is shown that certain systems can become noise-free with limited calibration of a subset of the channels, whereas others may require full calibration of all channels. The multi-rate formulation is also used as a tool to devise a three-channel parallel ADC that is insensitive to gain mismatches in the channels.

In Sec. 10.1, the proposed multi-rate formulation is presented, and the resulting signal transfer function of the parallel system is derived. In Sec. 10.2, the model of one channel of the parallel system is described. The model includes analog imperfections such as modulator non-idealities, channel gain, offset errors and modulation sequence mismatches. In Sec. 10.3, the formulation is used to analyze the sensitivities of a time-interleaved ADC, Hadamard-modulated ADC and a frequency-band decomposed ADC, and the new insensitive modulation scheme is presented. Finally, the quantization noise characteristics of a parallel modulated system is discussed briefly in Sec. 10.4.

This chapter has been published in [16].

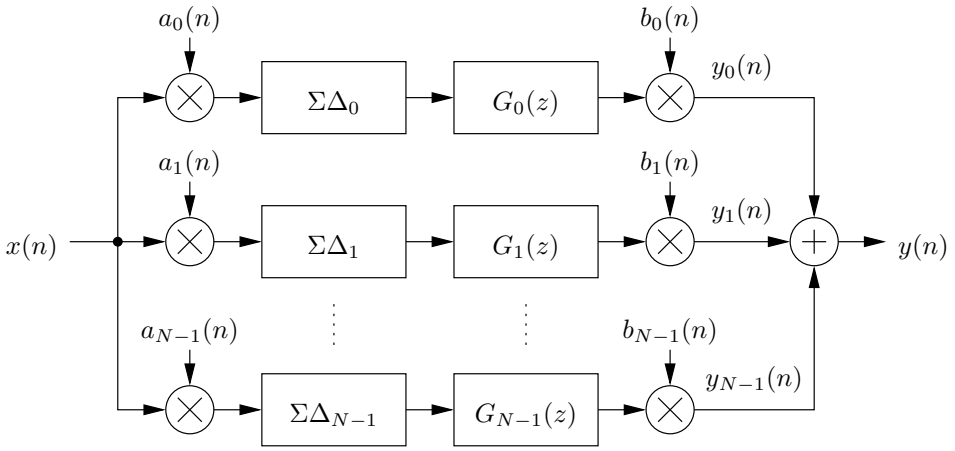
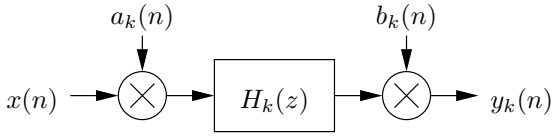


Figure 10.1 *ADC system using parallel ΣΔ-modulators and modulation sequences.*

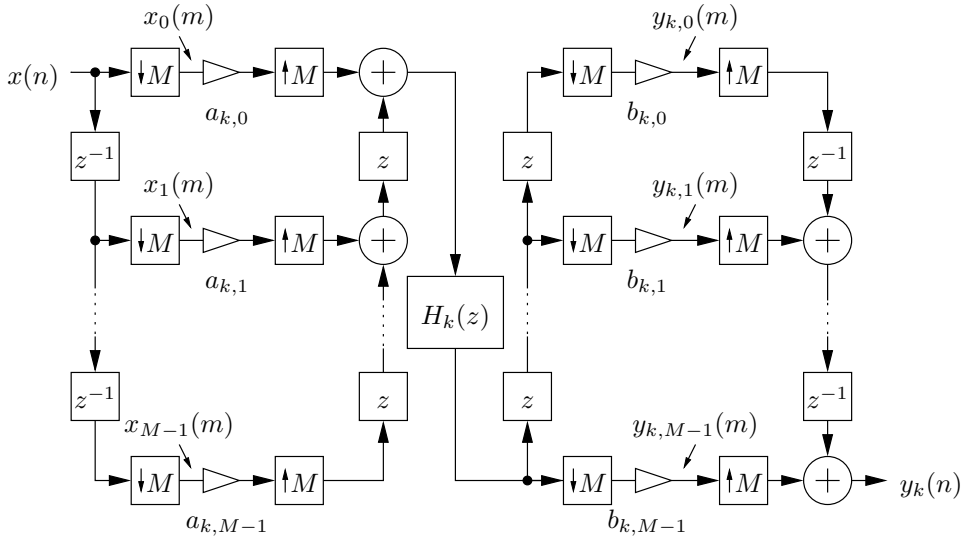
10.1 Linear system model

The system in Fig. 10.1 is considered. In this scheme, the input signal $x(n)$ is first divided into N channels. In each channel k, $k = 0, 1, \ldots, N-1$, the signal is first modulated by the M-periodic sequence $a_k(n) = a_k(n+M)$. The resulting sequence is then fed into a ΣΔ-modulator $\Sigma\Delta_k$, followed by a digital filter $G_k(z)$. The output of the filter is modulated by the M-periodic sequence $b_k(n) = b_k(n+M)$ which produces the channel output sequence $y_k(n)$. Finally, the overall output sequence $y(n)$ is obtained by summing all channel output sequences. The ΣΔ-modulator in each channel works in the same way as an ordinary ΣΔ-modulator. By increasing the channel oversampling, and reducing the passband width of the channel filter accordingly, more of the shaped noise is removed, and the resolution is increased. By using several channels in parallel, wider signal bands can be handled without increasing the input sampling rate to unreasonable values. In other words, instead of using one single ΣΔ-ADC with a very high input sampling rate, a number of ΣΔ-ADCs in parallel provide essentially the same resolution but with a reasonable input sampling rate. Traditional parallel systems that are covered by the proposed model include the time-interleaved, Hadamard modulated and frequency-band decomposed ADCs discussed in Sec. 9.3.

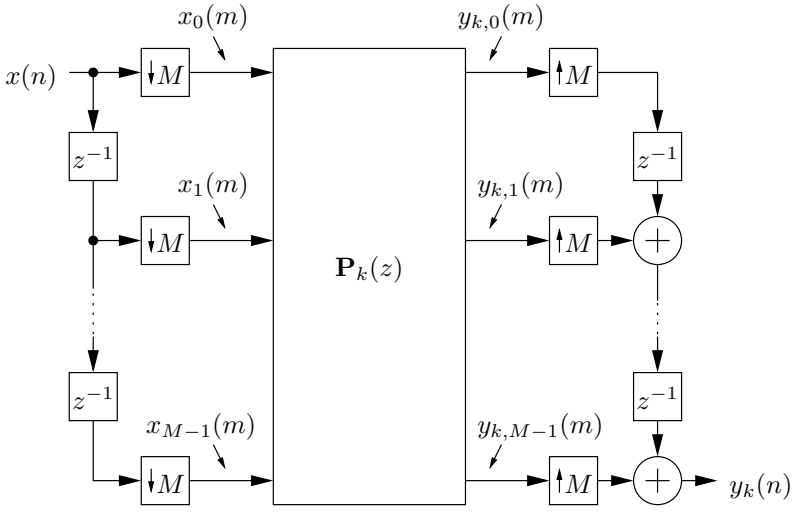
The overall output $y(n)$ is determined by the input $x(n)$, the signal transfer function of the system, and the quantization noise generated in the ΣΔ-modulators. Using a linear model for analysis, the signal input-to-output relation and noise input-to-output relation can be analyzed separately. The signal transfer function from $x(n)$ to $y(n)$ should equal (or at least approximate) a delay in the frequency band of interest. The main problem in practice is that the overall scheme is subject



(a) Model of channel.



(b) Polyphase decomposition of multipliers.



(c) Multirate formulation of a channel.

Figure 10.2 *Equivalent signal transfer models of a channel of the parallel system in Fig. 10.1*

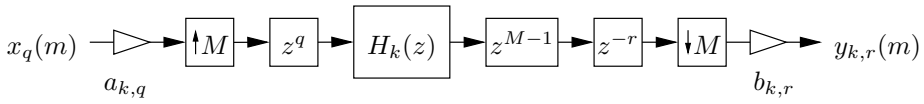


Figure 10.3 *Path from $x_q(m)$ to $y_{k,r}(m)$ in channel k as depicted in Fig. 10.2(b).*

to channel gain, offset, and modulation sequence level mismatches [4, 35, 57]. This is where the new general formulation becomes useful as it gives a relation between the input and output from which one can easily deduce a particular scheme's sensitivity to mismatch errors. The noise contribution, on the other hand, is essentially unaffected by channel mismatches. Therefore, the noise analysis can be handled in the traditional way, as in Sec. 10.4.

10.1.1 Signal transfer function

From the signal input-to-output point of view, the system depicted in Fig. 10.2(a) is obtained for channel k. Here, each $H_k(z)$ represents a cascade of the corresponding signal transfer function of the $\Sigma\Delta$-modulator and the digital filter $G_k(z)$. To derive a useful input-output relation in the z-domain, multirate filter bank theory [94] is used. As $a_k(n)$ and $b_k(n)$ are M-periodic sequences, each multiplication can be modelled as M branches with constant multiplications and with the samples interleaved between the branches. This is shown in the structure in Fig. 10.2(b), where

$$a_{k,n} = \begin{cases} a_k(0) & \text{for } n = 0 \\ a_k(M-n) & \text{for } n = 1, 2, \ldots, M-1 \end{cases} \tag{10.1}$$

$$b_{k,n} = b_k(M-1-n) \quad \text{for } n = 0, 1, \ldots, M-1. \tag{10.2}$$

Now, consider the system shown in Fig. 10.3, representing the path from $x_q(m)$ to $y_{k,r}(m)$ in Fig. 10.2(b). Denoting

$$\tilde{H}_k(z) = z^{M-1} H_k(z), \tag{10.3}$$

the transfer function from $x_q(m)$ to $y_{k,r}(m)$ is given by the first polyphase component in the polyphase decomposition of $z^q \tilde{H}_k(z) z^{-r}$, scaled by $a_{k,q} b_{k,r}$. For $p = q - r = 0, 1, \ldots, M-1$, the polyphase decomposition of $z^p \tilde{H}_k(z)$ can be written

$$z^p \tilde{H}_k(z) = \sum_{i=0}^{M-1} z^{p-i} \tilde{H}_{k,i}(z^M), \tag{10.4}$$

and the first polyphase component is $\tilde{H}_{k,p}(z)$, i.e., the pth polyphase component of $\tilde{H}_k(z)$ as specified by the Type 1 polyphase representation in Sec. 8.3. For

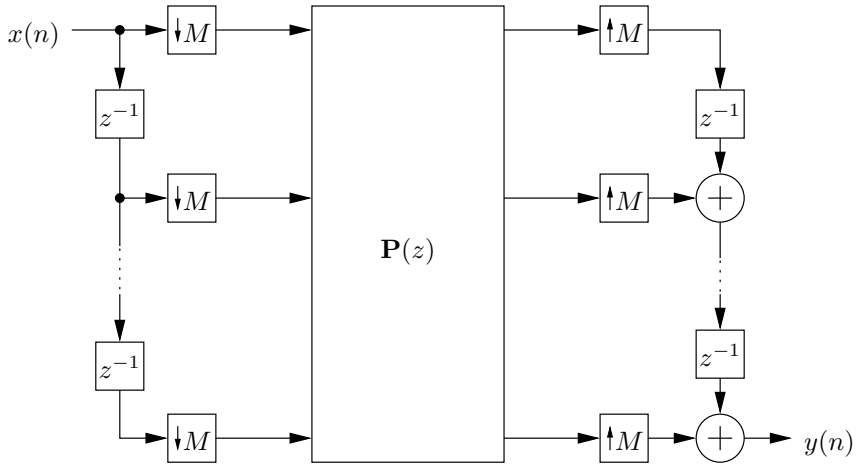


Figure 10.4 *Equivalent representation of the system in Fig. 10.1 based on the equivalences in Fig. 10.2.* $\mathbf{P}(z)$ *is given by* (10.8).

$p = -M+1, \ldots, -1,$

$$z^p \tilde{H}_k(z) = \sum_{i=0}^{M-1} z^{p-i+M} z^{-M} \tilde{H}_{k,i}(z^M) \tag{10.5}$$

and the first polyphase component is $z^{-1}\tilde{H}_{k,p+M}(z)$. Returning to the system in Fig. 10.2(b), the transfer functions $P_k^{r,q}(z)$ from $x_q(m)$ to $y_{k,r}(m)$ can now be written

$$P_k^{r,q}(z) = \begin{cases} b_{k,r}\tilde{H}_{k,q-r}(z)a_{k,q} & \text{for } q \geq r \\ b_{k,r}z^{-1}\tilde{H}_{k,q-r+M}(z)a_{k,q} & \text{for } q < r. \end{cases} \tag{10.6}$$

The relations can be written in matrix form as

$$\mathbf{P}_k(z) = \begin{bmatrix} a_{k,0}b_{k,0}\tilde{H}_{k,0}(z) & a_{k,1}b_{k,0}\tilde{H}_{k,1}(z) & \cdots & a_{k,M-1}b_{k,0}\tilde{H}_{k,M-1}(z) \\ a_{k,0}b_{k,1}z^{-1}\tilde{H}_{k,M-1}(z) & a_{k,1}b_{k,1}\tilde{H}_{k,0}(z) & \cdots & a_{k,M-1}b_{k,1}\tilde{H}_{k,M-2}(z) \\ a_{k,0}b_{k,2}z^{-1}\tilde{H}_{k,M-2}(z) & a_{k,1}b_{k,2}z^{-1}\tilde{H}_{k,M-1}(z) & \cdots & a_{k,M-1}b_{k,2}\tilde{H}_{k,M-3}(z) \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,0}b_{k,M-1}z^{-1}\tilde{H}_{k,1}(z) & a_{k,1}b_{k,M-1}z^{-1}\tilde{H}_{k,2}(z) & \cdots & a_{k,M-1}b_{k,M-1}\tilde{H}_{k,0}(z) \end{bmatrix}, \tag{10.7}$$

and it is thus obvious that one channel of the system can be represented by the structure in Fig. 10.2(c). In the whole system (Fig. 10.1) a number of such channels are summed at the output, and the parallel system of N channels can be represented

by the structure in Fig. 10.4, where the matrix $\mathbf{P}(z)$ is given by

$$\mathbf{P}(z) = \sum_{k=0}^{N-1} \mathbf{P}_k(z). \tag{10.8}$$

For convenience, we write (10.7) as

$$\mathbf{P}_k(z) = \mathbf{S}_k \cdot \tilde{\mathbf{H}}_k(z) \tag{10.9}$$

where the multiplication is element-wise, and where $\tilde{\mathbf{H}}_k(z)$ and $\mathbf{S}_k$ are given by

$$\tilde{\mathbf{H}}_k(z) = \begin{bmatrix} \tilde{H}_{k,0}(z) & \tilde{H}_{k,1}(z) & \cdots & \tilde{H}_{k,M-1}(z) \\ z^{-1}\tilde{H}_{k,M-1}(z) & \tilde{H}_{k,0}(z) & \cdots & \tilde{H}_{k,M-2}(z) \\ z^{-1}\tilde{H}_{k,M-2}(z) & z^{-1}\tilde{H}_{k,M-1}(z) & \cdots & \tilde{H}_{k,M-3}(z) \\ \vdots & \vdots & \ddots & \vdots \\ z^{-1}\tilde{H}_{k,1}(z) & z^{-1}\tilde{H}_{k,2}(z) & \cdots & \tilde{H}_{k,0}(z) \end{bmatrix} \tag{10.10}$$

and

$$\mathbf{S}_k = \begin{bmatrix} a_{k,0}b_{k,0} & a_{k,1}b_{k,0} & \cdots & a_{k,M-1}b_{k,0} \\ a_{k,0}b_{k,1} & a_{k,1}b_{k,1} & \cdots & a_{k,M-1}b_{k,1} \\ a_{k,0}b_{k,2} & a_{k,1}b_{k,2} & \cdots & a_{k,M-1}b_{k,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,0}b_{k,M-1} & a_{k,1}b_{k,M-1} & \cdots & a_{k,M-1}b_{k,M-1} \end{bmatrix}, \tag{10.11}$$

respectively. (10.11) can equivalently be written as

$$\mathbf{S}_k = \mathbf{b}_k^T \mathbf{a}_k \tag{10.12}$$

where

$$\begin{aligned} \mathbf{a}_k &= \begin{bmatrix} a_{k,0} & a_{k,1} & \cdots & a_{k,M-1} \end{bmatrix}, \\ \mathbf{b}_k &= \begin{bmatrix} b_{k,0} & b_{k,1} & \cdots & b_{k,M-1} \end{bmatrix}, \end{aligned} \tag{10.13}$$

and T stands for transpose. Examples of the $\mathbf{S}_k$-matrices and the $\mathbf{a}_k$- and $\mathbf{b}_k$-vectors are provided for a time-interleaved system, a Hadamard-modulated system and a frequency-band decomposed system in Sec. 10.3.1, Sec. 10.3.2, and Sec. 10.3.3, respectively.

10.1.2 Alias-free system

With the system represented as above, it is known that it is alias-free, and thus time-invariant, if and only if the matrix $\mathbf{P}(z)$ is pseudocirculant [94]. Under this condition, the output z-transform becomes

$$Y(z) = H_{\mathrm{A}}(z)X(z), \tag{10.14}$$

where

$$H_{\mathrm{A}}(z) = z^{-M+1} \sum_{k=0}^{N-1} \sum_{i=0}^{M-1} s_k^{0,i} z^{-i} \tilde{H}_{k,i}(z^M) = \sum_{k=0}^{N-1} \sum_{i=0}^{M-1} s_k^{0,i} z^{-i} H_{k,i}(z^M) \quad (10.15)$$

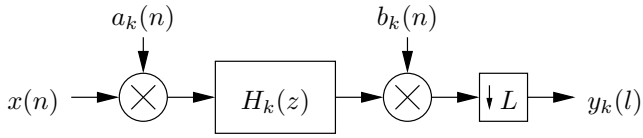
with $s_k^{0,i}$ denoting the elements on the first row of $\mathbf{S}_k$. This case corresponds to a Nyquist sampled ADC of which two special cases are the time-interleaved ADC [34, 58] and Hadamard-modulated ADC [40]. These systems are also described in the context of the multirate formulation in Sec. 10.3.1 and Sec. 10.3.2, respectively.

Regarding $\tilde{\mathbf{H}}_k(z)$, it is seen in (10.10) that it is pseudocirculant for an arbitrary $\tilde{H}_k(z)$. It would thus be sufficient to make $\mathbf{S}_k$ circulant for each channel k in order to make each $\mathbf{P}_k(z)$ pseudocirculant and end up with a pseudocirculant $\mathbf{P}(z)$. Unfortunately, the set of circulant real-valued $\mathbf{S}_k$ achievable by the construction in (10.12) is limited, because the rank of $\mathbf{S}_k$ is one. However, for purposes of error cancellation between channels it is beneficial to group the channels in sets where the matrices within each set sum to a circular matrix. The channel set $\{0, 1, \ldots, N-1\}$ is thus partitioned into the sets $C_0, \ldots, C_{I-1}$, where each sum $\sum_{k \in C_i} \mathbf{S}_k$ is a circulant matrix. It is assumed that the modulators and filters are identical for channels belonging to the same partition, $\mathbf{H}_k(z) = \mathbf{H}_l(z)$ whenever $k, l \in C_i$, and thus $\tilde{\mathbf{H}}_k(z) = \tilde{\mathbf{H}}_l(z)$. The matrix for partition i is denoted $\tilde{\mathbf{H}}_{0,i}(z)$. Sensitivity to channel mismatches are discussed further in Sec. 10.2.

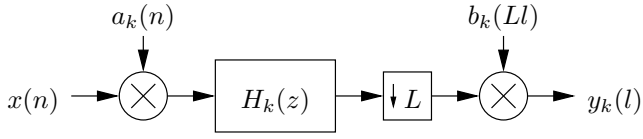
10.1.3 L-decimated alias-free system

A system is denoted an L-decimated alias-free system if it is alias-free before decimation by a factor of L. A channel of such a system is shown in Fig. 10.5(a). Obviously, the decimation can be performed before the modulation, as shown in Fig. 10.5(b), if the index of the modulation sequence is scaled by a factor of L. Considering the equivalent system in Fig. 10.5(c), it is apparent that the downsampling by L can be moved to after the scalings by $b_{k,l}$ if the delay elements z^{-1} are replaced by L-fold delay elements z^{-L}. The system may then be described as in Fig. 10.5(d), where $\mathbf{P}_k(z)$ is defined by (10.6). However, the outputs are taken from every Lth row of $\mathbf{P}_k(z)$, such that the first output $y_{k,L-1 \bmod M}(m)$ is taken from row L, the second output $y_{k,2L-1 \bmod M}(m)$ is taken from row $(2L-1 \bmod M)+1$, and so on. It is thus apparent that only rows $\gcd(L, M) \cdot i, i = 0, 1, 2, \ldots$ are used.

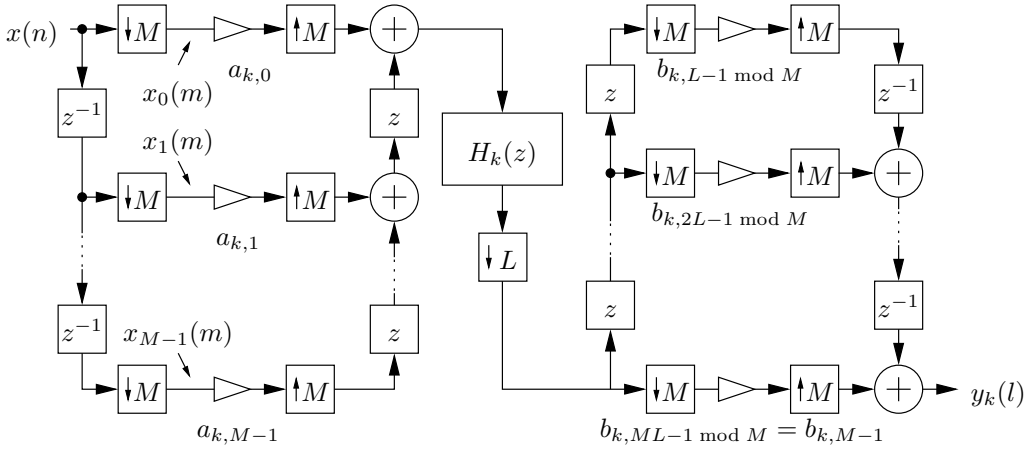
The L-decimated system corresponds to an oversampled ADC. The main observation that should be made is that the L-decimated system may be described in the same way as the critically sampled (non-oversampled) system, but that relaxations may be allowed on the requirements of the modulation sequences. As only a subset of the rows of $\mathbf{P}(z)$ are used, the matrix needs to be pseudo-circulant only on these rows. As in the critically sampled case, the channel set $\{0, 1, \ldots, N-1\}$ is partitioned into sets $C_0, \ldots, C_{I-1}$ where the matrix $\sum_{k \in C_i} \mathbf{S}_k$ is circulant on the rows $\gcd(L, M) \cdot i, i = 0, 1, 2, \ldots$, and $\tilde{\mathbf{H}}_k(z) = \tilde{\mathbf{H}}_l(z) = \tilde{\mathbf{H}}_{0,i}(z)$ when $k, l \in C_i$. The oversampled Hadamard-modulated system in [41] belongs to this category of the formulation, and another example of a decimated system is given in Sec. 10.3.4.



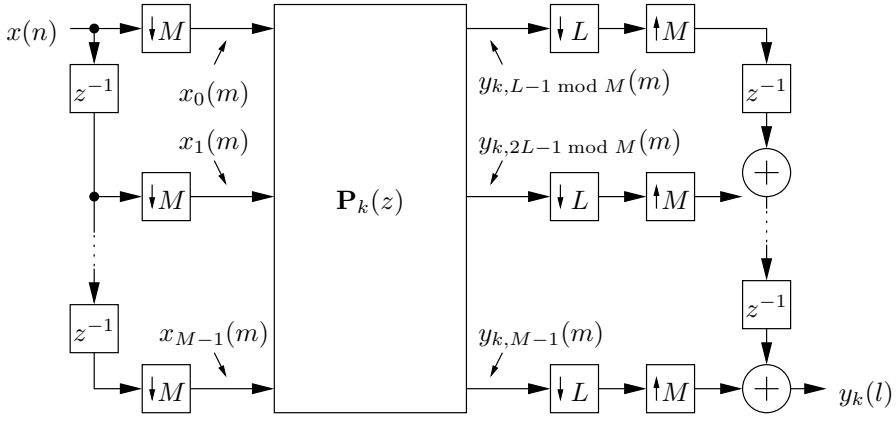
(a) Decimation at output.



(b) Internal decimation.



(c) Polyphase decomposition of input and output.



(d) Multirate formulation of a channel. $y_{k,L}(m)$ denotes the output pertaining to the Lth row of $\mathbf{P}_k(z)$.

Figure 10.5 *Channel model of L-decimated system.*

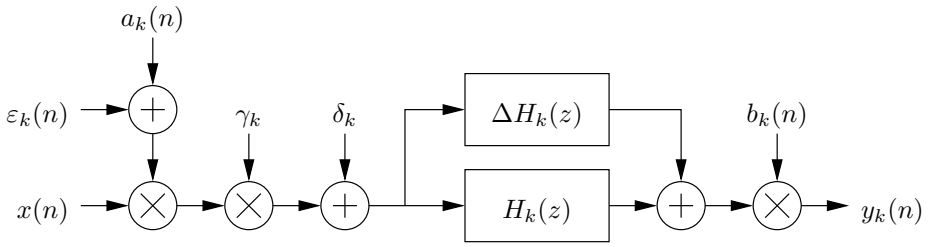


Figure 10.6 *Channel model with nonideal analog circuits.*

10.2 Sensitivity to channel mismatches

In this section, the channel model used for the sensitivity analysis is explained. Figure 10.6 shows one of the channels of a parallel ΣΔ-ADC, where several nonidealities resulting from imperfect analog circuits have been included. Difficulties in realizing the exact values of the analog modulation sequence are modelled by an additive error term $\varepsilon_k(n)$. The error is assumed to be static, i.e., it depends only on the value of $a_k(n)$, and is therefore a periodic sequence with the same periodicity as $a_k(n)$. The time-varying error $\varepsilon_k(n)$ may be a major concern when the modulation sequences contain non-trivial elements, i.e., elements that are not -1, 0, or 1. The trivial elements may be realized without a multiplier by exchanging, grounding, or passing through the inputs to the modulator, and are for this reason particularly attractive on the analog side.

A channel-specific gain γ_k is included in the sensitivity analysis, and analog imperfections in the modulator are modelled as the transfer function $\Delta H_k(z)$. The modulator nonidealities including channel gain and modulation sequence errors are analyzed separately in the context of the multirate formulation. In practice, there is also a channel offset δ_k which is not suitable for analysis in this context, as it is signal independent. Channel offsets are commented in Sec. 10.2.4.

10.2.1 Modulator nonidealities

Assume that the ideal system is alias-free, i.e., the matrix $\mathbf{P}(z) = \sum \mathbf{P}_k(z)$ is pseudocirculant. Due to analog circuit errors the transfer function of channel k deviates from the ideal $H_k(z)$ to $\gamma_k(H_k(z) + \Delta H_k(z))$, and $\tilde{H}_k(z)$ is replaced by $\hat{H}_k(z) = \gamma_k(H_k(z)+\Delta H_k(z))z^{M-1}$. The transfer matrix for channel k thus becomes $\hat{\mathbf{P}}_k(z)$ with elements

$$\hat{P}_k^{j,i}(z) = \begin{cases} b_{k,j}\hat{H}_{k,i-j}(z)a_{k,i} & \text{for } i \geq j \\ b_{k,j}z^{-1}\hat{H}_{k,i-j+M}(z)a_{k,i} & \text{for } i < j \end{cases} \tag{10.16}$$

where $\hat{H}_{k,p}(z)$ are the polyphase components of $\hat{H}_k(z)$. It is apparent that $\hat{\mathbf{P}}_k(z)$ is pseudocirculant whenever $\mathbf{P}_k(z)$ is. Thus a system where all the $\mathbf{S}_k$ matrices are circulant is completely insensitive to modulator mismatches.

In the general case, unfortunately, all $\mathbf{S}_k$ are not circulant and $\sum \hat{\mathbf{P}}_k(z) = \sum \mathbf{S}_k \cdot \hat{\mathbf{H}}_k(z)$ does not sum up to a pseudocirculant matrix as the matrices $\hat{\mathbf{H}}_k(z)$ are different between the channels. Partitioning the channel set into the sets C_i, as described in Sec. 10.1.1, and matching the modulators of channels belonging to the same partition C_i, i.e., defining $\gamma_k = \gamma_l$ and $\Delta H_k(z) = \Delta H_l(z)$ when $k, l \in C_i$, allows $\hat{\mathbf{P}}(z)$ to be written

$$\hat{\mathbf{P}}(z) = \sum_{k=0}^{N-1} \mathbf{S}_k \cdot \hat{\mathbf{H}}_k(z) = \sum_{i=0}^{I-1} \hat{\mathbf{H}}_{0,i}(z) \cdot \sum_{k \in C_i} \mathbf{S}_k \tag{10.17}$$

and it is apparent that each term in the outer sum is pseudocirculant, and thus that $\mathbf{P}(z)$ is as well. Thus the system is alias-free and non-linear distortion is eliminated.

10.2.2 Modulation sequence errors

It is assumed that the ideal system is alias-free, i.e., that $\mathbf{P}(z) = \sum \mathbf{P}_k(z)$ is pseudocirculant. Due to difficulties in realizing the analog modulation sequence, the signal is modulated in channel k by the sequence $\hat{\mathbf{a}}_k = \mathbf{a}_k + \boldsymbol{\varepsilon}_k$ rather than the ideal sequence $\mathbf{a}_k$. We consider here different choices of the modulation sequences.

Bi-level sequence for an insensitive channel

Assume that an analog modulation sequence with two levels is used for an insensitive channel, i.e., $\mathbf{S}_k = \mathbf{b}_k^T \mathbf{a}_k$ is a circular matrix. Examples of this type of channel include the first two channels of a Hadamard modulated system. Assuming that the sequence errors $\boldsymbol{\varepsilon}_k$ depend only on $\mathbf{a}_k$, i.e., $\varepsilon_k(n_1) = \varepsilon_k(n_2)$ when $a_k(n_1) = a_k(n_2)$, the modulation vector can be written $\hat{\mathbf{a}}_k = \alpha_k \mathbf{a}_k + \begin{bmatrix} \beta_k & \beta_k & \cdots & \beta_k \end{bmatrix}$ for some values of the scaling factor α_k and offset term β_k. The channel matrix $\hat{\mathbf{P}}_k(z)$ for the channel modulated with the sequence $\hat{\mathbf{a}}_k$ then becomes

$$\begin{aligned} \hat{\mathbf{P}}_k(z) &= \mathbf{b}_k^T (\alpha_k \mathbf{a}_k + \begin{bmatrix} \beta_k & \beta_k & \cdots & \beta_k \end{bmatrix}) \cdot \tilde{\mathbf{H}}_k(z) = \\ &= \alpha_k \mathbf{S}_k \cdot \tilde{\mathbf{H}}_k(z) + \beta_k \mathbf{B}_k \tilde{\mathbf{H}}_k(z), \end{aligned} \tag{10.18}$$

where $\mathbf{B}_k$ is a diagonal matrix consisting of the elements of $\mathbf{b}_k$. The first term is pseudocirculant, and thus the system is insensitive to modulation sequence scaling factors in channel k. The impact of the offset term β_k, i.e., the second term, is explained in Sec. 10.2.3.

Bi-level sequence for sensitive channels

Consider one of the subsets C_i in the partition of the channel set. The sum of the $\mathbf{S}_k$-matrices corresponding to the channels in the set, $\sum_{k \in C_i} \mathbf{S}_k$, is a circulant matrix, whereas the constituent matrices are not. Examples of this type of channels are the time-interleaved systems and the Hadamard modulated systems with more

than two channels. As in the insensitive case, the modulation vectors are written $\hat{\mathbf{a}}_k = \alpha_k \mathbf{a}_k + \begin{bmatrix}\beta_k & \beta_k & \cdots & \beta_k\end{bmatrix}$, and the sum of the channel matrices for the channel subset becomes

$$\sum_{k \in C_i} \hat{\mathbf{P}}_k(z) = \sum_{k \in C_i} \mathbf{b}_k^T (\alpha_k \mathbf{a}_k + \begin{bmatrix}\beta_k & \beta_k & \cdots & \beta_k\end{bmatrix}) \cdot \tilde{\mathbf{H}}_k(z) = \\ = \left(\tilde{\mathbf{H}}_{0,i}(z) \cdot \sum_{k \in C_i} \alpha_k \mathbf{S}_k \right) + \sum_{k \in C_i} \beta_k \mathbf{B}_k \tilde{\mathbf{H}}_k(z), \tag{10.19}$$

where $\mathbf{B}_k$ is a diagonal matrix consisting of the elements of $\mathbf{b}_k$. The first sum is generally not a pseudocirculant matrix, and the channels are thus sensitive to sequence gain errors. If the gains are matched, denote $\alpha_{0,i} = \alpha_k = \alpha_l$ when $k, l \in C_i$, the channel matrix sum may be written

$$\sum_{k \in C_i} \hat{\mathbf{P}}_k(z) = \left(\alpha_{0,i} \tilde{\mathbf{H}}_{0,i}(z) \cdot \sum_{k \in C_i} \mathbf{S}_k \right) + \sum_{k \in C_i} \beta_k \mathbf{B}_k \tilde{\mathbf{H}}_k(z), \tag{10.20}$$

and it is seen that the first term is a pseudocirculant matrix, and the channel set is alias-free. Again, the impact of the offset term β_k is explained in Sec. 10.2.3.

Multi-level sequences

If an insensitive channel is modulated with a multi-level sequence $\hat{\mathbf{a}}_k = \mathbf{a}_k + \boldsymbol{\varepsilon}_k$ the channel matrix becomes

$$\hat{\mathbf{P}}_k(z) = \mathbf{b}_k^T (\mathbf{a}_k + \boldsymbol{\varepsilon}_k) \cdot \tilde{\mathbf{H}}_k(z) = \mathbf{S}_k \cdot \tilde{\mathbf{H}}_k(z) + \mathbf{b}_k^T \boldsymbol{\varepsilon}_k \cdot \tilde{\mathbf{H}}_k(z), \tag{10.21}$$

which is pseudocirculant only if $\mathbf{b}_k^T \boldsymbol{\varepsilon}_k$ is a circulant matrix. Systems with multi-level analog modulation sequences are thus sensitive to level errors.

10.2.3 Modulation sequence offset errors

Consider the modulation sequence offset errors introduced in Sec. 10.2.2. The channel matrix for a channel with a modulation sequence containing an offset error can be written as (10.18). Thus the error pertaining to the sequence offset is additive, and can be modelled as in Fig. 10.7. The signal is thus first filtered through $H_k(z)$ and then aliased by the system $\mathbf{B}_k$, as $\mathbf{B}_k$ is not pseudocirculant unless the elements in the digital modulation sequence $\mathbf{b}_k$ are identical. However, as the signal is first filtered, only signal components in the passband of $H_k(z)$ will cause aliasing. If the signal contains no information in this band, aliasing will be completely suppressed. Typically the signal has a guard band either at the low-frequency or high-frequency region to allow transition bands of the filters, and the modulator can then be suitably chosen as either a low-pass type or high-pass type [80], respectively. Errors pertaining to sequence offsets are demonstrated in Sec. 10.3.1.

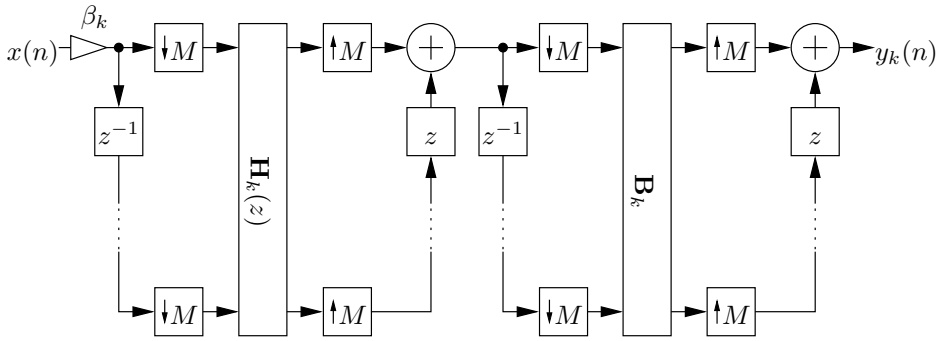


Figure 10.7 *Model of errors in a parallel system pertaining to sequence offsets.*

10.2.4 Channel offset errors

Channel offsets must be removed for each channel in order not to overload the $\Sigma\Delta$-modulator. Offsets affect the system in a nonlinear way and may not be analyzed using the multirate formulation. However, the problem has been well investigated and numerous solutions exist [4, 58, 81].

10.3 Simulation results

In this section, the proposed multi-rate formulation is used to analyze the sensitivity to channel mismatch errors for some different systems. Results are provided for a time-interleaved ADC, a Hadamard modulated ADC and a frequency-band decomposed ADC, as described in Sec. 9.3. Also, an example is provided of how the formulation can be used to derive a new architecture that is insensitive to channel matching errors.

10.3.1 Time-interleaved ADC

Consider a time-interleaved ADC with four channels. The samples are interleaved between the channels, each encompassing identical second-order low-pass modulators and decimation filters. Ideally, their z-domain transforms may be written

$$H_k(z) = H(z) = \begin{cases} z^{-1} & -\pi/4 \leq \omega T \leq \pi/4 \\ 0 & \text{otherwise.} \end{cases} \tag{10.22}$$

All modulators are running at the input sampling rate, with their inputs grounded between consecutive samples. Thus the modulation sequences are

$$\begin{aligned} a_0(n) &= b_0(n) = 1, 0, 0, 0, \ldots, \\ a_1(n) &= b_1(n) = 0, 1, 0, 0, \ldots, \\ a_2(n) &= b_2(n) = 0, 0, 1, 0, \ldots, \\ a_3(n) &= b_3(n) = 0, 0, 0, 1, \ldots, \end{aligned} \tag{10.23}$$

all periodic with period $M = 4$. The vectors $\mathbf{a}_k$ and $\mathbf{b}_k$ are as defined by (10.13)

$$\begin{aligned} \mathbf{a}_0 = \mathbf{b}_3 &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{a}_1 = \mathbf{b}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{a}_2 = \mathbf{b}_1 &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{a}_3 = \mathbf{b}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \end{aligned} \tag{10.24}$$

The matrices $\mathbf{S}_k$, defined by (10.12), then become

$$\mathbf{S}_0 = \mathbf{b}_0^T \mathbf{a}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{S}_1 = \mathbf{b}_1^T \mathbf{a}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

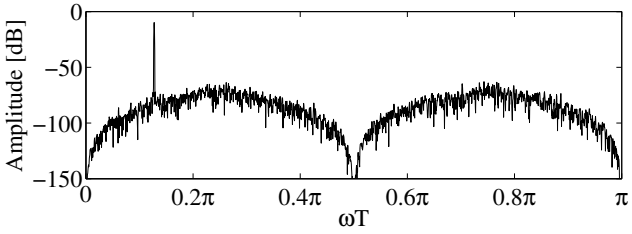
$$\mathbf{S}_2 = \mathbf{b}_2^T \mathbf{a}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{S}_3 = \mathbf{b}_3^T \mathbf{a}_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Because the sum of all $\mathbf{S}_k$-matrices is a circulant matrix, the system is alias-free and the transfer function for the system is given by (10.15) as

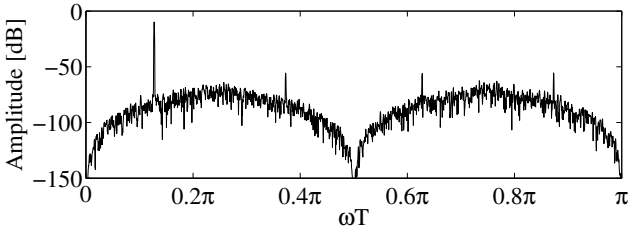
$$H_A(z) = z^{-1} s_3^{0,1} H_{3,1}(z^4) = z^{-1} \tag{10.25}$$

where $H_{3,1}(z) = 1$ is the second polyphase component in the polyphase decomposition of $H(z)$. The transfer function is thus a simple delay, and the system will digitize the complete spectrum.

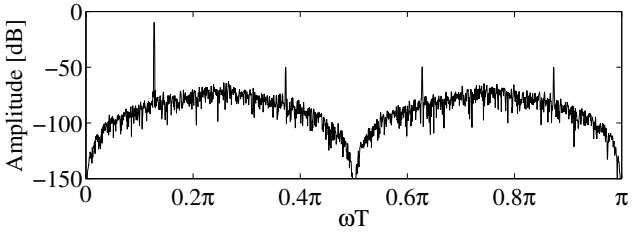
As none of the $\mathbf{S}_k$-matrices are circulant, and a circulant matrix can be formed only by summing all the matrices, the time-interleaved ADC requires matching of all channels in order to eliminate aliasing. Thus we define $C_0 = \{0, 1, 2, 3\}$, according to Sec. 10.1.2. The system has been simulated with modulator nonidealities and errors of bi-level sequences for sensitive channels, as described in Sec. 10.2. Figure 10.8(a) shows the output spectrum for the ideal case with no mismatches between channels ($\gamma_k = 1$ for all k). Applying 2% gain mismatch for one of the channels ($\gamma_0 = 0.98$, $\gamma_1 = \gamma_2 = \gamma_3 = 1$), the spectrum in Fig. 10.8(b) results, where the aliasing components can be clearly seen. In Fig. 10.8(c), the channel gains are set to one, and a 1% offset error has been added to the first modulation sequence ($\beta_0 = 0.01$, $\beta_1 = \beta_2 = \beta_3 = 0$), which results in aliasing. In Fig. 10.8(d), high-pass modulators have been used instead, and the distortions disappear, as predicted by the analysis in Sec. 10.2.3.



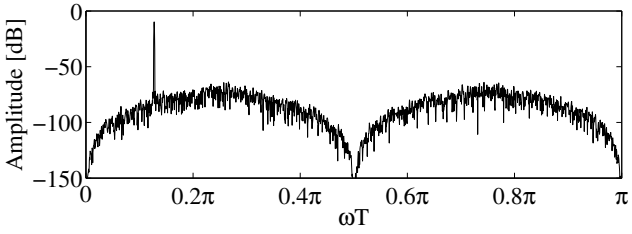
(a) Simulation using ideal system.



(b) Simulation with 2% gain mismatch in one channel.



(c) Simulation with 1% offset error in one modulation sequence.



(d) Simulation with 1% offset error in one modulation sequence using high-pass modulators instead of low-pass modulators.

Figure 10.8 *Example 1: Sensitivity of time-interleaved ADC.*

10.3.2 Hadamard-modulated ADC

Consider a non-oversampling Hadamard-modulated ADC with eight channels. In this case, every channel filter is an 8th-band filter ($H_k(z) = H(z), k = 0, \ldots, 7$) and the modulation sequences $a_k(n)$ and $b_k(n)$ are

$$\begin{aligned}
a_0(n) &= b_0(n) = 1, \quad 1, \quad 1, \quad 1, \quad 1, \quad 1, \quad 1, \quad 1, \ldots, \\
a_1(n) &= b_1(n) = 1, -1, \quad 1, -1, \quad 1, -1, \quad 1, -1, \ldots, \\
a_2(n) &= b_2(n) = 1, \quad 1, -1, -1, \quad 1, \quad 1, -1, -1, \ldots, \\
a_3(n) &= b_3(n) = 1, -1, -1, \quad 1, \quad 1, -1, -1, \quad 1, \ldots, \\
a_4(n) &= b_4(n) = 1, \quad 1, \quad 1, \quad 1, -1, -1, -1, -1, \ldots, \\
a_5(n) &= b_5(n) = 1, -1, \quad 1, -1, -1, \quad 1, -1, \quad 1, \ldots, \\
a_6(n) &= b_6(n) = 1, \quad 1, -1, -1, -1, -1, \quad 1, \quad 1, \ldots, \\
a_7(n) &= b_7(n) = 1, -1, -1, \quad 1, -1, \quad 1, \quad 1, -1, \ldots.
\end{aligned} \tag{10.26}$$

The vectors $\mathbf{a}_k$ and $\mathbf{b}_k$ become

$$\begin{aligned}
\mathbf{a}_0 = \mathbf{b}_0 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\
\mathbf{a}_1 = -\mathbf{b}_1 &= \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \\
\mathbf{a}_2 = \mathbf{b}_3 &= \begin{bmatrix} 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix} \\
\mathbf{a}_3 = -\mathbf{b}_2 &= \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \end{bmatrix} \\
\mathbf{a}_4 &= \begin{bmatrix} 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \\
\mathbf{b}_4 &= \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix} \\
\mathbf{a}_5 &= \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \end{bmatrix} \\
\mathbf{b}_5 &= \begin{bmatrix} 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \end{bmatrix} \\
\mathbf{a}_6 &= \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix} \\
\mathbf{b}_6 &= \begin{bmatrix} 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \end{bmatrix} \\
\mathbf{a}_7 &= \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \end{bmatrix} \\
\mathbf{b}_7 &= \begin{bmatrix} -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix}.
\end{aligned} \tag{10.27}$$

With $\mathbf{S}_k = \mathbf{b}_k^T \mathbf{a}_k$, the following matrices can be computed:

$$\mathbf{S}_0 = \mathbf{1}$$

$$\mathbf{S}_1 = \begin{bmatrix} -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{S}_2 + \mathbf{S}_3 = \begin{bmatrix} 0 & 2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 & 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & 0 & 2 & 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \end{bmatrix}$$

$$\mathbf{S}_4 + \mathbf{S}_5 + \mathbf{S}_6 + \mathbf{S}_7 = \begin{bmatrix} 0 & 4 & 0 & 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & -4 \\ -4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & -4 & 0 & 0 & 0 & 4 \\ 4 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \end{bmatrix}$$

It is seen that $\mathbf{S}_0$ and $\mathbf{S}_1$ are circulant matrices. Also, $\mathbf{S}_2 + \mathbf{S}_3$ is circulant. Further, the remaining matrices sum to a circulant matrix $\mathbf{S}_4 + \mathbf{S}_5 + \mathbf{S}_6 + \mathbf{S}_7$, whereas no smaller subset does. Thus, in order to eliminate aliasing, the channels are partitioned into the sets $C_0 = \{0\}$, $C_1 = \{1\}$, $C_2 = \{2, 3\}$ and $C_3 = \{4, 5, 6, 7\}$. The Hadamard-modulated ADC thus contains both insensitive channels 0 and 1, and sensitive channels $2, \ldots, 7$.

Using the model of the ideal system, the spectrum of the output signal is as shown in Fig. 10.9(a). Figure 10.9(b) shows the output spectrum for the system with 1% random gain mismatch ($\gamma_k \in [0.99, 1.01]$), where the aliasing distortions are readily seen. Matching the gains of the C_2-channels to each other (setting $\gamma_2 = \gamma_3$) and the gains of the C_3-channels to each other (setting $\gamma_4 = \gamma_5 = \gamma_6 = \gamma_7$), the spectrum in Fig. 10.9(c) results, and the distortions disappear. Although the Hadamard-modulated ADC is less sensitive than the time-interleaved ADC, the matching requirements for eight-channel systems and above are still severe.

10.3.3 Frequency-band decomposed ADC

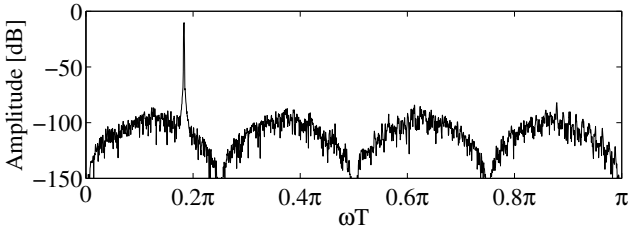
For the frequency-band decomposed ADC, the input signal is applied unmodulated to N modulators converting different frequency bands. Consider as an example a four-channel system consisting of a lowpass channel, a highpass channel, and two bandpass channels centered at $3\pi/8$ and $5\pi/8$.

As the signal is not modulated,

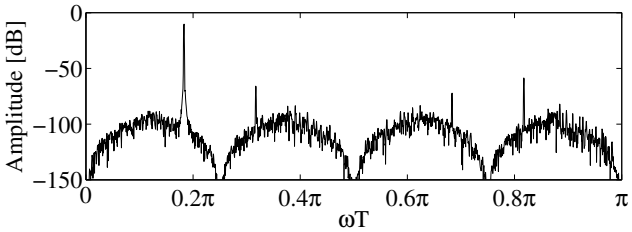
$$\mathbf{a}_k = \mathbf{b}_k = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \tag{10.28}$$

for all k, and

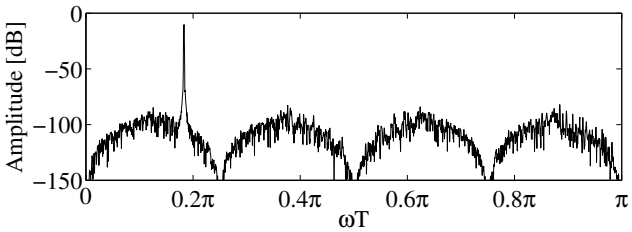
$$\mathbf{S}_k = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



(a) Simulation using ideal model.



(b) Simulation using 1% channel gain mismatch.



(c) Simulation using gain matching of sensitive channels.

Figure 10.9 *Example 2: Sensitivity of Hadamard-modulated ADC.*

for all k. As each $\mathbf{S}_k$-matrix is circulant, the system is insensitive to channel mismatches. Further, modulation sequence errors are irrelevant in this case, as the signal is not modulated. The frequency-band decomposed ADC is thus highly resistant to mismatches. Its obvious drawback, however, is the need to use bandpass modulators which are more expensive in hardware.

10.3.4 Generation of new scheme

This example demonstrates that the formulation can also be used to devise new schemes, although a general method is not presented. A three-channel parallel system using lowpass modulators is designed. The signal is assumed to be in the frequency band $-\pi/4 < \omega T < \pi/4$, and the ADC is thus an oversampled system and is described according to Sec. 10.1.3 with $L = 4$ and $M = 8$.

Using complex modulation sequences, three bands of width $\pi/4$ centered at

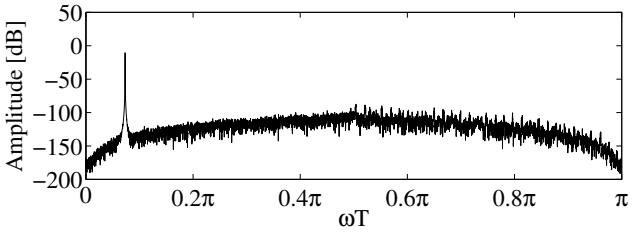


Figure 10.10 *Example 4: Sensitivity of new scheme. Simulation using 10% channel gain mismatch.*

$-\pi/4$, 0, and $\pi/4$ can be translated to baseband and converted with a lowpass ADC. These modulation sequences are $a_0(n) = 1$, $a_1(n) = \exp(j\pi n/4)$, $a_2(n) = \exp(-j\pi n/4)$ and $b_k(n) = {a_k}^*(n)$. Summing the resultant $\mathbf{S}_k$-matrices yields

$$\sum \mathbf{S}_k = \mathbf{1} + \begin{bmatrix} \sqrt{2} & 2 & \sqrt{2} & 0 & -\sqrt{2} & -2 & -\sqrt{2} & 0 \\ 0 & \sqrt{2} & 2 & \sqrt{2} & 0 & -\sqrt{2} & -2 & -\sqrt{2} \\ -\sqrt{2} & 0 & \sqrt{2} & 2 & \sqrt{2} & 0 & -\sqrt{2} & -2 \\ -2 & -\sqrt{2} & 0 & \sqrt{2} & 2 & \sqrt{2} & 0 & -\sqrt{2} \\ -\sqrt{2} & -2 & -\sqrt{2} & 0 & \sqrt{2} & 2 & \sqrt{2} & 0 \\ 0 & -\sqrt{2} & -2 & -\sqrt{2} & 0 & \sqrt{2} & 2 & \sqrt{2} \\ \sqrt{2} & 0 & -\sqrt{2} & -2 & -\sqrt{2} & 0 & \sqrt{2} & 2 \\ 2 & \sqrt{2} & 0 & -\sqrt{2} & -2 & -\sqrt{2} & 0 & \sqrt{2} \end{bmatrix} \tag{10.29}$$

Unfortunately, using complex modulation sequences is not practical. However, as the modulators and filters are identical for all channels ($H_k(z) = H(z)$ for all k), any other choice of modulation sequences resulting in the same matrix will perform the same function. Moreover, for a decimated system, relaxations may be allowed on the new modulation sequences. In this case, with decimation by four, it is sufficient to find replacing modulation sequences $\mathbf{a}'_k$ and $\mathbf{b}'_k$ such that the sum of the resulting $\mathbf{S}'_k$-matrices equal $\sum \mathbf{S}_k$ on rows 4 and 8, as $\gcd(L, M) = 4$. One such choice of modulation sequences is

$$\begin{aligned} \mathbf{a}'_0 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \mathbf{a}'_1 &= \begin{bmatrix} 1 & 1 & 0 & -1 & -1 & -1 & 0 & 1 \end{bmatrix} \\ \mathbf{a}'_2 &= \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{b}'_0 &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \\ \mathbf{b}'_1 &= \begin{bmatrix} 0 & 0 & 0 & -\sqrt{2} & 0 & 0 & 0 & \sqrt{2} \end{bmatrix} \\ \mathbf{b}'_2 &= \begin{bmatrix} 0 & 0 & 0 & (\sqrt{2}-2) & 0 & 0 & 0 & (2-\sqrt{2}) \end{bmatrix}. \end{aligned} \tag{10.30}$$

The analog modulation sequences $\mathbf{a}'_k$ can easily be implemented by switching or grounding the inputs to the modulators, whereas the nontrivial multiplications in

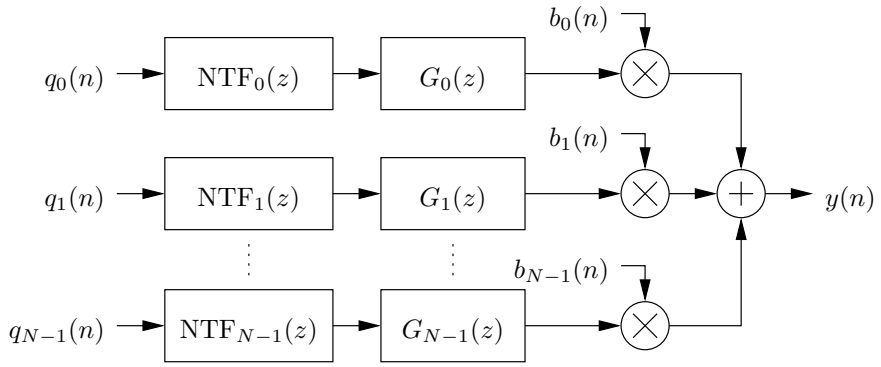


Figure 10.11 *Noise model of parallel system.*

$\mathbf{b}'_k$ can be implemented with high precision digitally. Note that

$$\sum \mathbf{b}'^{\,T}_k \mathbf{a}'_k = \mathbf{1} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & -\sqrt{2} & 0 & \sqrt{2} & 2 & \sqrt{2} & 0 & -\sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & \sqrt{2} & 0 & -\sqrt{2} & -2 & -\sqrt{2} & 0 & \sqrt{2} \end{bmatrix}, \tag{10.31}$$

which is equal to $\sum \mathbf{S}_k$ in (10.29) on rows 4 and 8. Note also that the $\mathbf{S}'_k$-matrices, given on rows 4 and 8 by

$$\begin{bmatrix} \mathbf{b}'_{0,3} \\ \mathbf{b}'_{0,7} \end{bmatrix} \mathbf{a}'_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{b}'_{1,3} \\ \mathbf{b}'_{1,7} \end{bmatrix} \mathbf{a}'_1 = \begin{bmatrix} -\sqrt{2} & -\sqrt{2} & 0 & \sqrt{2} & \sqrt{2} & \sqrt{2} & 0 & -\sqrt{2} \\ \sqrt{2} & \sqrt{2} & 0 & -\sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & \sqrt{2} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{b}'_{2,3} \\ \mathbf{b}'_{2,7} \end{bmatrix} \mathbf{a}'_2 = \begin{bmatrix} (\sqrt{2}-2) & 0 & 0 & 0 & (2-\sqrt{2}) & 0 & 0 & 0 \\ (2-\sqrt{2}) & 0 & 0 & 0 & (\sqrt{2}-2) & 0 & 0 & 0 \end{bmatrix},$$

are circulant on these rows, and thus the system is insensitive to channel mismatches. This is demonstrated in Fig. 10.10, where the channel gain mismatch is 10% and no aliasing results. However, as three levels are used in the analog modulation sequences $\mathbf{a}'_1$ and $\mathbf{a}'_2$, the system is sensitive to mismatches in the modulation sequences of these channels, as described in Sec. 10.2.

10.4 Noise model of system

The primary purpose of this work is to investigate the signal transfer characteristics of a parallel ΣΔ-system. However, a system's noise properties are also affected

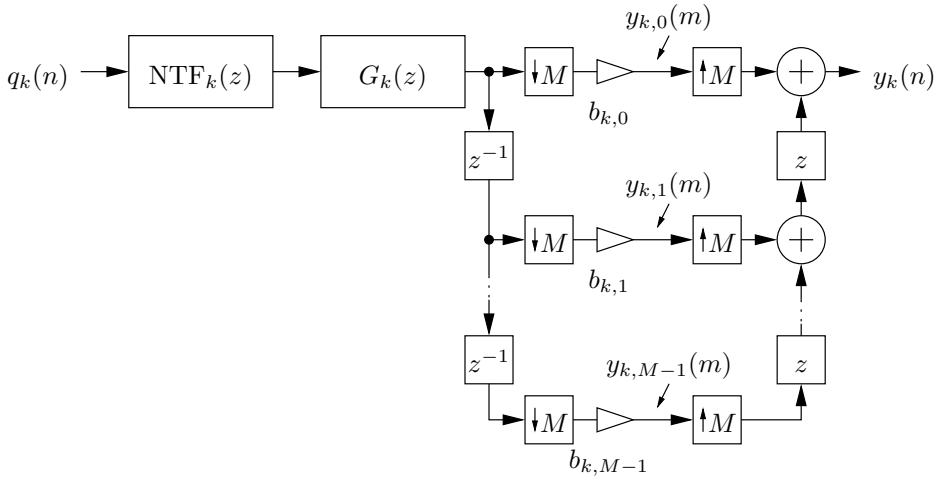


Figure 10.12 *Noise model of channel k.*

by the choice of modulation sequences, and therefore a simple noise analysis is included.

A noise model of the parallel $\Sigma\Delta$-system can be depicted as in Fig. 10.11. The quantization noise $q_k(n)$ of channel k is filtered through the noise transfer function $\mathrm{NTF}_k(z)$ and filter $G_k(z)$. The filtered noise is then modulated by the sequence $b_k(n)$. The channels are summed to form the output $y(n)$.

In order to determine the statistical properties of the output $y(n)$, channel k is modeled as in Fig. 10.12. Denoting the spectral density of the quantization noise of channel k by $R_{Q_k}(e^{j\omega})$, the spectral densities of the polyphase components $y_{k,m}$ of the channel output can be written

$$R_{y_{k,m}}(e^{j\omega}) = b_{k,m}^2 \sum_{l=0}^{M-1} \left|G_{k,l}(e^{j\omega})\right|^2 R_{Q_k}(e^{j\omega}) \tag{10.32}$$

where $G_{k,l}(z)$ are the polyphase components of the cascaded system $\mathrm{NTF}_k(z)G_k(z)$. It is seen that the noise power is scaled by the factor $b_{k,m}^2$, and it is thus of interest to keep the amplitudes of the modulation sequences low on the digital side. For example, in the generation of the new modulation scheme in Sec. 10.3.4, alternative choices of $\mathbf{a}_1$ and $\mathbf{b}_2$ would have been

$$\begin{aligned}\mathbf{a}_1 &= \begin{bmatrix}0 & 1 & 0 & -1 & 0 & -1 & 0 & 1\end{bmatrix}\\ \mathbf{b}_2 &= \begin{bmatrix}0 & 0 & 0 & -2 & 0 & 0 & 0 & 2\end{bmatrix}.\end{aligned}$$

However, in this case the noise power is larger. This shows that the smaller magnitudes of the digital modulation sequences, as in (10.30), are preferable from a noise perspective.

11

Sigma-delta ADC decimation filters

In this chapter, the design complexity requirements of decimation filters for ΣΔ-ADCs are investigated. For fast ΣΔ-ADCs with large oversampling factors, the power dissipation of the decimation filter may become a significant part of the whole ADC, and is often overlooked. The complexity of linear phase FIR filter-based decimators are considered for a wide range of oversampling ratios (OSR) and three different modulator structures. Simulation results show the SNR degradation of the ADC as a function of the transition bandwidth and the filter order.

In Sec. 11.1, needed notations are introduced and the filter design methodology is discussed. In Sec. 11.2, filter design results are provided.

This chapter has been published in [17].

11.1 Design considerations

Consider the system shown in Fig. 11.1. An analog signal $x(t)$ is sampled and quantized by a ΣΔ-modulator to a low resolution signal $u(n)$ with high sampling

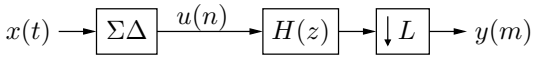
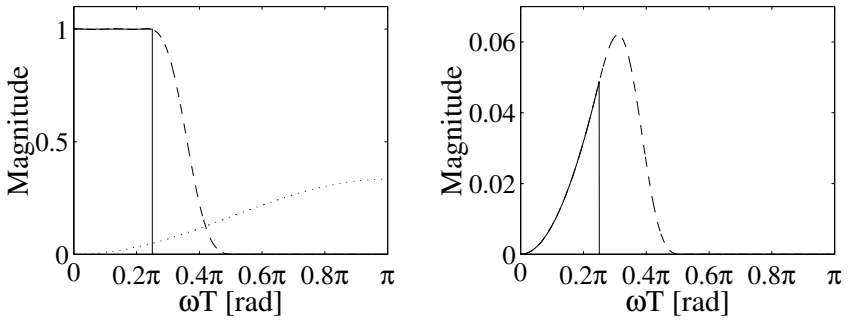


Figure 11.1 *ΣΔ-ADC with digital decimation filter*



(a) Ideal (solid line) and practical (dashed line) decimation filter, and shaped quantization noise before filtering (dotted line).

(b) Filtered quantization noise using ideal (solid line) and practical (dashed line) decimation filter.

Figure 11.2 *Output quantization noise spectral density function for a 1-bit first-order sigma-delta.*

rate. The signal is then filtered and downsampled, resulting in a signal $y(m)$ with high resolution and a low sampling rate.

11.1.1 FIR decimation filters

The filters in this chapter are FIR filters designed in the minimax sense (see Sec. 8.2). It may be argued that minimizing the energy through a least-squares design is more appropriate in order to maximize the SNR. However, in communications systems it is common that the suppression of blockers imposes minimax constraints on the filter, thus making minimax design interesting. The filters are also designed as single-stage filters, whereas they are in practice normally designed in several stages to relax the overall requirements. In particular, recursive comb decimators provide efficient filtering with few arithmetic operations. The results in this chapter do not contradict the fact that such structures are efficient, but rather concentrates on the relation between the stringency of the filter requirements and ADC parameters. As a measure of the filter requirements stringency, the filter order is used.

Ideally, a decimation filter would be used that passes through all the frequencies in the signal bandwidth, while completely suppressing all frequencies outside the signal bandwidth. Such a filter is shown by the solid line in Fig. 11.2(a). In this case, the signal is degraded only by the quantization noise in the same band as the signal. However, in practice the filter has a transition band, as exemplified by the dashed line in Fig. 11.2(a). Then, the signal quality is reduced also by the noise within the transition band. Moreover, as seen in the output power spectral density of the filtered quantization noise in Fig. 11.2(b), the noise in the transition band can consitute a considerable fraction of the total noise power.

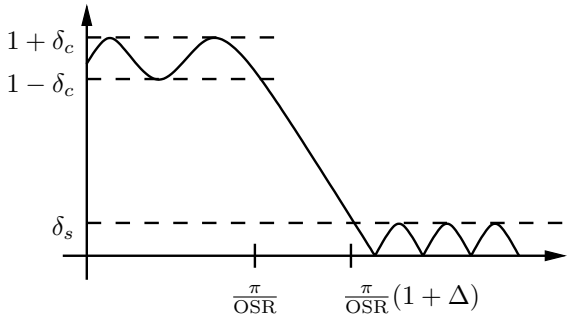


Figure 11.3 *Decimation filter specification.*

For FIR filter-based decimators, the filter complexity is to the largest extent determined by the transition bandwidth [49]. Further, a significant part of the quantization noise energy of an oversampled ΣΔ-modulator is located in this transition band, in particular when higher-order noise shaping is applied. Therefore, there exists a pronounced trade-off between decimation filter complexity and ΣΔ-modulator signal-to-noise-ratio (SNR). The work in this chapter presents investigations on how the SNR is degraded as a function of the filter transition bandwidth and filter order for various commonly used ΣΔ-modulator architectures and oversampling ratio (OSR). It is also demonstrated that, for a given filter order, there exists an optimum choice of the stopband ripple and stopband edge for equi-ripple filter solutions which minimizes the SNR degradation.

11.1.2 Decimation filter specification

The considered filters are Nth-order symmetric linear-phase FIR filters that satisfy the specification shown in Fig. 11.3 where δ_c, δ_s, $\omega_c T = \pi/\text{OSR}$, and $\omega_s T = \pi(1+\Delta)/\text{OSR}$, denote the passband ripple, stopband ripple, passband edge and stopband edge, respectively. The filters are designed in the minimax sense using the McClellan-Parks-Rabiner's (MPR) algorithm. The passband ripple is specified to be 0.01, but after each design it may be slightly smaller as there generally is a design margin. This is because the MPR algorithm only handles the ratio between the passband and stopband ripples.

The stopband edge is related to the passband edge through the relative transition bandwidth Δ that is defined as

$$\Delta = \frac{\omega_s T - \omega_c T}{\omega_c T}, \tag{11.1}$$

and shown in Fig. 11.3.

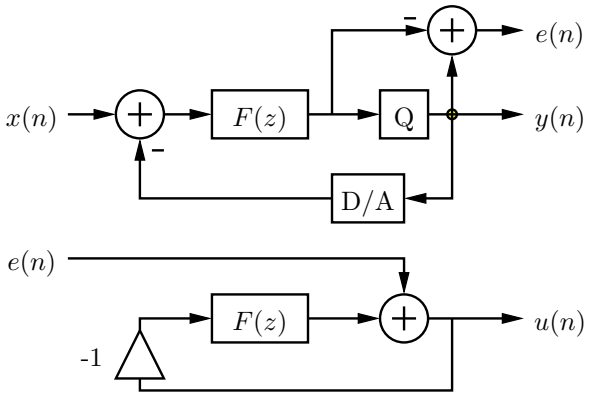


Figure 11.4 *Principle of the extraction of quantization noise from a ΣΔ-modulator and the filtering thereof through a time-domain realization of the ΣΔ-modulator noise transfer function.*

11.1.3 Signal-to-noise-ratio

Consider a ΣΔ-modulator with an input signal $x(n)$, output signal $y(n)$, and quantization error $e(n)$, as defined in Sec. 9.2. The output signal and noise power of the ΣΔ-modulator after filtering are denoted P_{yx} and P_{ye}, respectively. Assuming that the input signal $x(n)$ is bandlimited to $\omega_c T$, and the passband ripple δ_c of the filter $H(z)$ is small enough so its effect on the signal power can be neglected, P_{yx} equals the input signal power, i.e., $P_{yx} = P_x$. The contribution to the output noise power emanates from the passband, transition band, and stopband regions. The corresponding noise powers are denoted $P_{ye}^{(pb)}$, $P_{ye}^{(tb)}$, and $P_{ye}^{(sb)}$, respectively. The total noise power is thus $P_{ye} = P_{ye}^{(pb)} + P_{ye}^{(tb)} + P_{ye}^{(sb)}$, and the SNR at the output is given by

$$\mathrm{SNR} = 10 \log_{10} \frac{P_{yx}}{P_{ye}} \tag{11.2}$$

Further, the SNR degradation ΔSNR is defined as the degradation in SNR caused by using a practical filter instead of an ideal lowpass filter. Thus,

$$\Delta\mathrm{SNR} = 10 \log_{10} \frac{P_{yx}}{P_{ye}^{(pb)}} - 10 \log_{10} \frac{P_{yx}}{P_{ye}} = 10 \log_{10} \frac{P_{ye}}{P_{ye}^{(pb)}} \tag{11.3}$$

Using the linear model discussed in Sec. 9.1.3, the noise power is computed as

$$P_e = \frac{1}{2\pi} \int_{-\pi}^{\pi} \frac{Q^2}{12} \left|G(e^{j\omega T})\right|^2 \left|H(e^{j\omega T})\right|^2 \mathrm{d}(\omega T), \tag{11.4}$$

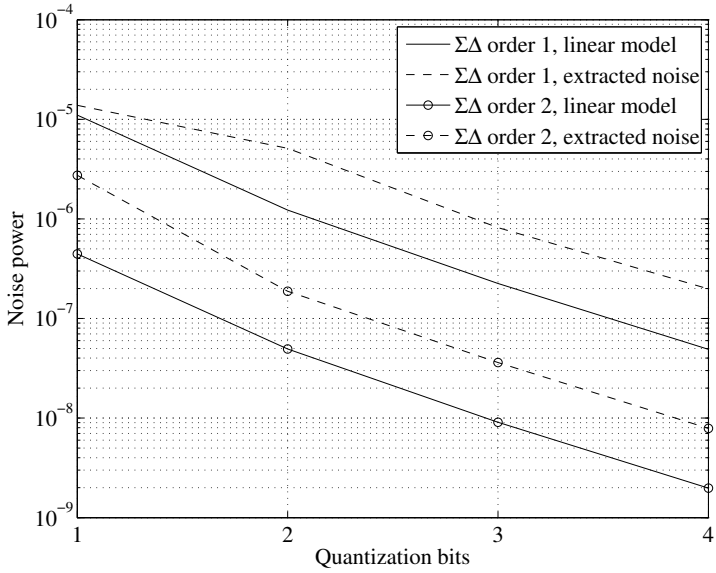
where Q is the quantization step, $G(z)$ is the noise transfer function of the ΣΔ-modulator, and $H(z)$ is the z-transform of the decimation filter. The noise power in

the different bands can be computed analogously by using the appropriate integration limits. However, the linear model tends to be less appropriate for coarse quantization steps. In particular, problems arise for one-bit quantization. Therefore, in this work, the noise power has been evaluated using the model in Fig. 11.4, where the extracted quantization error $e(n)$ has been filtered through a time-domain realization of the $\Sigma\Delta$-modulator noise transfer function $G(z)$. The obtained sequence $u(n)$ is then filtered through the decimation filter in order to get the final output noise. The noise in the different bands is then obtained from the DFT of the output noise signal. In Fig. 11.5, the noise powers of the two different cases are compared.

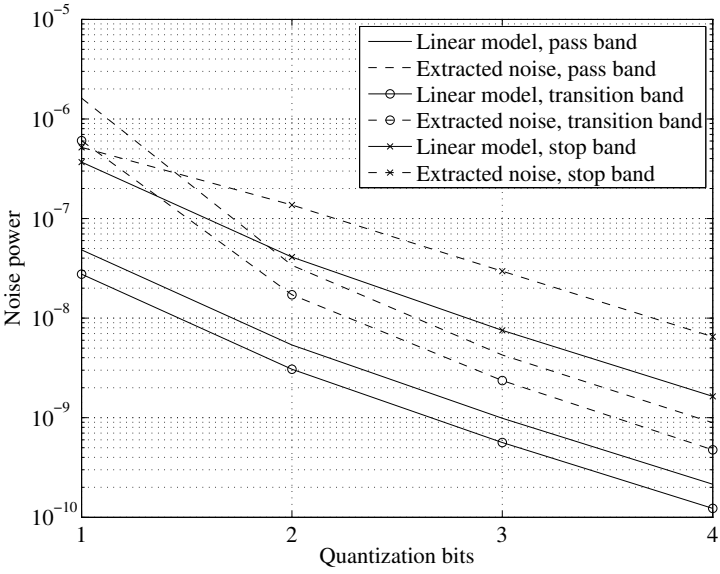
11.2 Simulation results

For the three $\Sigma\Delta$-modulator topologies shown in Figs. 9.5–9.7, two different investigations have been done. In the first, the transition bandwidth of the decimation filter has been varied for different OSR and filter orders and the corresponding SNR degradation has been calculated. The results of the investigation are plotted in Fig. 11.6. For a given transition bandwidth and filter order the stopband ripple is fixed. From the figures, it can be seen that the SNR degradation has a minimum for a certain choice of Δ.

In the second investigation, the optimal choice of the relative transition bandwidth Δ has been found for filter orders between 100 and 1000 for different OSR. The optimal Δ's and the corresponding SNR degradations for the three $\Sigma\Delta$-modulator topologies are shown in Figs. 11.7–11.9. Decreasing the transition bandwidth below the optimum causes the SNR to worsen considerably, because of rapidly decreasing stopband attenuation. Also, for large enough transition bands, increasing the filter order will yield no significant SNR improvements because essentially all the noise power is from the transition band region. It should be noted that optimal SNR may occur for a transition bandwidth several times wider than the passband width.

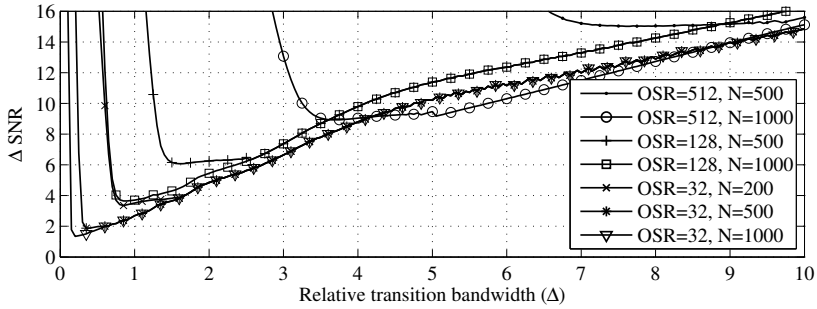


(a) ΣΔ-modulators of order one and two.

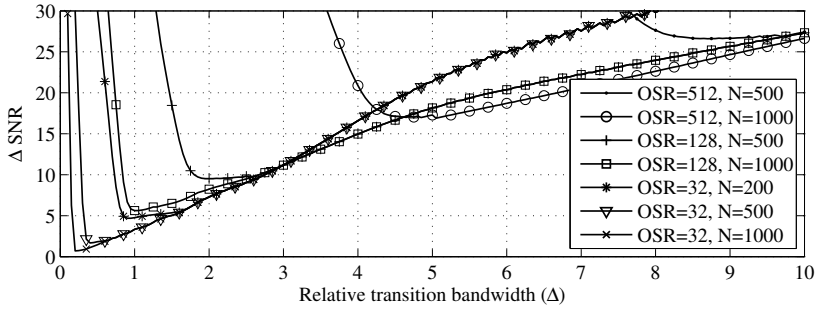


(b) Power in different bands of decimation filter, using second-order ΣΔ-modulator.

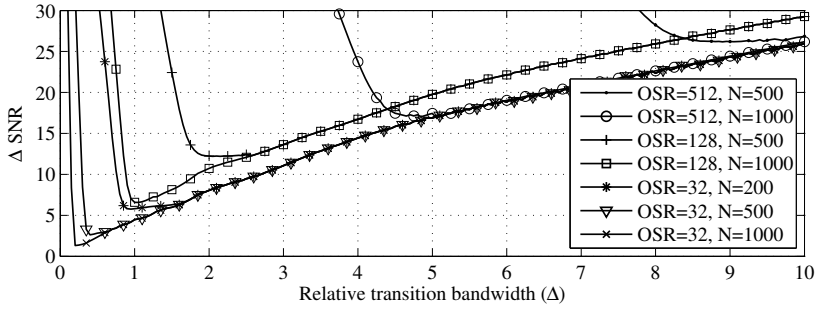
Figure 11.5 *Quantization noise power of ΣΔ-modulators after decimation filter, according to the linear noise model and the computed quantization noise. The OSR is 32.*



(a) First-order ΣΔ-modulator.

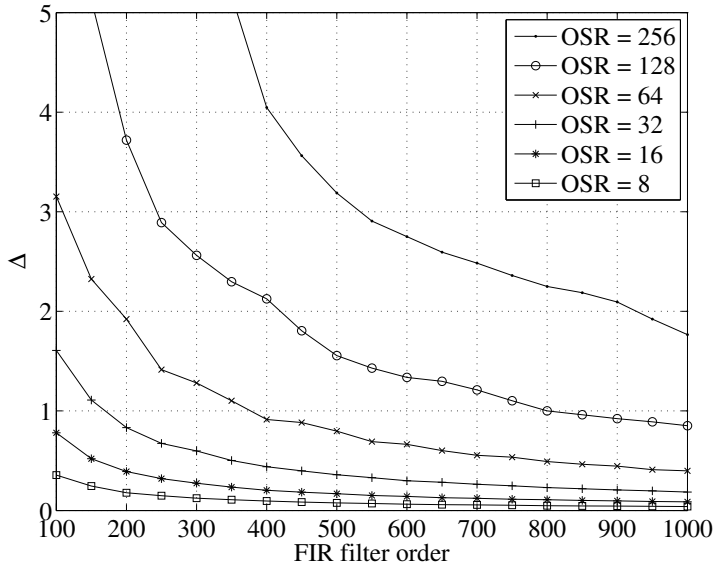


(b) Second-order ΣΔ-modulator.

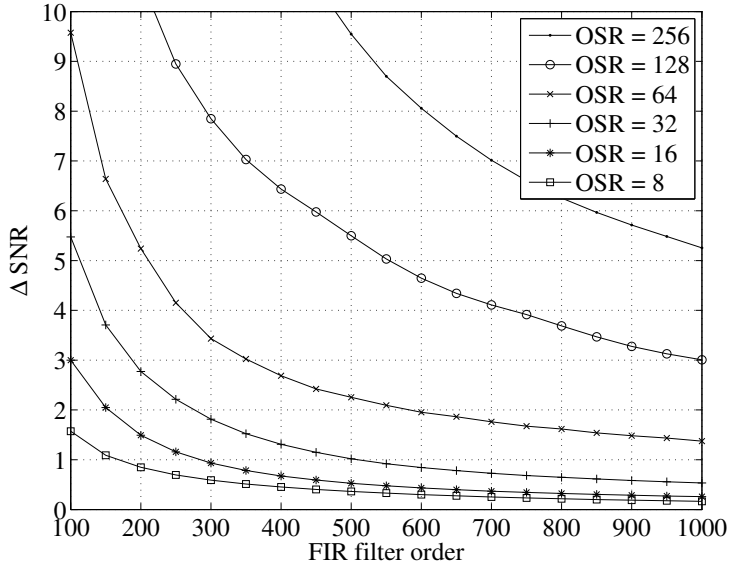


(c) MASH ΣΔ-modulator.

Figure 11.6 *SNR degradation as a function of relative transition bandwidth Δ for the different modulator structures.*

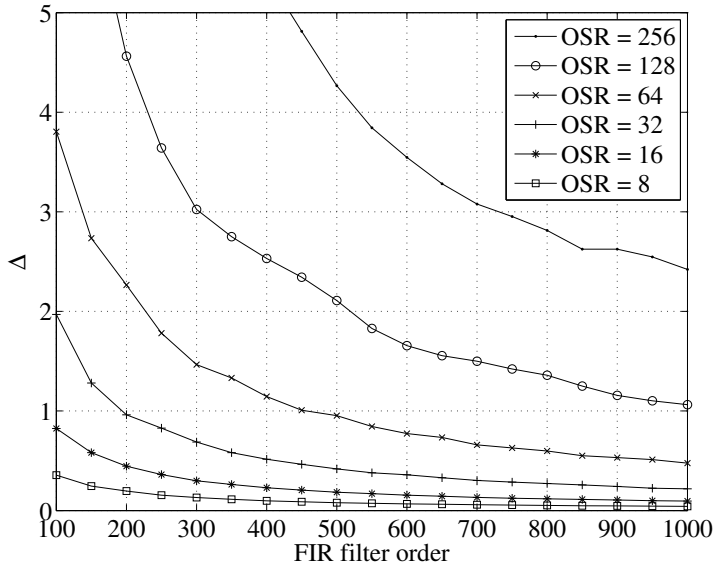


(a) SNR-optimal choice of Δ.

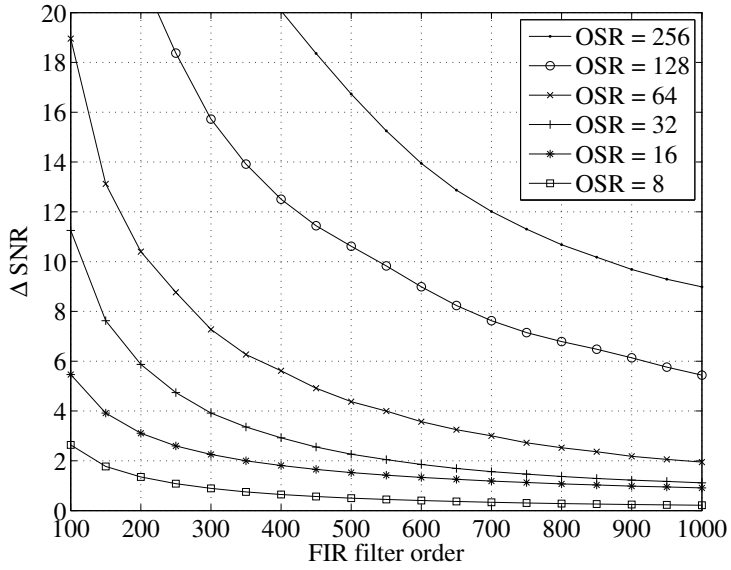


(b) Minimal SNR degradation.

Figure 11.7 *SNR-optimal choice of Δ and minimal SNR degradation as functions of the filter order for the first-order $\Sigma\Delta$-modulator.*

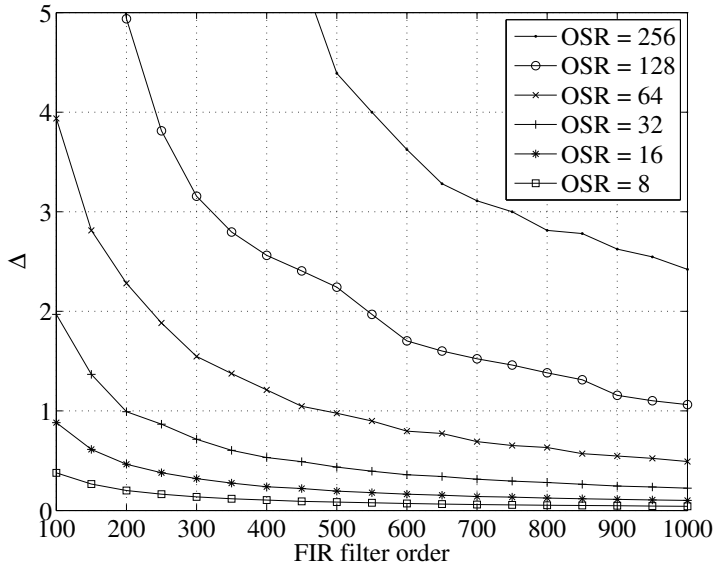


(a) SNR-optimal choice of Δ.

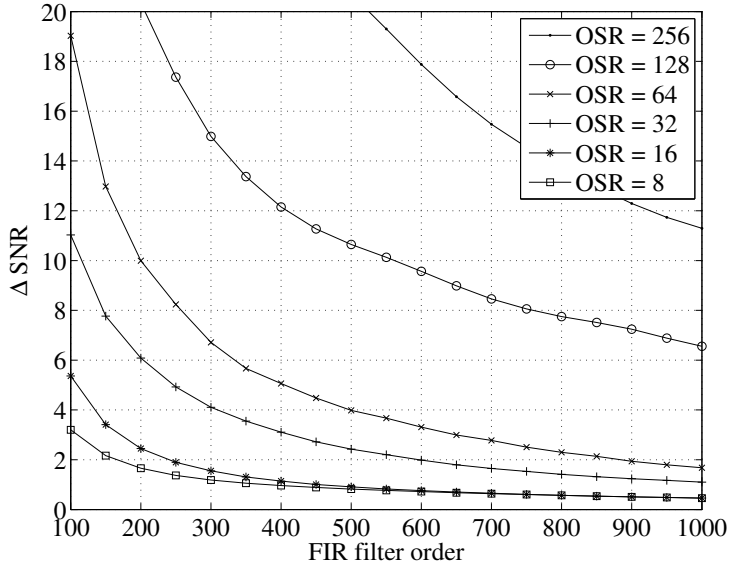


(b) Minimal SNR degradation.

Figure 11.8 *SNR-optimal choice of Δ and minimal SNR degradation as functions of the filter order for the second-order ΣΔ-modulator.*



(a) SNR-optimal choice of Δ.



(b) Minimal SNR degradation.

Figure 11.9 *SNR-optimal choice of Δ and minimal SNR degradation as functions of the filter order for the MASH $\Sigma\Delta$-modulator.*

12

High-speed digital filtering

In this chapter, the bit-level design of high-speed FIR filters is considered. For a given filter impulse response, the design process is divided into four steps: the choice of a filter architecture, the generation of partial products from the input, the identification of shared subexpressions, and the construction of an efficient pipelined reduction tree for the partial products. The focus is on the two last parts. For the identification of common subexpressions, a heuristic algorithm is proposed that identifies possible shared subexpressions before their allocation to adders in the reduction tree. In the construction of the reduction tree, a proposed bit-level optimization algorithm minimizes a weighted sum of the number of full adders, half adders and registers.

In Sec. 12.1, the investigated architectures are presented, including the direct-form structure, the transposed direct-form structure, and a new direct-form structure utilizing partial symmetry adders. In Sec. 12.2, the complexities of the architectures are estimated for different wordlengths and pipeline levels. In Sec. 12.3, the proposed algorithm for partial product redundance reduction is explained, and in Sec. 12.4.1, the proposed optimization algorithm is discussed. Results comparing the different architectures are finally in Sec. 12.5.

Secs. 12.1, 12.2 and 12.4.1 have been published in [8], whereas Sec. 12.3 has been published in [9].

12.1 FIR filter realizations

The two main architectures for FIR filters are the direct-form (DF) architecture and the transposed direct-form (TF) architecture. These were presented in Sec. 8.4, and multi-rate structures using partial product generation and reduction trees using carry-save adders were shown. The suitability of different architectures for high-speed FIR filters is investigated. For each of the considered architecture, the implementation can be partitioned into three parts: a partial product generation stage that realizes the filter coefficient multiplications, a carry-save adder (CSA) reduction tree to efficiently reduce the number of partial products, and a vector merge adder (VMA) to merge the CSA output to a non-redundant binary form.

In [1,42,45,67], different methods of generating partial products for given filters were investigated. However, here the focus is rather on the generation of an efficient pipelined reduction tree. This is done through a formulation as an integer linear programming (ILP) problem, with which a bit-level optimized reduction tree can be obtained. As cost function for the optimization algorithm, a weighted sum of the number of full adders, half adders, and registers is used. The model is similar to that presented in [59], but was not formulated as an ILP problem there.

Compared with the traditional heuristic methods: Wallace trees [97], Dadda trees [30], and Reduced Area [6] trees, the bit-level optimization yields better results for a number of reasons. The aggressive use of half adders in Wallace trees leads to fast reductions, but generally a more efficient use of half adders is possible. The Dadda structure uses half adders more restrictively only to try to maximize the opportunities to use full adders. However, only placing full adders as late as possible makes the structure unsuitable for pipelining. It is also the case that the heuristics work well for reduction trees for general multipliers but less so for other reduction trees. For example, the Reduced Area heuristic is claimed to be optimal in terms of hardware resources for general multipliers, but simulations provided in this paper show that this is not necessarily the case for general partial product trees. Moreover, the heuristics do not consider that partial products might be added at different levels in the reduction tree, which is the case for several of the architectures considered in this paper.

It should be noted that the proposed realizations can be applied to reduction trees in other applications as well. One possible application is Merged arithmetic [91], where the results of several multiplications are summed in a carry-save adder tree. Other examples are high-speed complex multipliers implemented using distributed arithmetic [5], and implementation of general functions as sums of weighted bit-products [54].

12.1.1 Architectures

The direct-form (DF1) architecture is depicted in Fig. 12.1. In this architecture, a delay chain at the input provides the algorithmic delays, and an adder tree sums the partial products generated from the taps. An interesting characteristic of the direct-form architecture is its ability to utilize the symmetry of linear-phase FIR

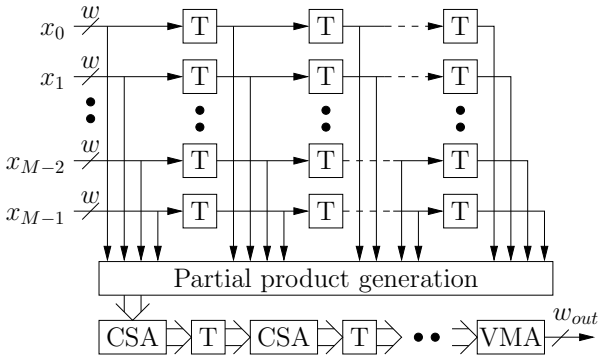


Figure 12.1 *Direct-form architecture (DF1).*

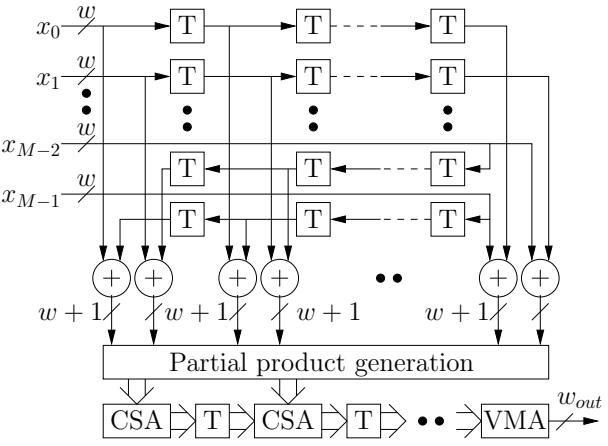


Figure 12.2 *Direct-form architecture utilizing coefficient symmetry (DF2).*

filter coefficients. This architecture is depicted in Fig. 12.2, and is denoted the DF2 architecture. The benefit of utilizing the symmetry is that the number of multiplications is halved. However, in the application of high-speed decimation filters, this feature is not necessarily efficient because of the need for short critical paths. The inability to implement the symmetry adders using carry save arithmetic leads to excessive use of registers in pipelined ripple-carry adders. This can be readily observed in the experimental results in Sec. 12.5. A solution to this problem was suggested in [33], but demonstrated in [45] to have limited efficiency. As a possible alternative solution, dividing the symmetry adders into smaller blocks of adders as depicted in Fig. 12.3 is considered. This solution is denoted partial symmetry, and will reduce the register complexity as the carry propagation chain is broken. However, partial symmetry results in slightly more partial products than

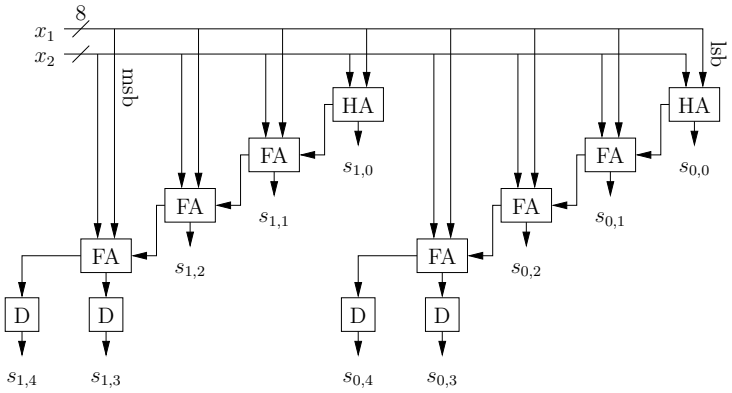


Figure 12.3 *Partial symmetry adder*

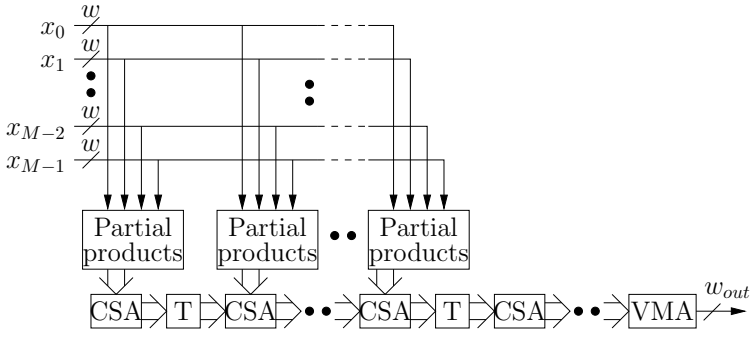


Figure 12.4 *Transposed direct-form architecture (TF).*

full symmetry does. The partial symmetry architecture is referred to as the DF3 architecture.

The TF architecture is depicted in Fig. 12.4. For high-speed decimation filters, the TF architecture may provide a more register-efficient realization, as the algorithmic delay elements are also used as pipelining registers in the summation tree. Because of the architecture's limited ability to utilize filter coefficient symmetry, however, the TF architecture may require more adders than the direct-form architecture. The reasons that symmetry cannot be utilized are two. First, the symmetric multiplier may be connected to a different input because of the polyphase decomposition. Second, instead of computing each multiplication explicitly, all generated partial products are merged in one carry-save tree. As the number of cascaded adders in each stage is restricted, there may be additional CSA stages to reduce the number of partial products before the VMA. Hence, the output of each CSA stage is not necessarily represented using at most two bits for each bit weight.

For each of the depicted architectures in Fig. 12.1, Fig. 12.2 and Fig. 12.4, the structure performs internal decimation by M. M may be 1 in order to describe a

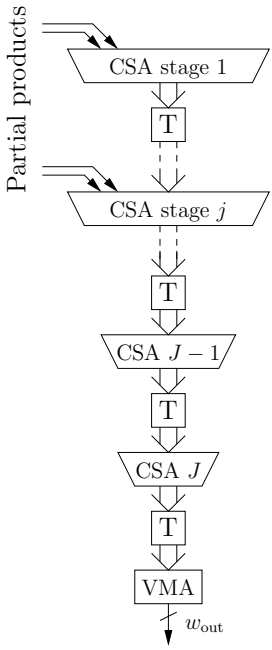


Figure 12.5 *Carry-save adder tree pipelined in J stages.*

single-rate filter. For an interpolating filter, $M = 1$ and the filter consists of several independent reduction trees for the outputs. Normally, the delays at the inputs are shared between the branches for the DF architectures. A structure may also contain both several inputs and several outputs in order to describe general multi-rate filters. However, the results in this thesis are constrained to simple decimation and interpolation filters.

For both the DF architectures and the TF architecture, the result after partial product generation is a number of partial products with different bit weights and different delays. The general structure for the summation tree is shown in Fig. 12.5, where the carry-save adder is divided into J stages. The stages are separated by pipeline registers, and inputs are accepted in all stages. Each stage has the structure shown in Fig. 12.6, allowing a maximum adder depth of K levels. Again, partial products may be added in every level. Considering the investigated architectures, for the DF1 architecture all partial products are added in the first level of the first stage. For the TF architecture partial products are added in the first level of several stages, as the pipeline registers are also used as algorithmic delays. Finally, partial products are added in several levels of the first stages for the DF2 and DF3 architectures, as the inputs are delayed by the symmetry adders.

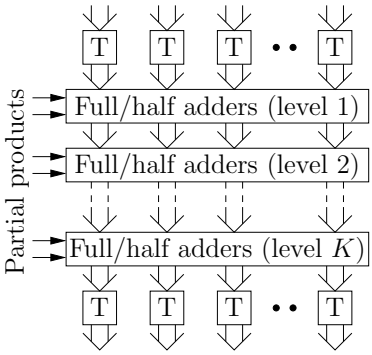


Figure 12.6 *One stage of a carry-save adder tree, with a maximum of K adders in the critical path.*

12.1.2 Partial product generation

As the multiplier coefficients are known it is not required to use general multipliers. Instead, only the partial products corresponding to non-zero coefficient bits are generated. Here, it is shown how partial products are generated using different representations of the coefficients and using both signed and unsigned input data.

An input data X with a wordlength of w_d bits can be written as

$$X = \sum_{i=0}^{w_d-1} x_i 2^i \tag{12.1}$$

for unsigned data with an input range of $0 \leq X \leq 2^{w_d} - 1$ and

$$X = -x_{w_d-1} 2^{w_d-1} + \sum_{i=0}^{w_d-2} x_i 2^i \tag{12.2}$$

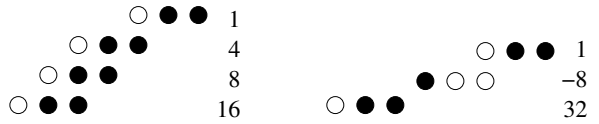
for signed (two's complement) data with an input range of $-2^{w_d-1} \leq X \leq 2^{w_d-1} - 1$, where for both (12.1) and (12.2) $x_i \in \{0, 1\}$. Note that the input data is considered to be integer instead of fractional. However, this is only to be consistent with the numbering used later on, where the bits corresponding to the smallest weight (the LSBs) have index 0.

Similarly, the w_c-bits coefficients, $h(n)$, can be written as

$$h(n) = \sum_{j=0}^{w_c-1} h_{n,j} 2^j \tag{12.3}$$

and

$$h(n) = -h_{n,w_c-1} 2^{w_c-1} + \sum_{j=0}^{w_c-2} h_{n,j} 2^j, \tag{12.4}$$



(a) Binary representation. (b) Signed-digit representation.

Figure 12.7 *Resulting partial products when multiplying a three-bits signed data with 29. White dots correspond to negated partial products.*

where $h_{n,j} \in \{0, 1\}$.

Considering the case of unsigned data and unsigned binary coefficients first, the multiplication of the input and a filter coefficient can be written

$$Xh(n) = \left(\sum_{i=0}^{w_d-1} x_i 2^i \right) \left(\sum_{j=0}^{w_c-1} h_{n,j} 2^j \right) = \sum_{j=0}^{w_c-1} \sum_{i=0}^{w_d-1} h_{n,j} x_i 2^{i+j}. \tag{12.5}$$

Now, as some of the $h_{n,j}$-bits are known to be zero, only bits corresponding to non-zero $h_{n,j}$ have to be added.

If instead a signed-digit (SD) representation of the coefficient is used, i.e., $h_{n,j} \in \{-1, 0, 1\}$, the number of non-zero $h_{n,j}$ can be decreased. A signed-digit representation with a minimum number of non-zero positions is called a minimum signed-digit (MSD) representation. In general there is no unique MSD representation, but introducing the constraint that no two adjacent positions should both be non-zero, i.e., $h_{n,j}h_{n,j+1} = 0$ will result in the canonic signed-digit (CSD) representation, which is both unique and minimum.

Using a SD representation now requires that partial products can be both added and subtracted. By enabling subtraction, signed input data and coefficients are also allowed. (12.5) will now contain partial products which may be both positive and negative. Note that $-a = \bar{a} - 1$ for $a \in \{0, 1\}$. Hence, negative partial products can be handled by inverting the partial product and adding a constant number. The constant numbers corresponding to all partial products can be merged to one constant binary number in the filter computation. In Fig. 12.7 the partial products resulting from multiplying a three-bits signed input with the coefficient 29 is illustrated for both binary and CSD representation of the coefficient. The corresponding constants to add are $-4-16-32-64 = -116$ and $-4-4-8-128 = -144$ for the binary and CSD representation, respectively. It should be noted that for the TF architecture, the output will be correct only after the first $\lceil (N-1)/M \rceil$ samples. If correct output from the first sample is required, one solution is custom initialization of each stage register.

12.2 Implementation complexity

12.2.1 Adder complexity

As only the full adders reduce the number of partial products, the required number of full adders for the carry-save summation tree can be easily calculated as the difference between the number of generated partial products and the output wordlength. For the DF1 architecture and the TF architecture, the number of generated partial products can be written $w_d(N+1)N_a$, where N is the filter order, and N_a is the average number of non-zero digits in the filter coefficients. For the DF2 architecture, the number of coefficients is halved, whereas the input wordlength is increased by one due to the symmetry adders. Thus the number of generated partial products can be written $(w_d+1)\left\lceil\frac{N+1}{2}\right\rceil N_a$. Finally, for the DF3 architecture, each symmetry adder increases its input wordlength with one, and hence the total number of partial products can be written $\left(w_d+\left\lceil\frac{w_d}{S}\right\rceil\right)\left\lceil\frac{N+1}{2}\right\rceil N_a$, assuming that partial symmetry adder groups of S bits are used. Depending on the representation used for coefficient and input data there may also be a number of constant ones to add.

In all architectures, the required number of output bits, assuming the general case with signed full-scale data and signed coefficients, can be written $w_{out} = w_d + \log_2 \sum |h(k)|$, where $h(k)$ is the impulse response. Thus the required number of full adders can be written

$$N_{\mathrm{FA,DF1}} = N_{\mathrm{FA,TF}} = w_d((N+1)N_a - 1) - \log_2 \sum |h(k)| \tag{12.6}$$

for the DF1 architecture and the TF architecture,

$$N_{\mathrm{FA,DF2}} = (w_d+1)\left\lceil\frac{N+1}{2}\right\rceil N_a - w_d - \log_2 \sum |h(k)| \tag{12.7}$$

for the DF2 architecture, and

$$N_{\mathrm{FA,DF3}} = \left(w_d+\left\lceil\frac{w_d}{S}\right\rceil\right)\left\lceil\frac{N+1}{2}\right\rceil N_a - w_d - \log_2 \sum |h(k)| \tag{12.8}$$

for the DF3 architecture. Also, the complexity of the symmetry adders for the DF2 architecture is $\left\lceil\frac{N+1}{2}\right\rceil$ w_d-bit adders, resulting in a number of adder cells equal to

$$N_{\mathrm{FA,sym,DF2}} = (w_d-1)\left\lceil\frac{N+1}{2}\right\rceil \tag{12.9}$$

and

$$N_{\mathrm{HA,sym,DF2}} = \left\lceil\frac{N+1}{2}\right\rceil. \tag{12.10}$$

For the DF3 architecture the number of partial symmetry adders is $w_d/S\left\lceil\frac{N+1}{2}\right\rceil$ S-bit adders, resulting in a number of adder cells equal to

$$N_{\mathrm{FA,sym,DF3}} = \left(w_d-\left\lceil\frac{w_d}{S}\right\rceil\right)\left\lceil\frac{N+1}{2}\right\rceil \tag{12.11}$$

and

$$N_{\mathrm{HA,sym,DF3}} = \left\lceil \frac{w_d}{S} \right\rceil \left\lceil \frac{N+1}{2} \right\rceil. \tag{12.12}$$

Using these equations, the total required number of adders for the DF architectures can be calculated. It should be noted, however, that the equations (12.6)–(12.8) do not take into account the half adders that are usually needed to rearrange the partial products, but nevertheless an approximate condition can be determined for when the DF2 architecture results in a structure with smaller adder complexity compared with DF1. This condition is

$$N_{\mathrm{FA,DF1}} > N_{\mathrm{FA,DF2}} + N_{\mathrm{FA,sym,DF2}} + N_{\mathrm{HA,sym,DF2}}, \tag{12.13}$$

which can be approximated to

$$w_d > \frac{N_a}{N_a - 1}, \tag{12.14}$$

if the costs of half and full adders are considered equal. However, as the half adders of the CSA trees have been ignored, the condition should be considered as a guideline rather than as a strict rule. In the investigated application where, typically, both w_d and N_a are low, utilizing the coefficient symmetry often does not lead to reduced adder complexity.

12.2.2 Register complexity

Regarding the register complexity, it is possible to find expressions that are asymptotically valid. However, for the considered applications these expressions convey little information, and expressions that are valid for low filter orders and short wordlengths are difficult to find. Thus, the register complexities due to algorithmic delays are calculated here, whereas those due to pipelining of the adder trees are determined experimentally.

For the DF architectures, the algorithmic delays are applied at the input, and the register complexity due to these can be written

$$N_{\mathrm{R,DF1}} = w_d N. \tag{12.15}$$

If the symmetry of the coefficients is utilized, the implementation carries an additional complexity due to pipelining of the symmetry adders. The way the pipelining is done is shown in Fig. 12.8. In addition to the registers needed to restrict the length of the critical path, registers are also placed at the outputs of the full adders just before the cuts. The reason for this is the definition of the reduction trees in Sec. 12.4.1, which does not accept inputs just before pipeline cuts. If an n-bit ripple-carry adder with a maximum of m adders in the critical path is considered, the required number of pipeline registers can be written

$$N_{\mathrm{R,RC}}(n,m) = \sum_{k=1}^{\lfloor n/m \rfloor} 2(n - mk + 1). \tag{12.16}$$

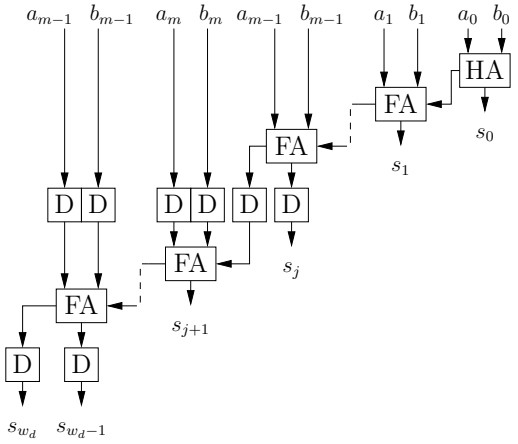


Figure 12.8 *Pipelined ripple carry adder.*

The register complexity of the DF2 architecture can then be written

$$N_{\mathrm{R,DF2}} = w_d N + \left\lceil \frac{N+1}{2} \right\rceil N_{\mathrm{R,RC}}(w_d, K), \tag{12.17}$$

where K is the maximum allowed number of adders in cascade. For the partial symmetry case, the number of registers is smaller (for $S < w_d$). The number of registers, assuming $S = nK$ for an integer n, is

$$N_{\mathrm{R,DF3}} = w_d N + \left\lceil \frac{N+1}{2} \right\rceil \left(\left\lfloor \frac{w_d}{S} \right\rfloor N_{\mathrm{R,RC}}(S, K) + N_{\mathrm{R,RC}}(w_d \bmod S, K) \right). \tag{12.18}$$

For (12.17) and (12.18) it is assumed that no sharing of registers between the algorithmic delays and symmetry adder pipeline registers has been performed. If sharing is considered, the register complexity of the DF2 architecture can be written

$$N_{\mathrm{R,DF2}} = w_d N + \left\lfloor \frac{w_d}{K} \right\rfloor (N+1) + \sum_{m=0}^{M-1} \sum_{i=0}^{\lceil \frac{N+1-m}{M} \rceil - 1} \sum_{k=1+i}^{\lfloor \frac{w_d}{K} \rfloor} (w_d - Kk). \tag{12.19}$$

If, for simplicity, $w_d = jS$ for an integer j is assumed, the register complexity of the DF3 architecture can be written

$$N_{\mathrm{R,DF3}} = w_d N + jn(N+1) + j \sum_{m=0}^{M-1} \sum_{i=0}^{\lceil \frac{N+1-m}{M} \rceil - 1} \sum_{k=1+i}^{n} (S - Kk). \tag{12.20}$$

For the TF architecture, the algorithmic delays are merged with the pipeline registers, and all registers are in the adder tree.

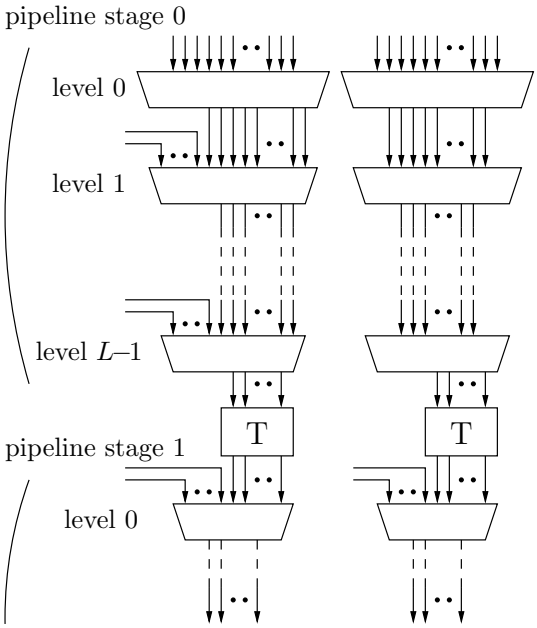


Figure 12.9 *Structure of original reduction tree.*

12.3 Partial product redundancy reduction

Consider the general structure of the original reduction tree as shown in Figs. 12.5 and 12.6. For clarity, the structure is expanded in Fig. 12.9. Let a subexpression denote three partial products with the same weight, on the same stage and level in the same reduction tree. The sharing is then based on identifying common subexpressions occurring in several places in the reduction trees, i.e., several occasions of the same set of three partial products. Thus the sharing identification is performed before the assignment of partial products to adders. Subexpressions can be shared in the same tree or between trees, and in the same stage or between stages. However, in order not to affect the delay of the intermediate partial products, subexpressions can not be shared between different levels. For each subexpression to be shared, the occurrences of the partial products are removed from the reduction trees, an adder for the subexpression is introduced, and partial products corresponding to the sum and carry outputs are added on the level below each subexpression occurrence. The resulting reduction tree structure is shown in Fig. 12.10.

Several different reduction trees can share the same full adder. However, there can not be any full adder that have inputs from more than one adder tree. Hence, the search for possible assignments need to be done inside each adder tree only, but to determine which assignment is most beneficial one should consider all adder trees. For the first level of the first stage all partial products are from the partial

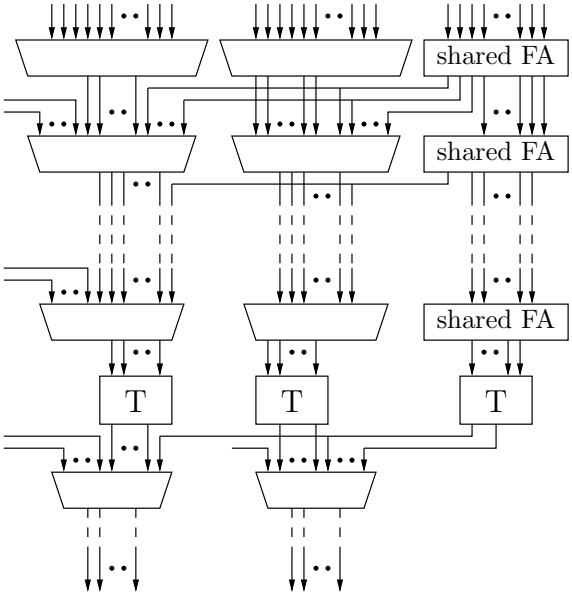


Figure 12.10 *Structure of reduction tree with redundancy reduction.*

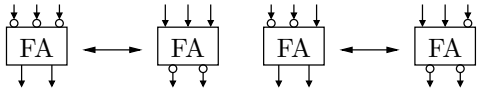


Figure 12.11 *Equivalence transforms for sign normalization of subexpressions.*

product generation stage. For latter levels there may still be redundancy as the same partial product result may end up in several columns. Considering the DF and TF architectures, adder sharing can not be expected to consistently yield better results with one of the architectures than the other. In the DF architecture, all partial products are added in the first level, increasing the number of possible subexpressions that can be considered for sharing. On the other hand, the delay chain at the input also increases the number of different partial products and decreasing the probability that the same partial products occur in several columns.

If signed data and/or coefficients are used, some of the partial products may be inverted. However, it is possible to apply transformations to increase the possible sharing by propagating inversions from the inputs to the outputs. This is illustrated in Fig. 12.11.

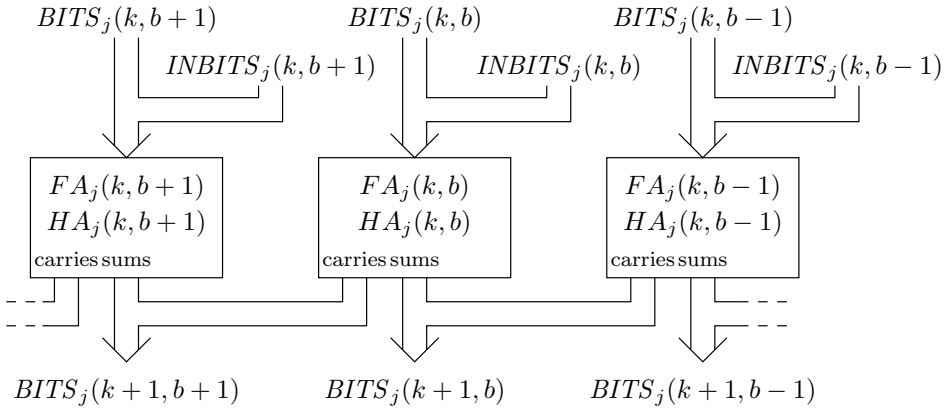


Figure 12.12 *Relations between the BITS variables in the ILP problem.*

12.3.1 Proposed algorithm

The goal of the proposed algorithm is to identify subexpressions that occur more than once. It can be described as follows:

1. Represent the filter coefficients in a suitable representation and generate one or more adder trees to be computed.
2. For each column in each level in each reduction tree, find the possible subexpressions, normalized in terms of possible inversion.
3. Determine the maximum frequency subexpression. In case of a tie, pick one at random.
4. If the chosen subexpression has only one occurrence, finish.
5. Assign the subexpression to a full adder, remove the corresponding entries from the adder trees, and add the sum and carry outputs as partial products in the corresponding columns at the level below each subexpression occurrence.
6. Go to 2.

12.4 ILP optimization

12.4.1 ILP problem formulation

Denote the stage height, i.e., the maximum number of cascaded adders, by K, as in Fig. 12.6. Denote also the number of stages by J, as in Fig. 12.5. Furthermore, denote the output wordlength by w_{out}, and the number of input partial products in each stage j, level k and bit position b by $INBITS_j(k, b)$, $k \in \{0, 1, 2, \ldots, K-1\}$,

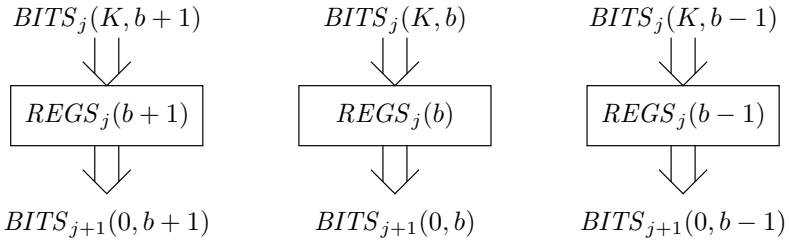


Figure 12.13 *Relations between the stages in the ILP problem.*

$b \in \{0, 1, 2, \ldots, w_{out} - 1\}$. As variables for the ILP problem, the number of full adders $FA_j(k, b)$ and half adders $HA_j(k, b)$ are used, with the same parameter bounds as $INBITS$. The resulting number of partial products in each level is denoted $BITS_j(k, b)$, and is defined $BITS_0(0, b) = INBITS_0(0, b)$ for the first level of the first stage. As each full adder reduces three partial products to one of the same weight and one of the next higher weight, and each half adder converts two partial products to one of the same weight and one of the next higher, the resulting number of partial products in each following level can be written

$$\begin{aligned} BITS_j(k+1, b) = {} & BITS_j(k, b) + INBITS_j(k, b) - 2FA_j(k, b) - HA_j(k, b) \\ & + FA_j(k, b-1) + HA_j(k, b-1), \end{aligned} \tag{12.21}$$

for $k \in \{0, 1, 2, \ldots, K - 1\}$, $b \in \{0, 1, 2, \ldots, w_{out} - 1\}$, and with variables equal to zero for out-of-bounds arguments. The relations between the $BITS$ variables are depicted in Fig. 12.12. Connections between the stages are defined by

$$BITS_{j+1}(0, b) = BITS_j(K, b). \tag{12.22}$$

Variables $REGS_j(b)$ denoting the number of pipeline registers for each stage are defined by

$$REGS_j(b) = BITS_j(K, b). \tag{12.23}$$

Thus, registers are added for all signals between the stages, as shown in Fig. 12.13. The number of adders in each level is limited by the constraint

$$3FA_j(k, b) + 2HA_j(k, b) \leq BITS_j(k, b) + INBITS_j(k, b), \tag{12.24}$$

as the number of adder inputs can not exceed the number of partial products, for each level k and bit position b. Also, in order to utilize a VMA to sum the output, the number of output bits from the last stage is limited to 2 by the condition

$$BITS_{J-1}(K, b) + INBITS_{J-1}(K, b) \leq 2, \tag{12.25}$$

for $b \in \{0, 1, 2, \ldots, w_{out} - 1\}$. Costs are defined for full adders, half adders and registers as C_{FA}, C_{HA}, and C_{REG}, respectively, and the filter structure is optimized

by minimizing the sum

$$C = C_{\mathrm{FA}} \sum_{j=0}^{J-1} \sum_{k,b} FA_j(k,b) + C_{\mathrm{HA}} \sum_{j=0}^{J-1} \sum_{k,b} HA_j(k,b) + C_{\mathrm{REG}} \sum_{j=0}^{J-1} \sum_{b} REGS_j(b) \tag{12.26}$$

The optimization problem as specified by (12.21)–(12.26) does not consider the length of the VMA. However, it may be possible to significantly reduce the length by introducing half adders to the least significant bits. The optimization problem can be modified to achieve a shorter VMA by adding a constraint to limit the number of output partial products to one for a number m of the least significant bits. This can be done by the constraint

$$BITS_{J-1}(K,b) + INBITS_{J-1}(K,b) \leq 1, \tag{12.27}$$

for $b \in \{0,1,2,\ldots,m-1\}$.

The problem complexity can be significantly reduced by the addition of additional constraints. In particular, there will never be a reason, in terms of efficient reduction of the number of partial products, not to insert a full adder where at least three partial products are available. Hence, the number of full adders in a given position can be defined based on the number of partial products available as

$$FA_k(l,b) = \left\lfloor \frac{BITS_k(l,b) + INBITS_k(l,b)}{3} \right\rfloor, \tag{12.28}$$

which can be formulated as two linear constraints for each variable.

It should be noted that the optimization problem, as formulated, is independent of the coefficient representation, i.e., binary as well as any signed-digit representation may be used. However, if signed digits are used, either if the data is signed or if the coefficient contains negative digits, a constant term must be added to the sum, as discussed in Sec. 12.1.2. As the placement of the term in the tree is arbitrary, the problem can be modified to insert the bits where they fit well. How to formulate the optimization problem to accomodate for the constant term is explained in Sec. 12.4.6. In Sec. 12.4.2 to 12.4.5, the presented architectures are formulated as initial conditions for the optimization problem. In these formulations, the coefficient digits $h_{n,j}$ are defined as in (12.3) or (12.4), with $h_{n,j} \in \{0,1\}$ for binary coefficients and $h_{n,j} \in \{-1,0,1\}$ for signed-digit coefficients.

12.4.2 DF1 architecture

For the DF1 architecture, all partial products are inserted in the first adder stage. The sum of the partial products is $\sum_{n=0}^{N} \sum_{j=0}^{w_d-1} \sum_{i=0}^{w_c-1} |h_{n,i}|\, 2^{i+j}$. Substituting $b = i + j$ and rearranging the sums allows the number of bitproducts of weight b to be written

$$INBITS_0(0,b) = \sum_{n=0}^{N} \sum_{j=0}^{w_d-1} |h_{n,b-j}|\,. \tag{12.29}$$

12.4.3 DF2 architecture

If the direct-form architecture is modified to utilize coefficient symmetry, the symmetry adders will add additional delay. Thus, the partial products involving bit 0 (the LSB) of the data are added in level 1, the partial products involving bit 1 of the data are added in level 2, and so on. Generally, the number of initial partial products in stage j and level k can be written

$$INBITS_j(k,b) = \sum_{n=0}^{(N+1)/2} |h_{n,b-Kj-k+1}| \tag{12.30}$$

for $1 \leq Kj + k \leq w_d - 1$, and

$$INBITS_j(k,b) = \sum_{n=0}^{(N+1)/2} (|h_{n,b-Kj-k+1}| + |h_{n,b-Kj-k}|) \tag{12.31}$$

for $Kj + k = w_d$.

12.4.4 DF3 architecture

For the partial symmetry case, the contributions of the different adders are separated. Assuming that a symmetry width of S adders is used, the partial products can be split into $\lceil w_d/S \rceil$ bins, where the first contains partial products from the S least significant data bits, the next bin contains partial products from the next S least significant data bits, and so on. Denoting the contribution from bin m by $B_j^m(k,b)$, the contributions for $m = 0, 1, 2, \ldots, \lceil w_d/S \rceil - 1$ can be written

$$B_j^m(k,b) = \sum_{n=0}^{(N+1)/2} |h_{n,b-Kj-k-mS+1}| \tag{12.32}$$

for $1 \leq Kj + k \leq w_d - 1$, and

$$B_j^m(k,b) = \sum_{n=0}^{(N+1)/2} (|h_{n,b-Kj-k-mS+1}| + |h_{n,b-Kj-k-mS}|) \tag{12.33}$$

for $Kj + k = w_d$. For $m = \lceil w_d/S \rceil$, the contribution can be written

$$B_j^m(k,b) = \sum_{n=0}^{(N+1)/2} |h_{n,b-Kj-k-mS+1}| \tag{12.34}$$

for $1 \leq Kj + k \leq (w_d \bmod S) - 1$, and

$$B_j^m(k,b) = \sum_{n=0}^{(N+1)/2} (|h_{n,b-Kj-k-mS+1}| + |h_{n,b-Kj-k-mS}|) \tag{12.35}$$

for $Kj + k = w_d \bmod S$. Finally, the combined contribution is the sum of the partial symmetry adder contributions

$$INBITS_j(k,b) = \sum_{m=0}^{\lceil w_d/S \rceil} B_j^m(k,b) \tag{12.36}$$

12.4.5 TF architecture

Denoting the polyphase factor by M, for the TF architecture the first M filter coefficient will be inserted in the last adder stage, the next M coefficients in the stage before, and so on. Thus, the number of initial partial products can be written

$$INBITS_{J-j-1}(0,b) = \sum_{n=Mj}^{M(j+1)-1} \sum_{t=0}^{w_d-1} |h_{n,b-t}| \,. \tag{12.37}$$

12.4.6 Constant term placement

If either the coefficient or the data contains digits with a negative sign, a constant compensation term must be added to the carry-save tree. However, these bits may be placed in an arbitrary stage, and in this section it is explained how the problem may be modified to place the bits optimally in terms of hardware resources. Define the constant, in two's complement representation, as

$$C = -c_{w_{out}-1} 2^{w_{out}-1} + \sum_{b=0}^{w_{out}-2} c_b 2^b, \tag{12.38}$$

where $c_b \in \{0,1\}$. Then define the ILP variables $CBITS_j(b) \in \{0,1\}$ for $j \in \{0,1,2,\ldots,J-1\}$, $b \in \{0,1,2,\ldots,w_{out}-1\}$, and add the constraint

$$\sum_{j=0}^{J-1} CBITS_j(b) = c_b \tag{12.39}$$

for $b \in \{0,1,2,\ldots,w_{out}-1\}$. Redefine (12.22) to

$$BITS_{j+1}(0,b) = BITS_j(K,b) + CBITS_{j+1}(b) \tag{12.40}$$

in order to add the constant bits to the carry-save tree.

12.5 Results

In this section, the different architectures are compared, and the choice of coefficient representation is investigated. For the energy and area estimations, a VHDL generator has been used to generate synthesizable VHDL code. The complete software package with ILP problem and VHDL code generator is available at [117].

12.5.1 Architecture comparison

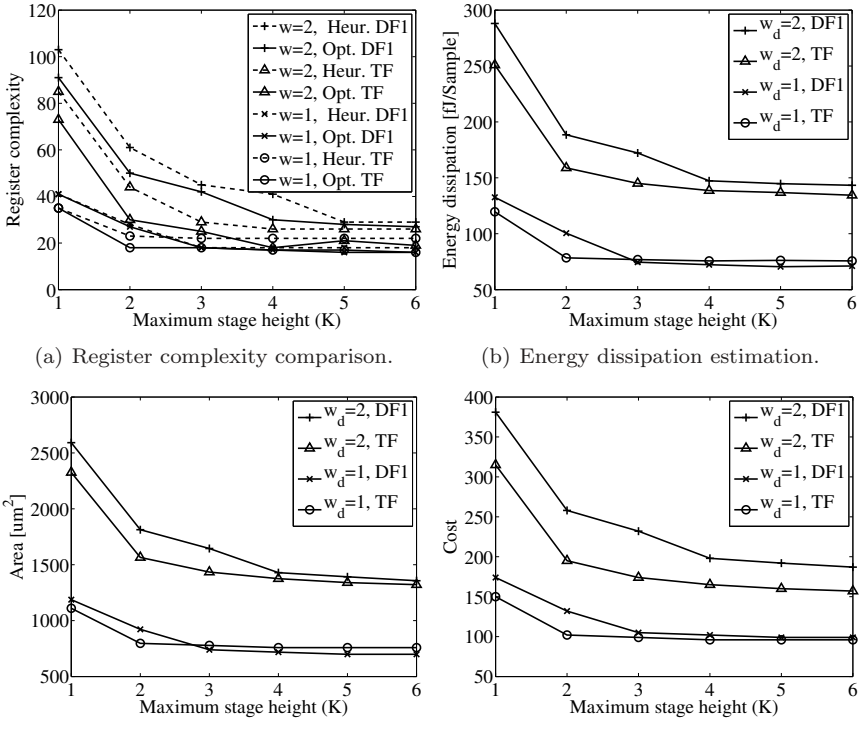
Two filters have been used to evaluate the optimization algorithm, and the relative performance of the architectures. The filters are based on 4-tap and 16-tap moving average filters, $M = 4$ and $M = 16$, respectively. Both filters consist of three cascaded filters ($L = 3$). In all simulations, the numbers of full adders correspond to those given by (12.6) and (12.7), and the number of registers given by (12.15) and (12.17) were added to the optimized result for the DF1 and DF2 architectures, respectively. The filters were optimized using the ILP problem solver SCIP [2] with the costs $C_{\mathrm{FA}} = 3$, $C_{\mathrm{HA}} = 2$, and $C_{\mathrm{REG}} = 3$. Even though the area of a full adder is roughly twice that of a half adder, it was chosen to increase the half adder cost slightly as the routing associated to one full adder is likely less than that of two half adders.

The optimized filters have been compared with filters obtained using the Reduced Area [6] heuristic. The Reduced Area heuristic is claimed to minimize the number of registers when used in a pipelined multiplier reduction tree. However, it is interesting to note that this is in general not true for the bitproduct matrices resulting from filters implemented with carry-save adder trees. Especially for the TF architecture, the bit-level optimized adder trees may result in significantly reduced register usage, while also using fewer half adders.

Figure 12.14(a) shows the impact of the pipeline factor on the first filter with short wordlengths. For the 4-tap filter, $N_a = 1.8$, and according to (12.14) utilizing the coefficient symmetry does not lead to reduced arithmetic complexity for $w_d < 2.25$, and the DF2 architecture has thus not been included. Also, all filters used six half adders. It can be seen that the bit-level optimized filters use significantly less registers, especially for heavily pipelined implementations. It can also be seen that the TF architecture has a lower register complexity except for implementations with large stage height and one bit input.

In Fig. 12.14(b) and Fig. 12.14(c), respectively, the energy dissipation and active cell area (excluding routing) are shown. The area and energy measures are based on a 90 nm standard cell library using Synopsys Design Compiler. The energy estimations are obtained using a zero-delay model and assuming uncorrelated input data. In the considered application, using a zero-delay model is expected to yield relevant results as the amount of glitches is small due to the short critical paths. Also, as decimation filter for a sigma-delta ADC the assumption of uncorrelated input data is considered to be relevant.

In Fig. 12.14(d), the total cost of the optimized filters are shown. These include additional logic and arithmetic such as the algorithmic delays for the DF1 architecture and the adders used in the ripple-carry VMA, which are not considered in the optimization. By comparing Fig. 12.14(d) with Figs. 12.14(b) and 12.14(c), it can be concluded that the used cost function is a relevant measure for optimizing both energy dissipation and cell area. Whereas energy dissipation and cell area in general do not have a strong correlation, they can be expected to correlate well when the amount of glitches is small and uncorrelated input data is used. Thus, in the rest of this paper, only complexity results and energy dissipation results will



(a) Register complexity comparison.

(b) Energy dissipation estimation.

(c) Synthesized cell area.

(d) Optimized cost.

Figure 12.14 *Optimization and synthesis results of FIR CIC filters using the TF and DF1 architectures with $M = 4$ and $L = 3$.*

be presented as cell area and cost are similar to energy dissipation.

In Fig. 12.15, the implementation complexity of the 16-tap filter is shown, using one bit input data. It is apparent that the bit-level optimized filter for the TF architecture offers a lower register complexity, while also reducing the number of half adders.

Figures 12.16(a), 12.16(b) and 12.16(c) show the full adder, half adder and register complexity, respectively, for increasing input wordlength w_d for the 4-tap filter. The stage height is $K = 2$. For the 4-tap filter, $N_a = 1.8$, and according to (12.14) the DF2 architecture has a lower arithmetic complexity for $w_d > 2$. However, even for $w_d = 6$, the reduction in arithmetic complexity is small compared to the increase in number of registers, as seen in Fig. 12.16(c). For the DF3 architecture, a symmetry adder group size of $S = 2$ was used, thus reducing the number of generated partial products by 25%. However, the register complexity is still close to the DF2 architecture with full symmetry, as shown in Fig. 12.16(c). Energy dissipation estimations for the optimized filters are shown in Fig. 12.16(d).

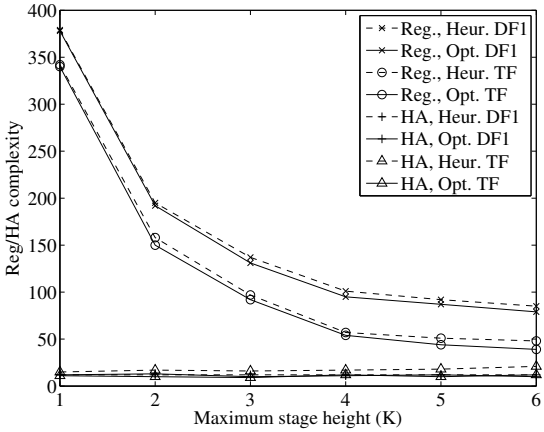


Figure 12.15 *Register/HA complexity of bit-level optimized FIR CIC filters with $M = 16$, $L = 3$, and $w_d = 1$. The number of half adders is 12 or 13 for all simulations.*

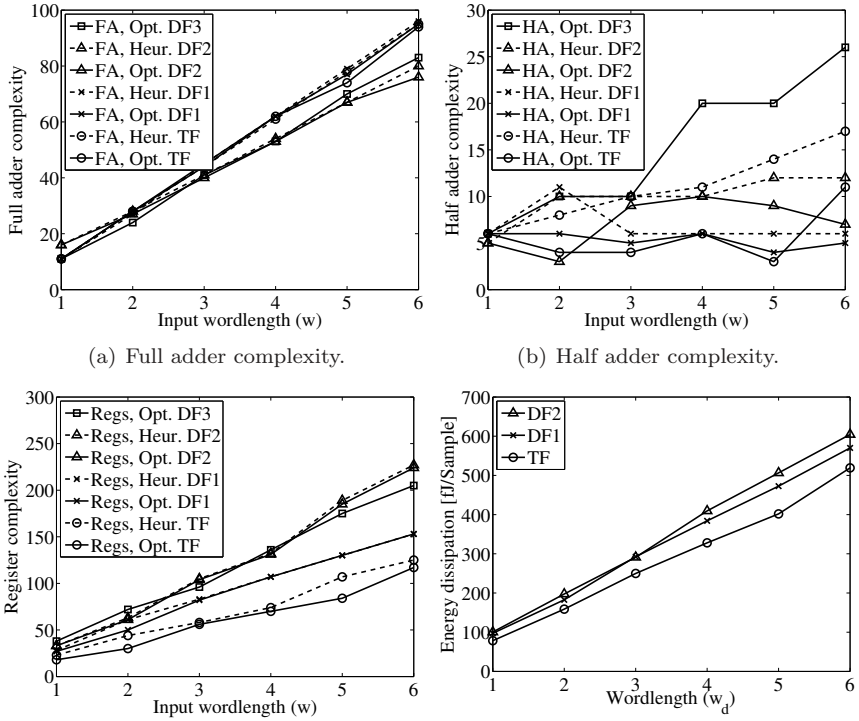
12.5.2 Coefficient representation

Different coefficient representations have been investigated for two filters shown in Fig. 12.17. Using signed-digit coefficients yields a small decrease in energy dissipation. That the gain is not larger is because the additional constant vector is relatively large compared with the number of bit-products for such small filters. Also, the simulation has not taken into account the fact that adders with a constant bit input may be simplified. It can be expected that the difference between binary and MSD coefficients would increase as the data and/or coefficient wordlength increases.

12.5.3 Subexpression sharing

In this section, adder sharing is evaluated for CIC decimation and interpolation filters with rate change factors of four and eight. Wordlengths between one and six bits were used for the decimation filters, and wordlengths between three and eight bits were used for the interpolation filters. The DF1 and TF structures were evaluated, and for all filters the resulting adder trees were bit-level optimized after all shared sub-expressions were identified and adders introduced. Binary arithmetic with a critical path of three adders were used in all cases. The savings in register complexity were limited, and thus only the savings in adder complexity are presented.

Three filters were used in cascade, and thus the single-rate impulse responses



(a) Full adder complexity.

(b) Half adder complexity.

(c) Register complexity.

(d) Energy dissipation of optimized filters.

Figure 12.16 *Complexity of FIR CIC filters with $M = 4$, $L = 3$, and $K = 2$.*

for the example filters are

$$h_4(k) = (1, 3, 6, 10, 12, 12, 10, 6, 3, 1)\,, \text{ and} \tag{12.41}$$

$$h_8(k) = (1, 3, 6, 10, 15, 21, 28, 36, 42, 46, 48, 48, 46, 42, 36, 28, 21, 15, 10, 6, 3, 1)\,, \tag{12.42}$$

for the filters of factors 4 and 8, respectively.

Results of adder sharing for the decimation filters are shown in Fig. 12.18(a). The bigger filter offers significantly more sharing potential, which is expected. Generally, increasing the data wordlength increases the number of partial products and therefore the sharing potential.

Results of adder sharing for interpolation filters are shown in Fig. 12.18(b). Since the multiple output branches of interpolation filters cannot be merged into a common adder tree as for the decimation filters, the number of candidate subexpressions is significantly reduced and the sharing potential is smaller. This is especially true for the transposed direct form, where the structural delays in the

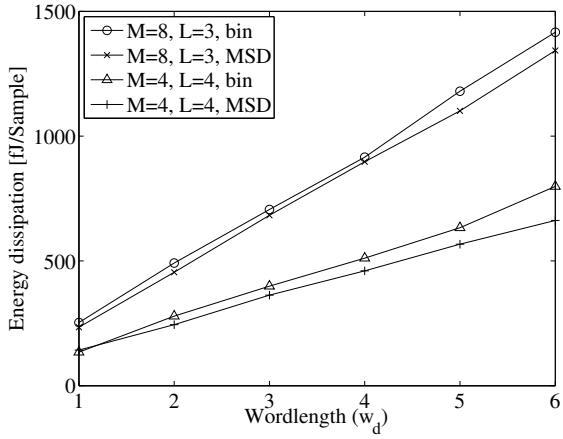
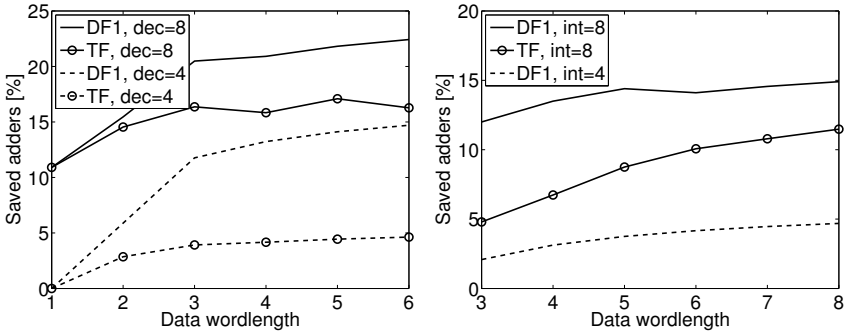


Figure 12.17 *Energy dissipation estimations using binary and MSD coefficients for FIR CIC filters with* $M = 4, L = 4$ *and* $M = 8, L = 3$.



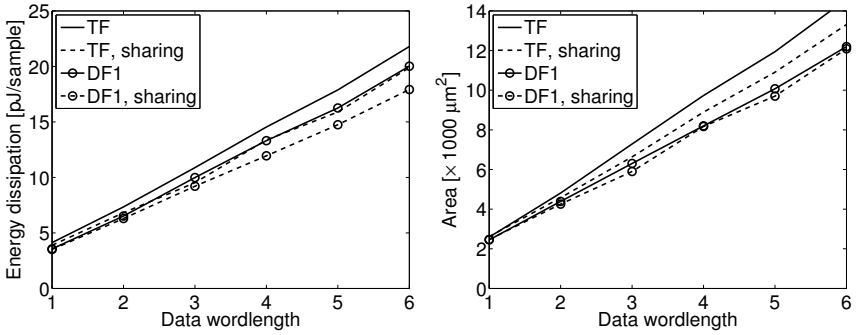
(a) Multirate FIR CIC decimation filters with factors 4 and 8.

(b) Multirate FIR CIC interpolation filters with factors 4 and 8.

Figure 12.18 *Subexpression sharing results for multirate FIR CIC filters.*

reduction tree also inhibits merging of partial products from all taps in each output branch. For the TF architecture with interpolation by four there was no sharing potential.

The factor-8 decimation filters in Figs. 12.18(a) and 12.18(b) have also been synthesized to standard cells of a 90nm CMOS process using Synopsys Design Compiler. Flipflops, full adders and half adders were used as leaf components, and the designs were synthesized for a clock frequency of 1.4 GHz. Energy and area estimations are shown in Figs. 12.19(a) and 12.19(b) respectively. It is seen that



(a) Energy dissipation estimation.

(b) Area estimation.

Figure 12.19 *Energy dissipation and area estimations of FIR CIC decimation filters with factor 8.*

the redundancy reduction reduces both the energy dissipation and the area.

13

Conclusions and future work

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

13.1 Conclusions

In part II of this thesis, contributions were made in the area of high-speed analog-to-digital conversion utilizing ΣΔ-ADCs. The contributions can be divided into two parts: a sensitivity analysis of ADC systems using parallel ΣΔ-modulators and the efficient bit-level realization of FIR filters with high speed and low wordlength, which can typically be used as the first stage decimation filter for a high-bandwidth ΣΔ-ADC.

In Chapter 10, a general model of an ADC using parallel ΣΔ-modulators is introduced. The model uses periodic modulation sequences for the different channels, and multirate filter bank theory is used to define a signal transfer function for the system. The proposed model comprises several of the traditional parallel systems, including time-interleaved systems, Hadamard-modulated systems, and frequency-decomposed systems. A channel model including non-linearities of the modulators, gain and offset errors were used, and the different systems were analyzed for matching requirements using the channel model. It was found that the sensitivities differ significantly between the different systems, with the time-interleaved system requir-

ing full matching between all channels, the Hadamard-modulated system requiring limited matching between subsets of the channels, and the frequency-decomposed system being inherently insensitive to mismatch problems. In addition, a new three-channel scheme is introduced, that is insensitive to gain errors and modulator non-idealities.

In Chapter 12, the realization of efficient high-speed FIR filters was considered, with one of the main motivations being as decimation filters for wide-bandwidth $\Sigma\Delta$-modulator. Due to the high speed and short wordlength, the popular recursive CIC structures are inefficient, and instead FIR filters realized using polyphase decomposition, partial product generation and reduction through carry-save adder trees were considered. The focus is on the efficient realization of partial product reduction trees. An ILP-based optimization algorithm was proposed, that minimizes a cost function defined as a weighted sum of the number of full adders, half adders and registers. The optimization algorithm was compared to the Reduced Area heuristic. Several different architectures, including direct form, transposed direct form, and direct form utilizing coefficient symmetry for linear-phase filters, were compared for several example filters and data wordlengths.

13.2 Future work

The following ideas are identified as possibilities for future work:

- The ILP-optimized reduction algorithm could be used also for other high-speed arithmetic. One such example is a multiply-and-accumulate, where the accumulation is recursive and will be an extra consideration in the optimization.
- The result of the optimization is a fixed structure with a constant filter impulse response. However, in the area of software defined radio, the ability to trade bandwidth for resolution is needed, which can naturally by done using a fixed $\Sigma\Delta$-ADC and varying the bandwidth of the decimation filter. Thus, a method of optimizing an adaptive filter with varying bandwidth is interesting.

References

[1] H. Aboushady, Y. Dumonteix, M.-M. Louerat, and H. Mehrez, "Efficient polyphase decomposition of comb decimation filters in ΣΔ analog-to-digital converters," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 10, pp. 898–903, Oct. 2001.

[2] T. Achterberg, "Constraint Integer Programming," Ph.D. dissertation, Technische Universität Berlin, 2007, available: http://opus.kobv.de/tuberlin/volltexte/2007/1611/.

[3] J. B. Anderson, *Digital Transmission Engineering*. IEEE Press, 1999.

[4] R. Batten, A. Eshragi, and T. S. Fiez, "Calibration of parallel ΔΣ ADCs," *IEEE Trans. Circuits Syst. II*, vol. 49, no. 6, pp. 390–399, Jun. 2002.

[5] A. Berkeman, V. Öwall, and M. Torkelson, "A low logic depth complex multiplier using distributed arithmetic," *IEEE J. Solid-State Circuits*, vol. 35, no. 4, pp. 656–659, Apr. 2000.

[6] K. Bickerstaff, M. Schulte, and E.E. Swartzlander, Jr., "Reduced area multipliers," in *Proc. Int. Conf. Application-Specific Array Processors*, Nov. 1993, pp. 478–489.

[7] A. Blad and O. Gustafsson, "Energy-efficient data representation in LDPC decoders," *IET Electron. Lett.*, vol. 42, no. 18, pp. 1051–1052, Aug. 2006.

[8] ——, "Integer linear programming-based bit-level optimization for high-speed FIR decimation filter architectures," *Springer Circuits, Syst. Signal Process. - Special Issue on Low Power Digital Filter Design Techniques and Their Applications*, vol. 29, no. 1, pp. 81–101, Feb. 2010.

[9] ——, "Redundancy reduction for high-speed FIR filter architectures based on carry-save adder trees," in *Proc. Int. Symp. Circuits, Syst.*, May 2010.

[10] ——, "FPGA implementation of rate-compatible QC-LDPC code decoder," in *Proc. European Conf. Circuit Theory Design*, Aug. 2011.

[11] A. Blad, O. Gustafsson, and L. Wanhammar, "An early decision decoding algorithm for LDPC codes using dynamic thresholds," in *Proc. European Conf. Circuit Theory Design*, Aug. 2005, pp. 285–288.

[12] ——, "A hybrid early decision-probability propagation decoding algorithm for low-density parity-check codes," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Oct. 2005.

[13] ——, "Implementation aspects of an early decision decoder for LDPC codes," in *Proc. Nordic Event ASIC Design*, Nov. 2005.

[14] ——, "An LDPC decoding algorithm utilizing early decisions," in *Proc. National Conf. Radio Science*, Jun. 2005.

[15] A. Blad, O. Gustafsson, M. Zheng, and Z. Fei, "Integer linear programming based optimization of puncturing sequences for quasi-cyclic low-density parity-check codes," in *Proc. Int. Symp. Turbo-Codes, Related Topics*, Sep. 2010.

[16] A. Blad, H. Johansson, and P. Löwenborg, "Multirate formulation for mismatch sensitivity analysis of analog-to-digital converters that utilize parallel sigma-delta modulators," *Eurasip J. Advances Signal Process.*, vol. 2008, 2008, article ID 289184, 11 pages.

[17] A. Blad, P. Löwenborg, and H. Johansson, "Design trade-offs for linear-phase FIR decimation filters and sigma-delta modulators," in *Proc. XIV European Signal Process. Conf.*, Sep. 2006.

[18] A. J. Blanksby and C. J. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[19] L. J. Breems, R. Rutten, and G. Wetzker, "A cascaded continuous-time $\Sigma\Delta$ modulator with 67-dB dynamic range in 10-MHz bandwidth," *IEEE J. Solid-State Circuits*, vol. 39, no. 12, pp. 2152–2160, Dec. 2004.

[20] J. Candy, "Decimation for sigma delta modulation," *IEEE Trans. Commun.*, vol. 29, pp. 72–76, Jan. 1986.

[21] J. C. Candy and G. C. Temes, *Oversampling methods for A/D and D/A conversion*. IEEE Press, 1992.

[22] Y.-J. Chang, F. Lai, and C.-L. Yang, "Zero-aware asymmetric SRAM cell for reducing cache power in writing zero," *IEEE Trans. VLSI Syst.*, vol. 12, no. 8, pp. 827–836, Aug. 2004.

[23] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[24] E. Choi, S. Suh, and J. Kim, "Rate-compatible puncturing for low-density parity-check codes with dual-diagonal parity structure," in *Proc. Symp. Personal Indoor Mobile Radio Commun.*, Sep. 2005, pp. 2642–2646.

[25] S.-Y. Chung, "On the construction of some capacity-approaching coding schemes," Ph.D. dissertation, Massachusets Institute of Technology, Sep. 2000.

[26] S.-Y. Chung, G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[27] S.-Y. Chung, T. J. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[28] F. Colodro, A. Torralba, and M. Laguna, "Time-interleaved multirate sigma-delta modulators," in *Proc. Int. Symp. Circuits, Syst.*, vol. 6, May 2005, pp. 5581–5584.

[29] R. F. Cormier, T. L. Sculley, and R. H. Bamberger, "Combining subband decomposition and sigma-delta modulation for wideband A/D conversion," in *Proc. Int. Symp. Circuits, Syst.*, vol. 5, May 1994, pp. 357–360.

[30] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, May 1965.

[31] D. Declercq and F. Verdier, "Optimization of LDPC finite precision belief propagation decoding with descrete density evolution," in *Proc. Int. Symp. Turbo-Codes, Related Topics*, Sep. 2003, pp. 479–482.

[32] G. Ding, C. Dehollain, M. Declercq, and K. Azadet, "Frequency-interleaving technique for high-speed A/D conversion," in *Proc. Int. Symp. Circuits, Syst.*, vol. 1, May 2003, pp. 857–860.

[33] Y. Dumonteix and H. Mehrez, "A family of redundant multipliers dedicated to fast computation for signal processing," in *Proc. Int. Symp. Circuits, Syst.*, vol. 5, May 2000, pp. 325–328.

[34] A. Eshraghi and T. S. Fiez, "A time-interleaved parallel $\Delta\Sigma$ A/D converter," *IEEE Trans. Circuits Syst. II*, vol. 50, no. 3, pp. 118–129, Mar. 2003.

[35] ——, "A comparative analysis of parallel delta-sigma ADC architectures," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 3, pp. 450–458, Mar. 2004.

[36] T. Etzion, A. Trachtenberg, and A. Vardy, "Which codes have cycle-free Tanner graphs?" *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2173–2181, Sep. 1999.

[37] M. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[38] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673–680, May 1999.

[39] R. Gallager, "Low-density parity-check codes," Ph.D. dissertation, Massachusetts Institute of Technology, 1963.

[40] I. Galton and H. T. Jensen, "Delta-sigma modulator based A/D conversion without oversampling," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 12, pp. 773–784, Dec. 1995.

[41] ——, "Oversampling parallel delta-sigma modulator A/D conversion," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 12, pp. 801–810, Dec. 1996.

[42] Y. Gao, L. Jia, J. Isoaho, and H. Tenhunen, "A comparison design of comb decimators for sigma-delta analog-to-digital converters," *Springer Analog Integrated Circuits, Signal Process.*, vol. 22, no. 1, pp. 51–60, Jan. 2000.

[43] M. Good and F. R. Kschischang, "Incremental redundancy via check splitting," in *Proc. Biennial Symp. Commun.*, 2006, pp. 55–58.

[44] F. Guillod, E. Boutillon, J. Tousch, and J.-L. Danger, "Generic description and synthesis of LDPC decoders," *IEEE Trans. Commun.*, vol. 55, no. 11, pp. 2084–2091, Nov. 2007.

[45] O. Gustafsson and H. Ohlsson, "A low power decimation filter architecture for high-speed single-bit sigma-delta modulation," in *Proc. Int. Symp. Circuits, Syst.*, vol. 2, May 2005, pp. 1453–1456.

[46] J. Ha, J. Kim, D. Klinc, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 728–738, 2006.

[47] J. Ha, J. Kim, and S. W. McLaughlin, "Rate-compatible puncturing of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. IT-50, no. 11, pp. 2824–2836, Nov. 2004.

[48] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389–400, Apr. 1988.

[49] O. Herrmann, R. L. Rabiner, and D. S. K. Chan, "Practical design rules for optimum finite impulse response digital filters," *Bell System Technical J.*, vol. 52, no. 6, pp. 769–799, 1973.

[50] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. Workshop, Signal Proc. Syst.*, Oct. 2004, pp. 107–112.

[51] E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *Proc. Int. Conf. Acoust. Speech, Signal Process.*, vol. 29, pp. 155–162, Apr. 1981.

[52] C. Howland and A. Blanksby, "A 220mW 1Gb/s 1024-bit rate-1/2 low density parity check code decoder," in *Proc. Custom Integrated Circuits Conf.*, May 2001, pp. 293–296.

[53] P. G. A. Jespers, *Integrated Converters: D to A and A to D Architectures, Analysis and Simulation.* New York: Oxford University Press, 2001.

[54] K. Johansson, O. Gustafsson, and L. Wanhammar, "Implementation of elementary functions for logarithmic number systems," *IET J. Computers, Digital Techniques, IET*, vol. 2, no. 4, pp. 295–304, Jul. 2008.

[55] S. J. Johnson and S. R. Weller, "Construction of low-density parity-check codes from Kirkman triple systems," in *Proc. Global Commun. Conf.*, Nov. 2003, pp. 970–974.

[56] R. Khoini-Poorfard and D. A. Johns, "Time-interleaved oversampling converters," *IET Electron. Lett.*, vol. 29, no. 19, pp. 1673–1674, Sep. 1993.

[57] ——, "Mismatch effects in time-interleaved oversampling converters," in *Proc. Int. Symp. Circuits, Syst.*, vol. 5, May 1994, pp. 429–432.

[58] R. Khoini-Poorfard, L. B. Lim, and D. A. Johns, "Time-interleaved oversampling A/D converters: Theory and practice," *IEEE Trans. Circuits Syst. II*, vol. 44, no. 8, pp. 634–645, Aug. 1997.

[59] K.-Y. Khoo, Z. Yu, and Alan N. Willson, Jr., "Bit-level arithmetic optimization for carry-save additions," in *Proc. Computer-Aided Design, Digest of Technical Papers*, Nov. 1999, pp. 14–18.

[60] J.-A. Kim, S.-R. Kim, D.-J. Shin, and S.-N. Hong, "Analysis of check-node merging decoding for punctured LDPC codes with dual-diagonal parity structure," in *Proc. Wireless Commun. Netw. Conf.*, 2007, pp. 572–576.

[61] E. King, F. Aram, T. Fiez, and I. Galton, "Parallel delta-sigma A/D conversion," in *Proc. Custom Integrated Circuits Conf.*, May 1994, pp. 503–506.

[62] S. K. Kong and W. H. Ku, "Frequency domain analysis of ΠΔΣ ADC and its application to combining subband decomposition and ΠΔΣ ADC," in *Proc. Midwest Symp. Circuits, Syst.*, vol. 1, Aug. 1997, pp. 226–229.

[63] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[64] M. Kozak, M. Karaman, and I. Kale, "Efficient architectures for time-interleaved oversampling delta-sigma converters," *IEEE Trans. Circuits Syst. II*, vol. 47, no. 8, pp. 802–810, Aug. 2000.

[65] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[66] T.-C. Kuo and J. A. N. Willson, "A flexible decoder IC for WiMAX QC-LDPC codes," in *Proc. Custom Integrated Circuits Conf.*, 2008, pp. 527–530.

[67] C. Kuskie, B. Zhang, and R. Schreier, "A decimation filter architecture for GHz delta-sigma modulators," in *Proc. Int. Symp. Circuits, Syst.*, vol. 2, May 1995, pp. 953–956.

[68] Y. Li, M. Elassal, and M. Bayoumi, "Power efficient architecture for (3,6)-regular low-density parity-check code decoder," in *Proc. Int. Symp. Circuits, Syst.*, vol. IV, May 2004, pp. 81–84.

[69] C.-Y. Lin, C.-C. Wei, and M.-K. Ku, "Efficient encoding for dual-diagonal structured LDPC codes based on parity bit prediction and correction," in *Proc. Asia-Pacific Conf. Circuits, Syst.*, 2008, pp. 1648–1651.

[70] C.-C. Lin, K.-L. Lin, H.-C. Chang, and C.-Y. Lee, "A 3.33Gb/s $(1200, 720)$ low-density parity-check code decoder," in *Proc. European Solid-State Circuits Conf.*, Sep. 2005, pp. 211–214.

[71] S. Lin, L. Chen, J. Xu, and I. Djurdjevic, "Near Shannon limit quasi-cyclic low-density parity-check codes," in *Proc. Global Commun. Conf.*, May 2003, pp. 2030–2035.

[72] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *Proc. Annual Symp. Theory Computing*, 1998, pp. 249–258.

[73] D. MacKay, "Good error-correcting codes based on very sparse matrices," in *Proc. Int. Symp. Inf. Theory*, Jun. 1997, p. 113.

[74] ——, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[75] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *IET Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[76] U. Meyer-Baese, S. Rao, J. Ramírez, and A. García, "Cost-effective Hogenauer cascaded integrator comb decimator filter design for custom ICs," *IET Electron. Lett.*, vol. 41, no. 3, pp. 158–160, Feb. 2005.

[77] O. Milenkovic, I. B. Djordjevic, and B. Vasic, "Block-circulant low-density parity-check codes for optical communication systems," *IEEE J. Sel. Topics Quantum Electron.*, vol. 10, no. 2, pp. 294–299, Mar. 2004.

[78] S. K. Mitra, *Digital Signal Processing: A Computer Based Approach*. McGraw-Hill, 2006.

[79] F. Naessens, V. Derudder, H. Cappelle, L. Hollevoet, P. Raghavan, M. Desmet, A. AbdelHamid, I. Vos, L. Folens, S. O'Loughlin, S. Singirikonda, S. Dupont, J.-W. Weijers, A. Dejonghe, and L. Van der Perre, "A 10.37 mm^2 675 mW reconfigurable LDPC and Turbo encoder and decoder for 802.11n, 802.16e and 3GPP-LTE," in *Proc. Symp. VLSI Circuits*, 2010, pp. 213–214.

[80] V. T. Nguyen, P. Loumeau, and J.-F. Naviner, "Analysis of time-interleaved delta-sigma analog to digital converter," in *Proc. Veh. Tech. Conf.*, vol. 4, Sep. 2002.

[81] S. R. Norsworthy, R. Schreier, and G. C. Temes, Eds., *Delta-Sigma Data Converters: Theory, Design, and Simulation*. Wiley-IEEE Press, 1996, ISBN 0780310454.

[82] H. Ohlsson, B. Mesgarzadeh, K. Johansson, O. Gustafsson, P. Löwenborg, H. Johansson, and A. Alvandpour, "A 16 GSPS 0.18 μm CMOS decimator for single-bit $\Sigma\Delta$-modulation," in *Proc. Nordic Event ASIC Design*, Nov. 2004, pp. 175–178.

[83] H. Y. Park, J. W. Kang, K. S. Kim, and K. C. Whang, "Efficient puncturing method for rate-compatible low-density parity-check codes," *IEEE Trans. Wireless Commun.*, vol. 6, no. 11, pp. 3914–3919, Nov. 2007.

[84] T. Parks and J. McClellan, "Chebyshev approximation for nonrecursive digital filters with linear phase," *IEEE Trans. Circuit Theory*, vol. 19, no. 2, pp. 189–194, 1972.

[85] L. Rabiner and O. Herrmann, "The predictability of certain optimum finite impulse response digital filters," *IEEE Trans. Circuit Theory*, vol. CT-20, no. 4, pp. 401–408, Jul. 1973.

[86] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. vol. 47, no. 2, pp. 619–637, Feb. 2001.

[87] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. vol. 47, no. 2, pp. 599–618, Feb. 2001.

[88] R. Schreier and G. C. Temes, *Understanding Delta-Sigma Data Converters*. John Wiley & Sons, NJ, 2005.

[89] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical J.*, vol. 27, pp. 379–423, 623–656, 1948.

[90] X.-Y. Shih, C.-Z. Zhan, and A.-Y. Wu, "A 7.39 mm^2 76mW (1944, 972) LDPC decoder chip for IEEE 802.11n applications," in *Proc. Asian Solid-State Circuits Conf.*, Nov. 2008, pp. 301–304.

[91] J. E. Swartzlander, "Merged arithmetic," *IEEE Trans. Comput.*, vol. C-29, no. 10, pp. 946–950, Oct. 1980.

[92] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[93] K. Uyttenhove and M. Steyaert, "Speed-power-accuracy trade-off in high-speed ADCs," *IEEE Trans. Circuits Syst. II*, vol. 4, pp. 247–257, Apr. 2002.

[94] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Eaglewood Cliffs, NJ: Prentice-Hall, 1993.

[95] B. Vasic and O. Milenkovic, "Combinatorial constructions of low-density parity-check codes for iterative decoding," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 1156–1176, Jun. 2004.

[96] B. Vasic, K. Pedagani, and M. Ivkovic, "High-rate girth-eight low-density parity-check codes on rectangular integer lattices," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 1248–1252, Aug. 2004.

[97] C. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[98] Y. Wang, J. Yedidia, and S. Draper, "Construction of high-girth QC-LDPC codes," in *Proc. Int. Symp. Turbo-Codes, Related Topics*, Sep. 2008, pp. 180–185.

[99] Z. Wang, Z. Cui, and J. Sha, "VLSI design for low-density parity-check code decoding," *IEEE Circuits Syst. Mag.*, vol. 11, no. 1, pp. 52–69, 2011.

[100] L. Wanhammar and H. Johansson, *Digital Filters*. Linköping University, 2002.

[101] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University, Linköping, Sweden, Jun. 1996.

[102] S. B. Wicker, *Error Control Systems*. Prentice Hall, 1995.

[103] N. Yaghini and D. Johns, "A 43mW CT complex $\Delta\Sigma$ ADC with 23MHz of signal bandwidth and 68.8dB SNDR," in *Proc. Int. Solid-State Circuits Conf., Digest of Technical Papers*, Feb. 2005, pp. 502–503.

[104] E. Yeo, B. Nikolic, and V. Anantharam, "Architectures and implementations of low-density parity check decoding algorithms," in *Proc. Midwest Symp. Circuits, Syst.*, Aug. 2002, pp. III–437 – III–440.

[105] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magn.*, vol. 37, no. 2, pp. 748–755, Mar. 2001.

[106] K. Zhang, X. Huang, and Z. Wang, "A high-throughput LDPC decoder architecture with rate compatibility," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 4, pp. 839–847, Apr. 2011.

[107] T. Zhang and K. K. Parhi, "Joint code and decoder design for implementation-oriented $(3, k)$-regular LDPC codes," in *Proc. Asilomar Conf. Signals, Syst., Comp.*, Nov. 2001, pp. 1232–1236.

[108] ——, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *Proc. Workshop, Signal Proc. Syst.*, Oct. 2002, pp. 127–132.

[109] ——, "An FPGA implementation of (3,6) regular low-density parity-check code decoder," *Eurasip J. Applied Signal Process.*, vol. 6, pp. 530–542, May 2003.

[110] ——, "Joint $(3, k)$-regular LDPC code and decoder/encoder design," *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1065–1079, Apr. 2004.

[111] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low density parity check codes decoder," in *Proc. Int. Symp. Circuits, Syst.*, May 2001, pp. 202–205.

[112] E. Zimmermann, G. Fettweis, P. Pattisapu, and P. K. Bora, "Reduced complexity LDPC decoding using forced convergence," in *Proc. Int. Symp. Wireless Personal Multimedia Commun.*, Sep. 2004.

[113] "ETSI EN 302 307, digital video broadcasting (DVB): Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2)," available:
http://www.dvb.org/technology/standards/.

[114] "ETSI EN 302 755, digital video broadcasting (DVB): Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)," available: http://www.dvb.org/technology/standards/.

[115] "ETSI EN 302 769, digital video broadcasting (DVB): Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2)," available: http://www.dvb.org/technology/standards/.

[116] "GSFC-STD-9100, low density parity check code for rate 7/8," available: http://standards.gsfc.nasa.gov/stds.html.

[117] "High-speed FIR filter generator," available: http://www.es.isy.liu.se/software/hsfir/.

[118] "IEEE 802.11n-2009: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 5: Enhancements for higher throughput," available: http://standards.ieee.org/getieee802/download/802.11n-2009.pdf.

[119] "IEEE 802.15.3c-2009: Wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs) amendment 2: Millimeter-wave-based alternative physical layer extension," available: http://standards.ieee.org/getieee802/download/802.15.3c-2009.pdf.

[120] "IEEE 802.16e-2005: Air interface for fixed and mobile broadband wireless access systems," available: http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.

[121] "IEEE 802.3an-2006: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications," available: http://standards.ieee.org/getieee802/download/802.3an-2006.pdf.

[122] "National standardization management committee. GB2060-2006. digital television terrestrial broadcasting transmission system frame structure, channel coding and modulation. Beijing: Standards press of China. Aug. 2006."