Frank Kienle

Architectures for Baseband Signal Processing

Springer

Architectures for Baseband Signal Processing

Frank Kienle

Architectures for Baseband Signal Processing

Springer

Frank Kienle
Department of Electrical Engineering
TU Kaiserslautern
Kaiserslautern
Germany

ISBN 978-1-4614-8029-7 ISBN 978-1-4614-8030-3 (eBook)
DOI 10.1007/978-1-4614-8030-3
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013940940

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

For Nadine, Mara and Amelie

Preface

Mobile communication devices like smart phones or tablet PCs enable us to consume information at every location and at every time. The rapid development of new applications and new services and the demand to access data in real time create an increasing throughput demand. The data have to be transmitted reliably to ensure the desired quality of service. Furthermore, an improved utilization of the bandwidth is desired to reduce the cost of transmission. All these demands lead to an increased complexity of the algorithms employed in wireless transceivers. In addition, the hardware realization of mobile transceivers has to feature a small area and power profile to reduce fabrication costs and to increase the user convenience. The rapid technology improvements of the chip industry allow the integration of more and more functionality on every chip.

Dealing with the increase of algorithmic complexity and required data throughput poses a big problem for the algorithm designer and the chip designer. An algorithm designer needs to understand the impact of his design decisions on the chip complexity, while a chip designer needs to understand the employed algorithms to enable an efficient implementation. Thus, a comprehensive understanding of algorithmic and architectural constraints becomes mandatory and has to be considered within educational programs, as well.

This manuscript addresses students in electrical engineering, computer engineering, and computer science with an interest in the field of communications engineering, architectures, and microelectronic design. An according lecture is given at the University of Kaiserslautern and the Karlsruhe Institute of Technology, both Germany. The manuscript puts a special focus on implementation aspects and implementation constraints of individual components that are needed in transceivers for current standards like UMTS, LTE, WiMAX, and DVB-S2. The application domain is the so-called outer receiver, which comprises the channel coding, interleaving stages, modulator, and multiple antenna transmission. Within a receiver device, the task of the outer receiver is to transmit information at the highest practical rate, as reliably as required by the application, i.e., voice or data

transmission. Throughout the manuscript, the focus lies on advanced algorithms that are actually in use in modern communications systems. Their basic principles are always derived with a focus on the resulting communications and implementation performance.

Kaiserslautern, December 2012 Frank Kienle

Acknowledgments

This manuscript results from my work as ‘Akademischer Rat’ from 2008 until 2012 at the Microelectronic System Design Research Group at the University of Kaiserslautern. I would like to thank at least some of the numerous people who have in some way contributed to this work.

Especially, I would like to thank my mentor Prof. Dr.-Ing. Norbert Wehn for the opportunity to work in the interesting field of digital communication systems and VLSI design, for the numerous insightful and hot discussions.

I am very grateful for the colleagues I had the chance to work within the Microelectronic System Design Research Group: Michael J. Thul, Timo Vogt, Christian Neeb, Torben Brack, Daniel Schmidt, Matthias May, Timo Lehnigk-Emden, Matthias Alles, Christian Brehm, Christian de Schryver, Thomas Ilnseher, Christina Gimmler-Dumont, Christian Weis, and our secretary Martina Jahn. It is the team spirit that makes this group successful.

Especially, I would like to thank my wife, children, and parents for their encouragement and continuous support during my work.

Contents

1 Introduction 1
1.1 Application: Outer Receiver. 8
1.2 Dependencies and Overview of Chapters 11
References 14

2 Digital Transmission System. 17
2.1 Channel Model 19
2.2 Digital Modulator and Demodulator. 24
2.3 Channel Capacity 28
2.4 Multiple Antenna Systems 32
References 35

3 Channel Coding Basics. 37
3.1 Overview Channel Coding 38
3.2 Linear Block Codes 39
3.3 General Decoding Problem 42
3.3.1 Maximum Likelihood (ML) Decoding 42
3.3.2 ML Decoding by Solving IP Problems 44
3.3.3 Symbol-by-Symbol MAP Decoding. 49
3.3.4 Max-Log-MAP Approximation 51
3.4 Convolutional Codes 52
3.4.1 ML Decoding of Convolutional Codes 55
3.4.2 Max-Log-MAP Decoding of Convolutional Codes 58
3.5 Soft-Input Soft-Output (SISO) Decoder 62
References 64

4 Hardware Design of Individual Components 67
4.1 Design Flow 68
4.1.1 Algorithmic Design Space Exploration 68
4.1.2 Hardware Design Space Exploration 70
4.2 SRAM Memories 73
4.2.1 SRAM Design Space Exploration 76
4.2.2 Exemplary SRAM Data: Area, Power, Cycle Time 79

4.2.3 Importance of Memory Exploration. 84
4.3 Individual Component: Interleaver. 87
4.3.1 Interleaver Types 87
4.3.2 Hardware Realization 90
References 95

5 Data Path of Individual Components 97
5.1 Data Flow to Data Path Example 98
5.1.1 Serial Data Path: One min (x, y) Unit 100
5.1.2 Serial Data Path: Two $min(x, y)$ Units 104
5.1.3 Data Path: Parallel Processing 106
5.2 Deriving Processing Units 108
5.2.1 Recursion Units 109
5.2.2 Look-Up Tables 111
5.2.3 Deriving an Approximation Function. 112
Reference 116

6 Turbo Codes 117
6.1 Encoder Structure 117
6.2 The Iterative Decoding Procedure 120
6.2.1 Convergence Progress (EXIT Charts) 122
6.2.2 Communications Performance 125
6.2.3 Fixed-Point Realization and Robustness. 128
6.3 Turbo Codes Architecture. 131
6.3.1 Serial MAP Architecture 131
6.3.2 Design Space and Design Choices 135
6.3.3 Dependencies of Architectural Parameters 142
References 144

7 Low-Density Parity-Check Codes 147
7.1 Basic Definition 147
7.2 Decoding 150
7.3 Structured LDPC Codes 155
7.3.1 Quasi-Cyclic LDPC Codes. 155
7.3.2 Hidden Nodes/Multi-Edge Type LDPC Codes 158
7.3.3 Encoder, H Matrix, Tanner Graph. 160
7.4 LDPC Decoder Architecture 163
7.4.1 Design Space and Design Choices 164
7.4.2 Check Node Implementation 166
7.4.3 Mapping from Parity Check Matrix to Hardware 169
7.4.4 Advanced Topics for LDPC Decoder Realization 172
7.5 Joint Code Construction and Architecture Design 176
References 180

8 Bit-Interleaved Coded MIMO System 185
8.1 State-of-the-Art BICM-MIMO Systems 187
8.1.1 MIMO Transmitter 187
8.1.2 BICM-MIMO Receiver 188
8.1.3 Communications Performance of State-of-the-Art Systems 194
8.2 Architecture Feasibility BICM-MIMO 197
8.3 Joint Architecture-Algorithm Design 201
8.3.1 Sphere Decoder Aware Bit Interleaver Design 204
8.3.2 Sphere Decoder Aware LDPC Code Design 207
References 209

9 Comparing Architectures 211
9.1 Reference Designs 215
9.2 Suitable Metrics 217
9.3 Approach for Design Space Exploration 219
9.3.1 Implementation Driven Design Space Exploration 220
9.3.2 Communications Performance Driven Exploration 222
9.4 Comparison of Published Decoder Architectures 224
9.4.1 Technology Parameters 224
9.4.2 Communications Performance 227
9.4.3 VLSI Efficiency and Communications Performance at Once 228
References 229

Appendix A: Channel Coding Data 233

Appendix B: Communications Performance Results 237

Appendix C: Probability Terms 243

Appendix D: 4-AM Demodulator 249

Index 257

Acronyms

Mathematical Terms

$\boldsymbol{A}$,	Bold capital letters represent matrices
a_i,	Bold lowercase letters with index represent column vectors of a matrix
$\boldsymbol{x}$,	Bold lowercase letters represent vectors
A_{ij}	Entry of row i and column j of matrix $\boldsymbol{A}$
x_i	Element i of vector $\boldsymbol{x}$
$\boldsymbol{Q}^T$	Transpose of $\boldsymbol{Q}$
$\boldsymbol{Q}^H$	Hermitian transpose of $\boldsymbol{Q}$
$\Re\{x\}$	Real part of complex value x
$\Im\{x\}$	Imaginary part of complex value x
LSB	Least significant bit is always located at the most right position

Basic Transmission System

μ	Mean value
σ^2	Variance
σ	Standard deviation
N_0	One-sided noise power spectral density
E_b	Energy per information bit
E_S	Energy per coded symbol
SNR=E_S/N_0	Signal-to-noise ratio
$\sigma^2 = N_0$	Real-valued channel
$\sigma^2 = \frac{N_0}{2}$	Per complex dimension
$\frac{E_b}{N_0} = \log_2(Q) \cdot R \cdot \frac{E_s}{N_0}$	Information bit to noise ratio
C	Channel capacity
$\boldsymbol{W}$	Bandwidth
N	Codeword length
N_s	Number of symbols (modulated codeword)
$\boldsymbol{K}$	Number of information bits

$\boldsymbol{M}$	Number of parity bits
i,k	Time or step indices
$\boldsymbol{u}$	Information vector
u_k	Information bit at position k
$\hat{\boldsymbol{u}}$	Estimated information bits
$\boldsymbol{x}$	Codeword
x_k	Coded bit at position k
$\boldsymbol{s}$	Transmitted symbol vector after mapping (BPSK, QPSK, 16-QAM,…)
s_k	Transmitted symbol at time step k
$\boldsymbol{n}$	Gaussian noise vector
$\boldsymbol{y}$	Received vector
y_k	Received symbol at position k
$P(\ldots)$	Probability of $(\ldots)$
$P_{\mathrm{x}}^{(x)}$	The probability that the random variable X takes on the value x
Q	Number of bits per modulated symbol
2^Q	Modulation alphabet
Π	Interleaver
Π^{-1}	Deinterleaver

Channel Coding and Decoding

$\boldsymbol{G}$	Generator matrix of a linear block code
$\boldsymbol{H}$	Parity-check matrix of a linear block code
R	Code rate
K_c	Constraint length of a convolutional codes
M	Number of memories of a convolutional encoder
I_i	Input polynomial i
G_{FB}	Feedbackward polynomial
G_i	Feedforward polynomial i
$X_K^{S(i)}$	ith systematic bit at trellis step k
$x_k^{p(i)}$	ith parity bit at trellis step k
$S_k^{(m)}$	Identifier for the mth state in trellis step k
$\gamma_k^{x_i,x_j+1}$	Branch metric for a bit pattern x_i, x_{i+1} at trellis step k
α_k^m	Forward recursion state metric of S_k^m
β_k^m	Backward recursion state metric of S_k^m
$\boldsymbol{\lambda}^s$	APPs of the received systematic part
$\boldsymbol{\lambda}^p$	APPs of the received parity part
$\boldsymbol{\Lambda}$	Maximum a posteriori values expressed in LLRs
Λ_i	Maximum a posteriori value at position i
$\boldsymbol{L}^e$	Extrinsic information (log-likelihood)

$\boldsymbol{L}_a$	A priori information (log-likelihood)
d_{min}	Minimum Hamming distance
(d_v,d_c)	Column and row weight of H
f	Degree distribution of variable nodes
g	Degree distribution of check nodes
VN	Variable node
CN	Check node
d_v^{max}	Maximum VN degree
d_c^{max}	Maximum CN degree
d_v^i	Degree of VN at position i
IN	Information (variable) node associated to an information bit

Architectures

t_{cond}	Technology node and operating conditions w.r.t. PVT settings
PVT	Process, voltage, and temperature settings
t_{cyc}	Cycle time
f_{cyc}	Cycle frequency of the design
Q_{in}	Quantization of the input data
Q_{ext}	Quantization of the extrinsic data
iter	Number of iterations of the channel decoder (half iterations for turbo decoding)
BL	Processed block length, may differ to codeword size
WL	Window length of the trellis processing
AL	Acquisition length, the number of training steps for recursions
rdx	Radix-2 or radix-4 realization of the recursion units
C_N	Latency due to network to realize interleaving
P	Parallelization of the decoder architecture: for turbo codes parallelization of the MAP architecture, for LDPC codes the number of concurrently processed edges
$P_{I/O}$	Parallelization of the input/output
E	Number of edges in a Tanner-graph
T_{payl}	Payload throughput, architecture throughput of the information bits in [bit/s]
$T_{latency}$	Architecture dependent latency
$\delta_{overhead}$	Additional fixed overhead due to architecture constraints (e.g. fill the pipeline)
$\frac{\#cycles}{d_{VN}}$	Number of cycles required to process one block average variable node degree in the Tanner graph (LDPC codes)
$\# \frac{bits}{cycle}$	Normalized throughput: number of information bits decoded per clock cycle

MIMO

M_T	Number of transmit antennas
M_R	Number of receive antennas
T	Number of channel uses per codeword
$\boldsymbol{H}$	Channel matrix, $M_R \times M_T$ matrix
$\boldsymbol{Y}$	Received values, $M_R \times T$ matrix
$\boldsymbol{S}$	Modulated symbols, $M_T \times T$ matrix
$\boldsymbol{N}$	Gaussian noise, $M_T \times T$ matrix
$\boldsymbol{y}_t$	Received vector, column t of Y, t $\in$ [1,T]
$y_t(i)$	Received symbol i of received vector y_t, $i \in [1,M_r]$
$\boldsymbol{\lambda}$	LLRs after demodulator
$\boldsymbol{\lambda}^e$	LLRs after demodulator without a priori LLRs, $\boldsymbol{\lambda} - \boldsymbol{L}^a$
$\boldsymbol{\lambda}^a$	Channel values, input LLRs for channel decoder, $\Pi\{\boldsymbol{\lambda}^e\}$
$\boldsymbol{\Lambda}$	APP LLRs after channel decoder
$\boldsymbol{L}^e$	Extrinsic LLRs of channel decoder, $\boldsymbol{\Lambda} - \lambda^a$
$\boldsymbol{L}^a$	A priori LLRs of demodulator, $\Pi\{\boldsymbol{L}^e\}$
$\boldsymbol{Q}$	Orthonormal matrix of QR decomposition of $\boldsymbol{H}$
$\boldsymbol{R}$	Upper-triangular matrix of QR decomposition of $\boldsymbol{H}$
$\hat{\boldsymbol{Y}}$	Result of $\boldsymbol{Q}^{\boldsymbol{H}} \cdot \boldsymbol{Y}$
$T_i\left(\boldsymbol{S}^{(i)}\right)$	Partial (squared) Euclidian Distance (PED) in level i of the tree
$\left\|e_i\left(\boldsymbol{S}^{(i)}\right)\right\|^2$	Distance increment for PED

Chapter 1
Introduction

Practically all modern communication systems, like digital video broadcasting (DVB), wireless local area networks (WLAN), and all mobile cellular standards currently in use, transmit information in digital form. The shift from analog transmission to digital transmission in practical applications and products was fueled by the rapid development of integration densities of digital circuits. Today's so called very-large-scale integration(VLSI) processes allow millions (or even billions) of transistors to be integrated in one single chip. The development of new communication systems and the rapid progress of chip technology lead to ever more complex systems. With every system generation, larger integration densities allow to integrate a widened set of functionalities into these systems, which in turn drives the development of larger integration densities. Transceivers of future cellular standards will have to support virtually all existing wireless standards, shall provide a throughput of up to 1 Gbit/s, and all that within the limited power budget of a mobile phone.

Both domains, chip design and communication systems, show a rapid growth in complexity. Every designer of such a system-on-a-chip (SoC) is required to have a comprehensive, interdisciplinary understanding of both domains, which are typically thought as separate major subjects in university courses. To achieve efficient designs—efficient in terms of constraints posed by the application and in terms of chip design, e.g. area and power consumption—it is of increasing importance that engineers learn to bridge these two distinct engineering domains.

Many good text books exist in the area of hardware design, e.g. [1–4], as well as in the field of digital communication systems, e.g. [5–9]. The list of good references is far from being complete, for both areas. However, it is difficult to find a text book which covers both domains, the algorithm design—even when it does not specialize in communication systems—and the hardware design. This manuscript tries to fill this void and bridge the two domains, using the so called outer receiver of a digital communication system as an exemplary application.

The next sections give an overview of the basic trends in VLSI technology as well as in wireless communications, followed by the introduction of the application example which is used throughout the manuscript.

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_1, © Springer Science+Business Media New York 2014

The terms chip or integrated circuit (IC) are general expressions, which just refer to an electronic circuit based on semiconductor materials. One very important semiconductor material is silicon, which is why the IC manufacturers are often called as well the silicon industry. One important property of a semiconductor material is that it is possible to change its conductivity by inserting positively or negatively charged atoms into the material, a process which is called doping.

Every structural element within an integrating circuit is thus composed of a different combination of doped semiconductor materials. Additional materials are utilized as well during chip fabrication, as for example highly conductive materials and isolation materials. Very-large-scale integration (VLSI) is the current integration technique which enables the instantiation of millions of transistors on one chip. The basic idea here is the repeated usage of regular structures to ensure a homogeneous fabrication process.

One of the most important logic families is called complementary metal oxide semiconductor (CMOS) logic. A CMOS structure is typically composed of two 'complementary' metal-oxide-semiconductor field-effect transistors (MOSFET). Fabrication processes for highly integrated CMOS circuits—so called CMOS processes—are thus a good indicator of technological progress.

Figure 1.1 shows the development trends of CMOS processes over the years. The y-axis shows the maximum number of transistors that can be integrated in one chip and the minimum gate length in μ m for a MOSFET, which can be seen as the minimum geometric object size in an integrated circuit. The future trend for process

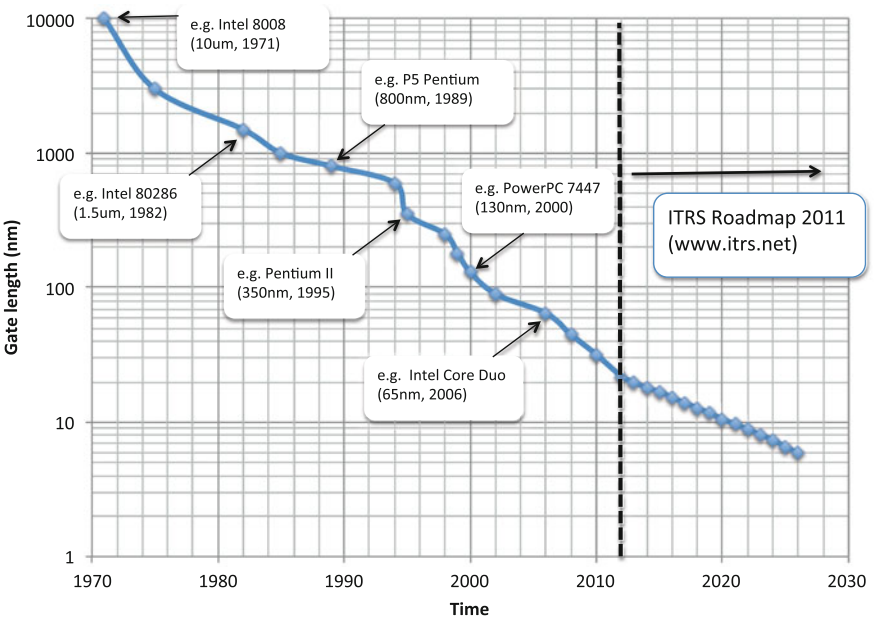


Fig. 1.1 Old and current trends in semiconductor integration densities, for future trends see [10]

shrinking is given by the International Technology Road Map for Semiconductors (ITRS) [10], which is a partnership network of semiconductor manufacturers. The ITRS road map comprises the common assumption of the manufactures about the future trends and challenges for the chip technology.

The terms CMOS and MOSFET are not explained in detail here. The important message of Fig. 1.1 is that the current technology progress still follows Moore's law of integrated circuits, which states that the number of transistors that can be placed on a given area doubles approximately every 2 years [11].

Special care has to be taken when evaluating VLSI efficiency characteristics like area and power consumption. For every given technology node, e.g. 40 nm, there are different fabrication processes. A process developer like the Taiwan Semiconductor Manufacturing Company (TSMC) can provide a so called low power process or a high performance process. Note, that more process types are possible. The differences of some important implementation figures of an ARM [12] processor for two different processes are shown exemplary in Table. 1.1. We highlight here three important aspects about power, area, and performance.

- Power consumption: For the power consumption of a chip we have to distinguish between dynamic power consumption and static power consumption. Dynamic power consumption refers to the case when the processor is working. The static power consumption is of importance when the processor is idle. A low power process is often a low leakage process which optimizes the static power consumption. A mobile phone is one typical example of a device which is in idle mode for most of the time. Thus, a low power process is utilized. However, under the same usage conditions (voltage, work load, temperature), the dynamic power consumption of a circuit fabricated in a low power process will be higher than if it was fabricated in a high performance process.
- Area: The area for different process types does not change for these two process variants. Furthermore, it can be seen that 50 % the overall area of this ARM device is in determined by memories. For future devices it is assumed that the area for memories will more and more determine the entire area. Thus, for area comparison it is always important to specify the storage capabilities as done in Table. 1.1.

Table 1.1 Difference between a low power process and a high performance process, both at a 40 nm technology node [13]

	ARM Cortex-A5 Performance, Power, Area	
	TSMC 40LP	TSMC 40G
Process type/Nominal voltage	**Low leakage, 1.1.V**	**Performance, 1.0 V**
Frequency (optimized)	530–600 MHz	$\geq$1 GHz
Area excluding RAMs/cache	0.27 mm^2	0.27 mm^2
Area with 16 kbyte RAMs/16 kbyte cache	0.53 mm^2	0.53 mm^2
Dynamic power	0.12 mW/MHz	$\leq$0.08 mW/MHz

- Performance: The performance is often measured in operations or instructions per second. This number depends of course on the achieved frequency which largely differs between these two process variants.

Table. 1.1 describes the influence of the process type for one ARM processor which will be embedded in a larger system. The ARM processor is a programmable device and, thus, offers a large flexibility with respect to a given application.

A processor is only one implementation possibility, however, many different implementation styles are possible. With the term 'implementation styles' we refer to the choice of different hardware platforms and thus different possibilities with respect to area efficiency, power efficiency, and flexibility. The implementation style is one major decision which has to be done early during product development.

General purpose processors (GPP) provide the highest flexibility, however, with the highest power consumption compared to other implementation styles. ASICs are up to 5 orders of magnitude more power and area efficient than GPPs. An application-specific integrated circuit (ASIC) provides the highest performance and lowest energy consumption. ASICs are designed for one specific task and are optimized with respect to the required performance. An overview figure is presented in Fig. 1.2, with the area efficiency depicted on the x-axis and the energy efficiency on the y-axis, respectively. Here, area efficiency is defined as operations per seconds (given in mega operations

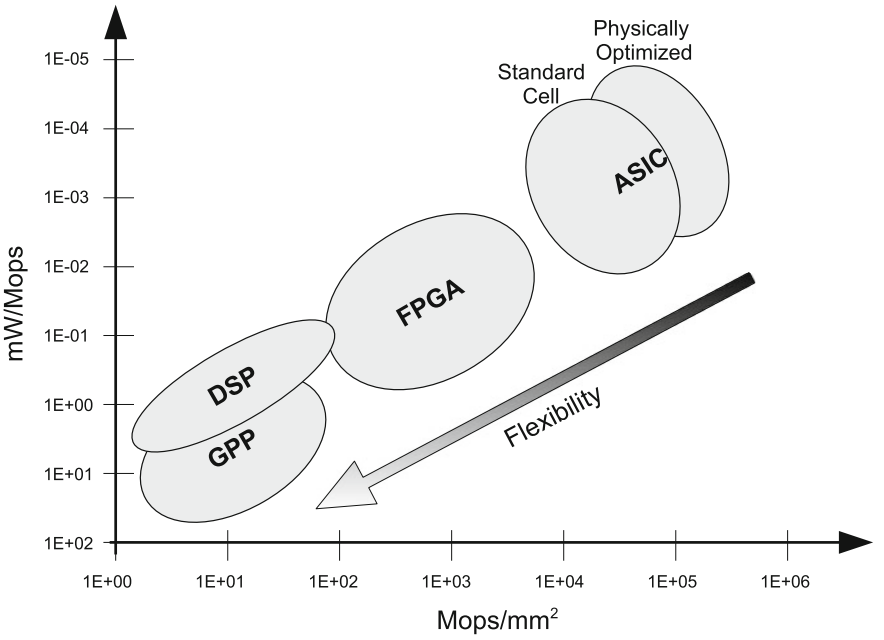


Fig. 1.2 Comparison of peak performance, power efficiency, and flexibility of different implementation styles, derived from [14]

per seconds or Mops) divided by area, while energy efficiency is defined as power consumption divided by number of operations per second.

Many different implementation styles exist between the two extremes, with respect to their power and area efficiency. For example digital signal processors (DSP) or application specific instruction-set processors (ASIP). A DSP is a processor with an instruction set optimized to perform operations for digital signal processing applications. A DSP is typically designed with a wide scope of applications in mind and the same DSP could be for example used efficiently for video processing as well as for processing tasks within a digital communication receiver. The goal of an ASIP design is to provide just enough flexibility to cover the flexibility requirements of a given application domain. It has thus an optimized instruction set to provide maximum performance at a minimum area and energy budget.

All implementation styles presented so far have a special architecture in mind, which is implemented utilizing a VLSI fabrication process. Once fabricated we cannot change the design anymore. One hardware platform which allows a reconfiguration even after fabrication is a field-programmable gate array (FPGA). FPGAs are integrated circuits which can be configured by a designer and, thus, offer huge flexibility. It is possible to map practically every architecture—e.g., an application specific design, the architecture of a general purpose processor, or an ASIP—onto an FPGA. A certain architecture mapped onto FPGAs will always have an inferior area and power efficiency compared to a direct VLSI realization. FPGAs are often utilized for rapid prototyping or smaller product series.

With every new technology process every implementation style moves as well with respect to their power efficiency and as well their peak performance. For example the required processing in a mobile phone to perform a voice call was realized in the year 2000 by dedicated hardware (ASIC). With the rapid progress in technology a phone call today is typically processed by an embedded GPU, e.g. ARM core, within a smart phone. Thus, a hardware designer implementing an identical application may have to deal with different implementation styles when moving to the next technology node.

Trends in Communication Standards

For communication systems we typically have to distinguish between different transport media over which communication takes place. There is a difference for transmissions via optical, wireline or wireless communication channels. Many more different transport media exist. The one thing uniting them is that for all of these communication media a trend to higher throughput demands can be observed. This trend for higher data throughputs is driven by requests to access data at every place, every time, and very importantly often in real time. In the following we will highlight the trends in mobile communications.

Mobile phone communications are one major technological part of our daily lives. The number of acronyms for telecommunication standards is immense, which can be

seen just by looking at the advertisement from mobile communication providers. We have to distinguish between the standardization bodies, the organizing parties, and the resulting technology generations. All of these parts exists with different naming and acronyms within Europe, north America, and Asian regions. In the following we will mainly focus on the European region.

- Standardization bodies: ETSI is responsible for standardizing information and communication technologies (ICT) in Europe. ETSI stands for European Telecommunications Standards Institute and is an independent, non-profit, standardization organization in the telecommunications industry.
- Organizing bodies: The International Telecommunication Union (ITU) is an agency of the United Nations responsible for the ICT domain and coordinates, e.g. standardization activities and as well the harmonization of international frequency-spectrum. The 3rd Generation Partnership Project (3GPP) is an association of companies, ranging from telecommunication providers to chip manufacturers. The task of the 3GPP initiative is to define and evolve the generations of mobile standards.
- Mobile standards generation: Every mobile standard generation stands for an evolution in employed techniques with respect to an efficient frequency usage and an increased throughput. The 2nd generation, for example, was the first digital cellular technology and already defined in 1989. The 3rd generation was released in 1999, the 4th generation in 2008. Each generation is still further evolved with the objective of being compatible to older generations but achieving increased data throughput. The list of acronyms is large and not all will be explained in this text, e.g. 2nd generation evolution (GSM →GPRS→EDGE→EDGE advanced), 3rd generation evolution (UMTS → HSPA → HSPA advanced) 4th generation evolution (LTE →LTE advanced).

One of the most successful, nowadays global, telecommunication standards is GSM (Global System for Mobile Communications), The world wide coverage of GSM is already larger than 70 %, i.e. more than 70 % of all people world wide have access to this mobile network, see www.gsma.com. GSM belongs to the 2nd generation of mobile communications systems and has a defined throughput of only 2kbit/s. New features and applications led to the demand of higher throughput, thus the third generation of mobile communication standards was developed. The 3rd Generation Partnership Project (3GPP) had the task to evolve the GSM standard to a 3rd generation telecommunication standard. It is worth noting that the 3GPP was just one initiative towards the 3rd generation organized in Europe, which fulfilled requirements defined by the ITU. In Europe the first 3rd generation was denoted UMTS (universal mobile telecommunication standard) and was released in the year 1999. The documents describing these standard are often denoted as Release99. Starting form this basic definition new evolved standards were developed, as for example the High speed packet access (HSPA) with higher throughput requirements, e.g. released in 2005 and 2008. In 2008 a revised mobile communications standard with new features was released, as well as a further branch, which is called long term evolution (LTE) or often as well 3.5th or sometimes directly the 4th generation.

Table 1.2 Mobile phone trends in 5-year intervals, table adapted from [16]

Year	1995	2000	2005	2010	2015	2020
Cellular standard	2G GSM	3G UMTS	3.5G HSPA	pre-4G LTE	4G LTE-A	5G
Downlink bitrate (Mbit/s)	0.01	0.1	1	10	100	1000
Workload (GOPS)	0.1	1	10	100	1000	10000
CMOS (ITRS, nm)	350	180	90	50	22	7.4
Battery energy (Wh)	1	2	3	4	5	6
Phone CPU clock (MHz)	20	100	200	500	1000	2000
Phone CPU power (W)	0.05	0.05	0.1	0.2	0.3	0.5
PC CPU power (W)	5	20	100	200	200	300

For mobile communication systems we have to distinguish between the so called uplink and downlink communications. For mobile communication typically base station handles all mobile communications within a specific area. The base station serves as a central hub in which all mobile devices have to be registered. An uplink defines the communication link from a mobile terminal to a base station while the downlink defines the communication from a base station to a mobile terminal.

Table. 1.2 shows the currently assumed mobile communication trends in 5-year intervals. The standard documents for the different releases and thus years of approval can be found at www.3gpp.org. It can be seen that the throughput demands of a new generation is rapidly increasing, with an approximative increase of 10 times within 5 years teme interval. Note that an identical trend can be seen as well for wireline communications, as shown in [15].

The presented table shows the trend for the data rates, it tells us nothing about the new techniques and algorithms which have to be realized to enable this throughput increase.

Design Challenges

Table 1.2 is adapted form [16] and comprises three important aspects: the trend in CMOS technology, the trend for downlink throughputs, and the trend of the workload caused by the employed algorithms for transmission and reception.

The trend in downlink throughput and CMOS integration was already described. Here, additional information is given, like the clock frequency and the power consumption of the CPUs in a personal computer (PC) and a mobile phone. This is related to the already discussed implementation styles, where one is optimized for high performance, the other for low power applications, respectively.

Especially for mobile phones it is assumed that the total battery energy given in [Wh] will increase only moderately over the years, while the workload demand is increasing rapidly. One reason is the already mentioned increased throughput

demand, which is also linked to the number of pixels that can be displayed on modern high resolution screen.

The major bottleneck will be the productivity of a designer. The designer has to map a complex algorithm down to an architectural design. A typical top down design flow, may result in an inefficient VLSI performance as measured by chip area and power consumption. Top down means the separation of algorithm designer and hardware designer in terms of programming or design language. The algorithm designer programs the basic functionality in C, C++, or Matlab. Taking the resulting program, the hardware designer utilizes a design language like VHDL or Verilog, for the architecture design. This clear separation of algorithm and hardware design may lead to inefficient designs and results as well in-productivity in terms of design time.

Two promising approaches to enhance the productivity are summarized: The first possibility is the introduction of so called high-level synthesis, a research domain which has received a lot of attention in recent years. A tool featuring high-level synthesis will improve the productivity, since the design will be described in a high-level language, e.g. C++ or Java. The mapping to an architecture will be done automatically. This will be a big benefit for the designer, since designing an architecture in VHDL is time consuming as well as error prone, due to the fact that the reference system is programmed in another language than the architectural description. It is like writing assembler compared to writing C. However, the inherent parallelism of a hardware design poses big challenges for the high-level design flow and sometimes as well for the designer. A second possibility to close the productivity gap can be already observed during the standardization process of LTE or DVB-S2 standard. In these standards, algorithmic parts were introduced with an efficient implementation strategy in mind. The mapping from algorithmic level down to hardware is then predefined and will, thus, result in efficient designs.

Both methods to improve the productivity require knowledge of the application as well as a basic understanding of hardware design. For the sake of simplicity we focus in this manuscript on individual components within a larger receiver realization. Already for a simple component design a comprehensive understanding of algorithm and architecture becomes mandatory to achieve efficiency in communications performance and VLSI performance. The next section gives an overview of the application which serves as an example throughout the rest of this manuscript.

1.1 Application: Outer Receiver

Designing a transceiver or receiver of a digital transmission system requires an abstract model of the transmission chain. The model of the digital transmission chain approximates the real world behavior of sending information from one point to some other physical location, as shown in Fig. 1.3. The transmission of information form one source to one sink is typically denoted as point-to-point communication. The research of the basics of discrete information processing has started mainly with the

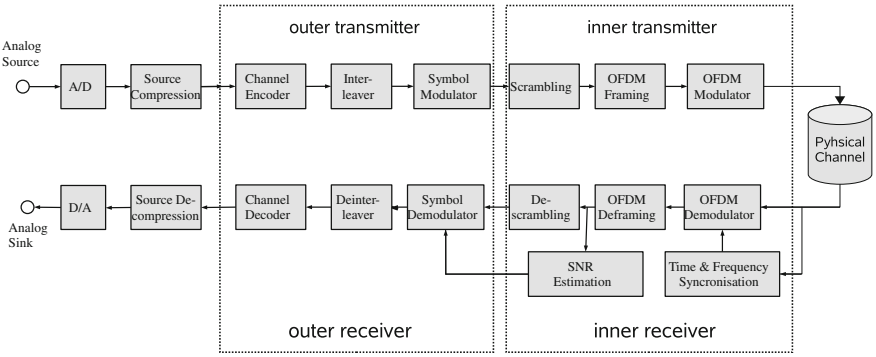


Fig. 1.3 Transmission system, only the outer transceiver part is discussed in the following chapters

Nyquist criterion, already formulated in 1928. Nyquist stated the basic principle that band-limited signals can be fully characterized in a discrete time bases. An analog signal has to be sampled at a rate of f_S Hz, which must be greater than twice the highest frequency component in the analog signal. The resulting time discrete stream of continuous (real-valued) samples, contains all information of the band-limited analog signal. These real values are then converted to the digital domain by a quantizer (A/D unit), see Fig. 1.3. The resulting time and value discrete samples are passed to the next stage. From now on the discrete stream can be represented, stored and processed in a binary form, where each discrete sample may be represented by an n-bit binary word. The so called source compression removes redundant information from this binary sample stream. The output of the source compression can be modeled as a sequence of statistically independent symbols, in which each symbol has an equal likelihood of appearance. Thus, the entire part of analog source, A/D conversion, and source compression can be modeled as a binary random source with 0 and 1 to be equally likely.

Transmitting this binary information via a communication channel can result in bit errors (or bit flips) in the received bit sequence. The flipped bits are caused by the noise of the channel, which is especially large for wireless channels. Thus, additional stages are mandatory to ensure a reliable transmission of the information. Typically, we can split the transmitter and receiver into two parts: the inner and outer transceiver part as indicated by the box in Fig. 1.3. The task of the inner transceiver is to present a good time discrete channel from its outer transmitter input to the outer receiver output. The task of the outer transceiver is to transmit the information at the highest practical data rate.

In the following we roughly sketch the parts of Fig. 1.3. The first stage of the outer transmitter is the channel encoder. The binary input stream is passed to the channel encoder, which adds additional information to the data stream, which it generates in a deterministic manner from the input bits. This additional information is called redundancy. The channel decoder on the receiver side will exploit this redundant information to detect and correct errors which occurred during transmission.

The output sequence of the channel encoder is interleaved, i.e. the positions of the bits within the sequence are permuted. The symbol modulator or higher-order modulation maps bits to a real-valued symbol.

Thus, the output of the outer transceiver is a sequence of real-valued symbols, in which each symbol comprises the information of one or more bits. The task of this symbol modulation is to increase the so called spectral efficiency. The spectral efficiency gives a measure for the number of bits that are transmitted per unit of bandwidth. The inner receiver now has the task to allocate the symbol information to a certain carrier frequency.

The symbol stream input to the inner transmitter is first passed to a scrambler unit. This unit ensures that undesirable symbol sequences (e.g. long sequences of zeros) will be eliminated. This output stream is regrouped, which is then passed to the carrier modulator. Typically, to transmit information via a physical channel we modify one characteristic, e.g. the amplitude, of a higher frequency signal, the so called carrier. This carrier modulation is done in a systematic manner depending on the data we would like to transmit. Here, a so called orthogonal frequency division multiplexing (OFDM) modulator is indicated which is one possibility to modulate the information on the carries. An OFDM technique is for example utilized in the LTE downlink scenario, while in UMTS a so called code division multiple access (CDMA) scheme is employed, instead. Many different setups exist for the inner transmitter, some of which also have additional stages. The detailed composition of the inner transmitter depends on the carrier modulation technique as well as the type of communications channel.

The channel, the medium of transmission, will disturb the sent signals, e.g. by superposition of waveforms from other transmitters, or by simple reflections of the original signal. Thus we will receive a disturbed information, with disturbances across multiple symbols. The inner receiver has the task to separate the superimposed received signals into individual symbols of information. The separation of the individual symbols can only be accomplished if the channel conditions are good. The obtained individual time discrete samples are passed to the outer receiver (after re-framing and descrambling). Often, a major problem is the correct synchronization in time and frequency to ensure the correct separation of the symbols. Furthermore, we have to extract the corresponding signal-to-noise (SNR) which has to be passed to the outer receiver, as well.

The outer receiver will obtain a sequence of symbols which is still corrupted by discrete noise. The discrete noise reflects the imperfect separation process. Ensuring a nearly correct input data stream to the source decoder, is the goal of the outer receiver. The outer receiver has the task to provide the best guess about the sent information with respect to its input information. The outer receiver with the individual components of demodulation, deinterleaving, and channel decoding will be the application in this manuscript.

The output of the outer receiver has to fulfill a certain type of quality. Depending on the type of application the nearly correct input data stream for the source decoder may mean for example 1 faulty bit out of 100 bits for voice decoders, or 1 faulty bit out of 10000 bits for MPEG decoders. A very important factor for the entire

Table 1.3 Selection of communication standards and their channel codes with maximum number of coded information bits and the maximum defined throughput

Standard	Channel codes	Num. information bits	Throughput
GSM	Convolutional codes	Up to 876	Up to 12 kbit/s
EDGE	Convolutional codes	Up to 870	Up to 384 kbit/s
UMTS	Convolutional codes	Up to 504	Up to 32 kbit/s
	Turbo codes	Up to 5114	Up to 2 Mbit/s
HSPA	Turbo codes	Up to 5114	Up to 14.4 Mbit/s
LTE	Turbo codes	Up to 6144	Up to 300 Mbit/s
LTE-A	Turbo codes	Up to 6144	Up to 1000 Mbit/s
CDMA2000	Convolutional codes	Up to 744	Up to 28 kbit/s
	Turbo codes	Up to 20730	Up to 2 Mbit/s
IEEE802.11n	Convolutional codes	Up to 4095	Up to 450 Mbit/s
	LDPC codes	Up to 1620	Up to 450 Mbit/s
IEEE802.16e	Convolutional codes	Up to 864	Up to 75 Mbit/s
(WiMax)	Turbo codes	Up to 4800	Up to 75 Mbit/s
	LDPC codes	Up to 1920	Up to 75 Mbit/s

transmission is the resulting ratio of transmit power to noise power, often called signal-to-noise ratio (SNR).

For every component in this transmission chain many design options exist. For example for the channel encoder there exist a many different channel codes, throughput demands, and flexibility requirements. Table. 1.3 shows a selection of wireless standards and the utilized channel codes. While convolutional codes (CC) are utilized in practically all communications standards, turbo codes (TC), and low-density parity-check codes (LDPC) are employed in standards only since the mid of the nighties. CC, TC and LDPC codes are discussed in Chaps. 3, 6, and 7 respectively.

1.2 Dependencies and Overview of Chapters

This manuscript addresses students in electrical engineering, computer engineering, and computer science with an interest in the field of communications engineering, architectures, and microelectronic design. Figure 1.4 shows the overview of the chapters and the dependencies. An according lecture is given at the University of Kaiserslautern and the Karlsruhe Institute of Technology, both Germany. The numbering indicates the current sequence of lectures, each with a durtion of 2 h. The manuscript puts a special focus on implementation aspects and implementation constraints of individual components that are needed for outer transceiver design of current standards like UMTS, LTE, WiMAX and DVB-S2. White boxes in Fig. 1.4 indicates content related to communications engineering, gray boxes are related to the design or architectures. The application domain is the so called outer receiver, which comprises the channel coding, interleaving stages, modulator, and multiple antenna

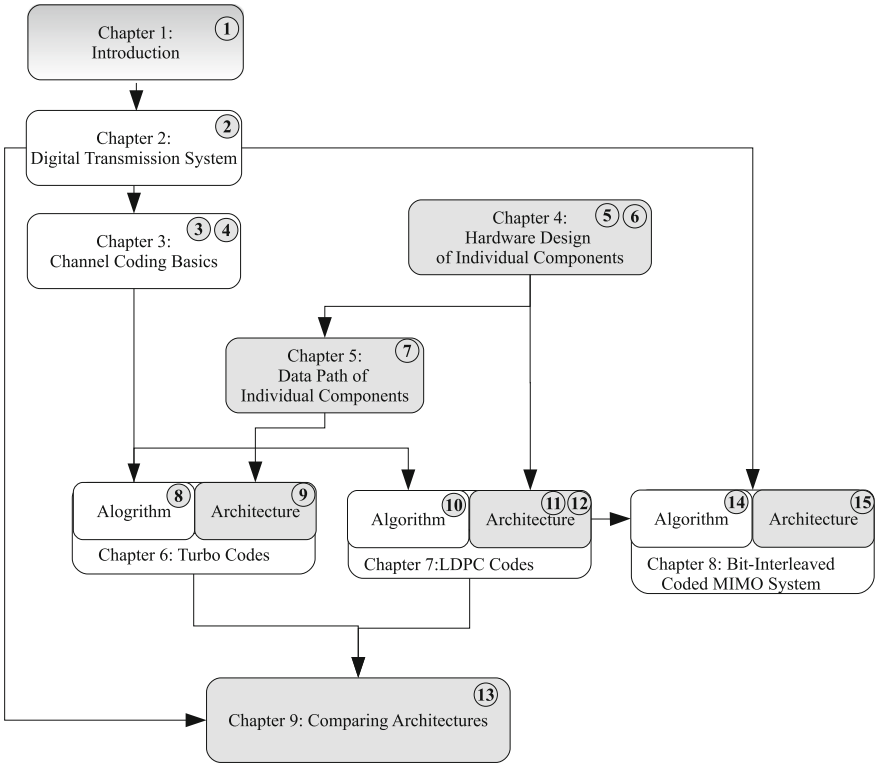


Fig. 1.4 Structure of the treatise and dependencies of the chapters. The numbers indicate the sequence of the lectures. *White boxes* indicate content related to communications engineering, *gray boxes* are related to the design or architectures

transmission. Throughout the manuscript the focus lies on advanced algorithms that are currently in use in modern communications systems. Their basic principles are always derived with considering the resulting communications **and** implementation performance.

The individual chapters and their dependencies are described in the following:

- **Chapter** 2 introduces the basic transmission scheme which is utilized throughout this manuscript. The so called bit-interleaved coded modulation (BICM) scheme is explained together with its most important channel model, the additive white Gaussian noise (AWGN) channel. This type of channel is explained and its capacity discussed. The channel capacity is by definition the maximum amount of information for a reliable transmission over this channel. By using multiple antenna systems we can establish a channel with high capacity. Its basic concept is sketched shortly.
- **Chapter** 3 introduces the linear block codes, which are the basis of nearly all channel coding schemes in use in current communication standards. The

general decoding problem is addressed, which is to solve the maximum likelihood (ML) criterion. The ML solution is the most likely sequence which was probably sent. Sometimes an additional probability term for each output result is required. Then the symbol-by-symbol maximum a posteriori (MAP) criterion has to be solved. Examples are presented how to calculate the ML and MAP solutions. The important class of convolutional codes is introduced and the decoding algorithm is derived. All introduced algorithms are derived using examples and already in a form which is suitable for a hardware realization.

- The basic design flow for the design of individual components is described in **Chap**. 4. Memories are one major building block of hardware systems and are investigated in this chapter, with a special focus on static random access memories (SRAM). The high level design of SRAMs is reviewed with area and power trends for different SRAM instances. Interleaving is one typical stage within a communication system. However, the realization of a parallel interleavers poses challenges for the architecture and especially for the utilization of the SRAM memories. It is worth noting that such a simple functionality poses big problems. Here, in Sect. 4.3 for the first time in this manuscript a joint design of algorithm and hardware is introduced. The same methodology was utilized for designing LTE turbo codes.
- The data path defines the alignment of processing units. The steps to design a data path of individual components are introduced in **Chap**. 5. The data path of a so called forward-backward algorithm is derived for two different levels of parallelism: one which results in a serial architecture and one which results in a parallel architecture. Processing units have to be designed to realize the correct functionality. Complicated functions can either be realized by look-up tables or by deriving an approximation which can be realized in hardware efficiently. Examples of both techniques are presented.
- **Chapter** 6 introduces turbo codes and the corresponding encoders with respect to communication standards. The iterative decoding process of turbo codes is described and the communications performance is shown, taking into account the quantization effects of fixed-point calculations. The importance of an SNR insensitive algorithm is demonstrated which reflects a more realistic instantiation within a full system. The basic component of a turbo decoder is a maximum a posteriori (MAP) decoder. Its most common realization is shown which is linked to the previously derived data path of a forward-backward algorithm. The chapter also summarizes the essential questions, which are most commonly used to drive industrial design process for turbo decoders and recap the huge possibilities within the architectural design space exploration.
- The basic communications performance of low-density parity-check codes and the architectural constraints on theses codes are presented in **Chap**. 7. For the decoding of a WiFi code communication performance results are given and compared to an optimal ML decoding, which demonstrates the quality of the utilized iterative decoding algorithm. The joint design of decoder architecture and channel code—a key enabling method to efficient decoder designs for e.g. DVB-S2, WiFi, and WiMax—is highlighted. The essential questions for the design of LDPC decoders

and the design space exploration are given as well. The chapter closes with a design study for a high throughput LDPC decoder.

- LTE specified a new technique which transports information by employing multiple antennas for transmission as well as reception. These so called multiple input and multiple output (MIMO) systems are characterized by a high computational complexity for the outer receiver, as well. Design challenges to realize the resulting MIMO-BICM systems are highlighted. An example of joint design of MIMO detection and LDPC code design is presented which will reduce the implementation complexity and improve the communications performance. Even though the example applies only one idea with respect to a certain MIMO detection algorithm, it highlights the strengths of joint design of algorithm and architecture. This is a common motif throughout the manuscript.
- Meaningful efficiency metrics are required to explore and evaluate the huge resulting design space. In **Chap**. 9 different metrics are introduced and discussed. It will be shown that suitable energy and area efficiency metrics are based on decoded information bit per energy and throughput per area unit. This chapter demonstrates that the understanding of the trade-offs between implementation efficiency, communications performance and flexibility is key for designing efficient baseband receivers.

References

1. Hennessy, J.L., Patterson, D.A.: Computer Architecture a Quantitive Approach. Morgan Kauffmann Publishers, Inc., San Francisco (1996)
2. Rabaey, J.M., Chandrakasan, A., Nikolic, B.: Digital Integrated Circuits—A Design Perspective, 2nd edn. Prentice Hall, Upper Saddle River (2004)
3. Ashenden, P.J.: The Designer's Guide To VHDL, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2002)
4. Rabaey, J.M.: Low Power Design Essentials, 1ed edn. Springer, New York (2009)
5. Moon, T.K.: Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, Hoboken (2005)
6. Meyr, H., Moeneclaey, M., Fechtel, S.A.: Digital Communication Receivers. Wiley, New York (1998)
7. MacKay, D.: Information Theory, Inference, and Learning Algorithms. Cambridge University Press, Cambridge (2003)
8. Lin, S. Jr, Costello, D.J.: Error Control Coding, 2nd edn. Prentice Hall PTR, Upper Saddle River (2004)
9. Bossert, M.: Kanalcodierung, 2nd edn. B.G. Teubner, Stuttgart (1998)
10. International Technology Roadmap for Semiconductors 2011: ITRS home page. http://public.itrs.net
11. Moore, G.E.: Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff. IEEE Solid-State Circuits Newsl. **20**(3), 33–35 (2006). doi:10.1109/N-SSC.2006.4785860
12. ARM Ltd. http://www.arm.com
13. ARM Ltd. Cortex-A5. http://www.arm.com/products/processors/cortex-a/cortex-a5.php. Accessed Feb 2012

14. Blume, H., Feldkaemper, H.T., Noll, T.G.: Model-based exploration of the design space for heterogeneous systems on chip. In: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, vol. 1, pp. 29–40 (2002). doi:10.1007/s11265-005-4936-4
15. Transition to 4G: 3GPP Broadband Evolution to IMT-Advanced. http://www.rysavy.com
16. van Berkel, C.H.: Multi-core for mobile phones. In: Proceedings of the Design, Automation. Test in Europe Conference and Exhibition (DATE '09), pp. 1260–1265 (2009)

Chapter 2
Digital Transmission System

Throughout the manuscript, we use as application the outer receiver, which is a part of the complete transmission chain as shown in Fig. 1.3. The model for the entire transceiver chain can be very complex. Thus it is essential to make further abstractions of this transmission chain model. As mentioned, the task of the inner transceiver is to create a good time discrete channel from the inner transmitter input to the inner receiver output. Assuming a perfect synchronization of the demodulator in time and frequency we can abstract the outer transmitter chain by a simple baseband transmission chain as shown in Fig. 2.1. The term baseband means that there is no frequency component present in the model. The final simple baseband model features a binary random source and a binary sink, which replaces the source compression/decompression of Fig. 1.3. This outer transceiver chain composed of channel encoder, interleaver, and modulator is called bit-interleaved coded modulation (BICM) and is often utilized in communication systems.

In the following we describe the nomenclature of the variables shown in Fig. 2.1. Most channel codes used in digital communications systems, such as the linear block codes that we explain in Chap. 3, operate on vectors. The vector input to the channel encoder is $\boldsymbol{u}$ with the vector $\boldsymbol{u} = [u_0, u_1, \ldots, u_{K-1}]$ of length K. $\boldsymbol{u}$ is called *information word*, with $u_k \in \{0, 1\}$ being the *information bits*. The purpose of the channel encoder is to introduce, in a controlled manner, redundancy to the information word $\boldsymbol{u}$. The redundancy can be used at the receiver side to overcome the signal degradation encountered during the transmission through the discrete channel. The output of the channel encoder is the *codeword* $\boldsymbol{x} = [x_0, x_1, \ldots, x_{N-1}]$ of length $N > K$. The interleaver permutes the sequence of the bits in the codeword, creating the interleaved codeword $\boldsymbol{x}'$. The interleaving is explained in more detail in Sect. 4.3.

This sequence is mapped by the digital modulator onto a sequence of symbols $\boldsymbol{s}$. We can always map groups of Q bits on one modulation symbol. Thus, there exist 2^Q distinct symbols which we can transmit via the channel. One of the most common modulation schemes is the *Binary Phase Shift Keying* (BPSK) with $Q = 1$, where $x \in \{0, 1\}$ is mapped to $s \in \{-1, +1\}$. The final mapped transmit sequence $\boldsymbol{s}$ of length N_s is then disturbed by noise. For wireless transmission and the already

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_2, © Springer Science+Business Media New York 2014

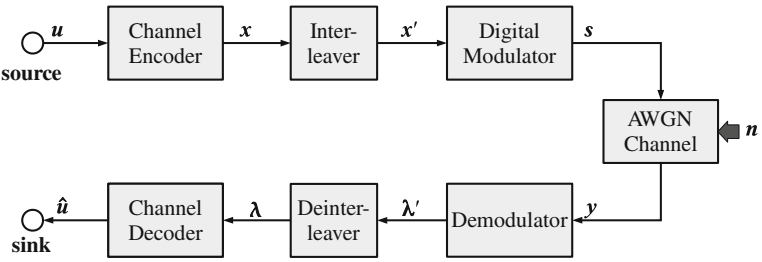


Fig. 2.1 Digital transmission system model of the outer transceiver

mentioned perfect synchronization, we model the noise as *additive white Gaussian noise* (AWGN) with zero mean and variance σ^2. This channel model is used for all simulations in this manuscript. The received vector $\boldsymbol{y}$ results in

$$\boldsymbol{y} = \boldsymbol{s} + \boldsymbol{n} \tag{2.1}$$

with $\boldsymbol{n}$ the noise vector which is added to the sent transmit sequence.

The task of the demodulator is to transform the disturbed input sequence of symbols $\mathbf{y}$ into a sequence of bit probabilities. This sequence $\boldsymbol{\lambda}'$ contains likelihood values, indicating the probability for each bit in the output of the channel encoder having been sent as a 1 bit or a 0 bit. This is done by calculating the conditional probability for every bit position $i \in 0, N-1$ under the condition that $\boldsymbol{y}$ was received. $\boldsymbol{\lambda}'$ is de-interleaved to $\boldsymbol{\lambda}$ which means we restore the original index positions of the channel decoder output. Note, that the prime to indicate an interleaved version of a sequence is omitted in the following. The channel decoder now has the task to detect and correct occurring errors. Thus it calculates an estimation of the sent information bits $\hat{\boldsymbol{u}}$. The AWGN channel and the demodulator calculations are explained in Sects. 2.1 and 2.2 respectively.

The modeling of different stages within the outer receiver chain requires a basic understanding of probability theory. For example, the model of a channel requires a probabilistic description of the effects of noise caused by the environment. Many more stages like the demodulator or the channel decoder deal with probabilities, as they calculate estimations of which information was most likely sent given the received disturbed input information. All decoding methods for LDPC codes, turbo codes, and convolutional codes were originally derived in the probability domain. The basic probability terms, which introduce terms like joint probability distribution, conditional probability, and marginalization are described in Appendix C. The probability terms are as well required to describe the information content which can be transmitted via an AWGN channel. This is denoted as channel capacity and is introduced in Sect. 2.3. The amount of information we can transmit via a channel can even be increased by using multiple antennas at the transmitter and/or receiver side. This so called MIMO transmission is introduced in Sect. 2.4. Especially the MIMO transmission with respect to a BICM transmission system will be further addressed in Chap. 8.

2.1 Channel Model

Typically, real world channels have complicated characteristics with all kind of interferences, e.g multi-path propagations, frequency selectivity and so on. However, as already mentioned, the task of the inner receiver is to provide a discrete channel to the outer receiver. The remaining discrete channel is modeled as additive white Gaussian noise (AWGN) channel, which is the most important channel model for the examination of channel codes. The AWGN channel model has no memory and adds to each random input variable a random noise variable N with density function:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(\mathrm{x} - \mu)^2\right\} \tag{2.2}$$

The noise variable is Gaussian distributed with variance σ^2 and a mean value $\mu = 0$, see Sect. C: The output of the time discrete channel for each time step i is:

$$y_i = s_i + n_i. \tag{2.3}$$

In the following a *Binary Phase Shift Keying* (BPSK) modulation is assumed with $s \in \{-\sqrt{E_S}, \sqrt{E_S}\}$. $\sqrt{E_S}$ represents the magnitude of the signal such that the signal energy results in E_S. The received samples y_i are corrupted by noise. When we plot the distribution of the received samples two independent Gaussian distributions occur. The two distributions are centered at their respective mean value and are shown in Fig. 2.2. The three sub-figures show the different distributions for different values of the variance σ^2 and $\mu = \pm 1$ respectively. Depending on the variance of the additive Gaussian noise the overlapping of the two distributions, the gray shaded area in Fig. 2.2, changes.

One way to estimate the received bits ($\hat{x}$) is to apply a threshold detection at the 0 level. If the received information $y_i \geq 0$ then $\hat{x}_i = 1$, and vice versa. The resulting errors are indicated by the gray shaded areas in Fig. 2.2. Depending on the variance of the Gaussian distribution the possible error region gets smaller or larger. We define the ratio of incorrectly received bits to the total number of received bits as the *bit-error rate* (BER).

We can calculate the probability of errors for the presented binary transmission by using two shifted Gaussian distributions:

$$p(y|s = \sqrt{E_s}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y - \sqrt{E_s})^2\right\} \tag{2.4}$$

and

$$p(y|s = -\sqrt{E_s}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y + \sqrt{E_s})^2\right\}. \tag{2.5}$$

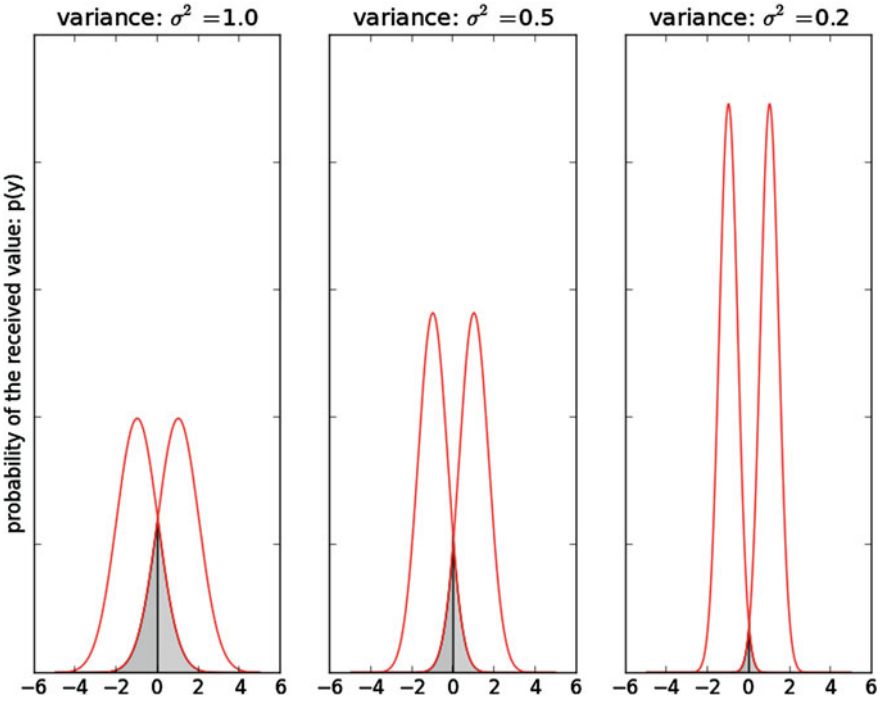


Fig. 2.2 Distributions of received values y with BPSK modulation for different variances of the noise

As mentioned, E_S is the transmission energy of one symbol, while the noise energy (N_0) is part of the channel properties. The noise energy is directly related to the variance of the Gaussian distribution, which is given by:

$$\frac{N_0}{2} = \sigma^2. \tag{2.6}$$

The quantity $\frac{N_0}{2}$ is the two-sided noise power spectral density. The probability of an error for the binary detection can be calculated using the Q-function.

$$Q(t) = Q(N > t) = \frac{1}{\sqrt{2\pi}} \int_t^{\infty} \exp\left\{-n^2/2\right\} \mathrm{dn}. \tag{2.7}$$

The integral evaluates the probability that the random variable N is larger than a threshold t. A Gaussian distribution with zero mean and variance one is assumed in this function. However, for our transmission system we have to include the corresponding mean and variance values.

The resulting error rate when transmitting symbols with $s = -\sqrt{E_s}$ is

$$Q(Y > t) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_t^{\infty} \exp\left\{-\frac{1}{2\sigma^2}(y + \sqrt{E_s})^2\right\} dn = Q\left(\frac{t + \sqrt{E_s}}{\sigma}\right). \tag{2.8}$$

The probability of a bit error assuming BPSK is the weighted probability of both possible transmissions ($s = \pm\sqrt{E_s}$), i.e.

$$P_b = Q\left(\frac{t + \sqrt{E_S}}{\sigma}\right) P\left(-\sqrt{E_S}\right) + Q\left(\frac{\sqrt{E_S} - t}{\sigma}\right) P\left(\sqrt{E_S}\right). \tag{2.9}$$

Both BPSK symbols are transmitted equally likely, $P\left(\sqrt{E_S}\right) = P\left(-\sqrt{E_S}\right)$, while the threshold t for a detected bit error lies at $t = 0$ as shown in Fig. 2.2. Thus P_b evaluates to:

$$P_b = Q\left(\frac{\sqrt{E_S}}{\sigma}\right) = Q\left(\sqrt{\frac{2E_S}{N_0}}\right). \tag{2.10}$$

For the evaluation of the obtained results, typically graphs as shown in Fig. 2.3 are used. Shown is the bit error rate on the y-axis in logarithmic scale, the x-axis represents the signal-to-noise ratio in E_S/N_0 for an AWGN channel.

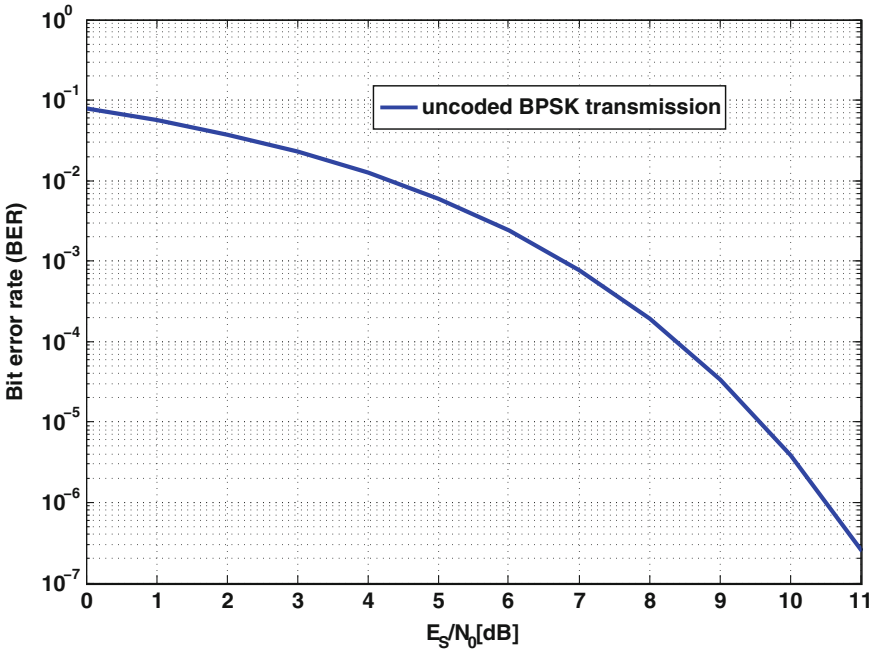


Fig. 2.3 Bit error rate for a BPSK transmission without channel coding (uncoded)

Quality of Service

For many services, data are packed into frames (blocks, packets,…). For packet based transmission like LTE up to 25 codewords are grouped into one transmission time interval frame (TTI), which implies a higher-level grouping of bits with additional header information. However, in the context of channel coding in the baseband model of Fig. 2.1, the terms frame, block, and packet are used interchangeably and all denote one transmitted codeword. Often it is irrelevant how many bits could be received correctly within one frame. It only matters whether the frame contains errors or not, as in many services every erroneous frame is considered lost, including all the data therein. This leads to the ratio of the number of erroneous frames to the total number of frames, the *frame-error rate* (FER). Both BER and FER are usually expressed in the logarithmic domain:

$$BER|_{dB} = \log\left(\frac{\text{number of erroneous bits}}{\text{total number of bits}}\right), \tag{2.11}$$

and

$$FER|_{dB} = \log\left(\frac{\text{number of erroneous frames}}{\text{total number of frames}}\right). \tag{2.12}$$

BER and FER always have to be put in relation to the noise present in a channel. The relative amount of noise is expressed by the *signal-to-noise ratio* (SNR).

$$\text{SNR} = \frac{E_S}{N_0}. \tag{2.13}$$

The energy needed for the transmission of one information bit (E_b) is generally not equal to E_S. E_b incorporates also information about the *code rate (R)* and the number of modulation bits Q. The code rate is defined as the ratio of information bits per coded bit. A code rate $R = 1/3$ for example denotes that three bits are transmitted to communicate one information bit. The number of modulation bits determines how many bits are grouped to one modulation symbol. For the already mentioned BPSK modulation this is $Q = 1$. E.g. in the case of $Q = 2$ we use $2^2 = 4$ different symbols for transmission, see Sect. 2.2. The SNR can consequently also be expressed in relation to the information bit energy instead of the symbol energy. The relation between the two forms of expression is:

$$\frac{E_S}{N_0} = Q \cdot R \cdot \frac{E_b}{N_0}, \tag{2.14}$$

or in the logarithmic domain:

$$\left.\frac{E_S}{N_0}\right|_{dB} = 10 \cdot \log(Q) + 10 \cdot \log(R) + \left.\frac{E_b}{N_0}\right|_{dB}. \tag{2.15}$$

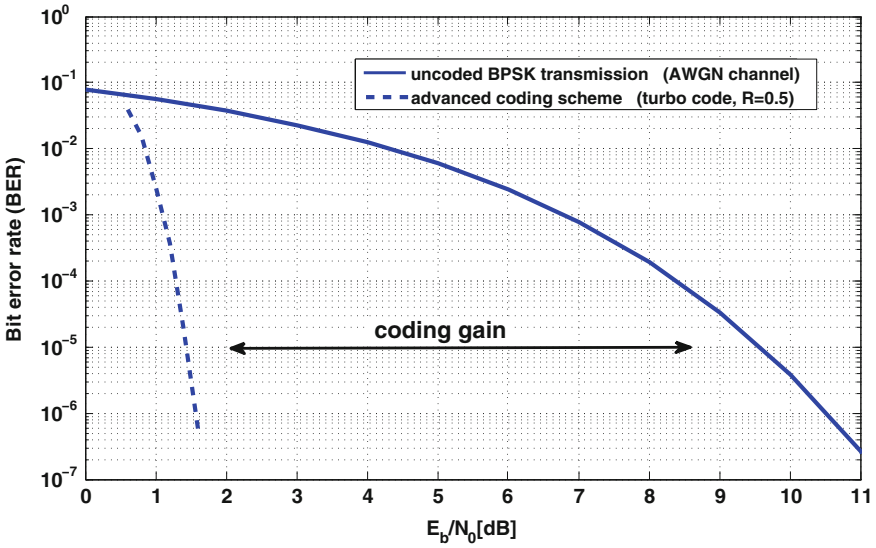


Fig. 2.4 Bit error rate for a BPSK transmission with and without channel coding

For comparing different channel coding schemes the required SNR level has to be evaluated with respect to a certain frame error rate. In order to obtain a statistic for the frame error rate at a given signal-to-noise ratio so called Monte Carlo simulations can be performed. According to the modeling of Fig. 2.1 a large number of frames are simulated, always at one fixed signal-to-noise ratio and one fixed channel coding setup (type, block length, code rate). Then, the simulations are repeated for different noise levels, by changing the variance of the Gaussian noise.

Figure 2.4 shows again the bit error rate on the y-axis in logarithmic scale, while the x-axis represents the signal-to-noise ratio in E_b/N_0 for an AWGN channel. Shown is the BER for an uncoded system and a system using a channel code, a so called turbo code. Turbo codes are introduced in Chap. 6 and are defined in the 3GPP communication standards. A code rate of $R = 0.5$ is used which means two bits are transmitted for each information bit. Even though transmitting more coded bits, the energy required to transmit an information bit reliable is lower than in the uncoded case. We can see a large gain in terms of signal-to-noise ratio to obtain a specific BER. In other words: With respect to a higher noise level we can obtain the same quality-of-service when using a channel coding scheme. This gain between coded and uncoded transmission is denoted as coding gain.

For turbo codes we transmit the information in blocks and we should use the resulting frame error rate to judge on the achieved communications performance. Figure 2.5 shows FER curves over SNR for the same kind of turbo code, however, applying different code rates for transmission.

All simulations have a fixed number of information bits of $K = 6144$, while the codeword size ranges from $N = 18432$ to $N = 6400$ bits, which corresponds to

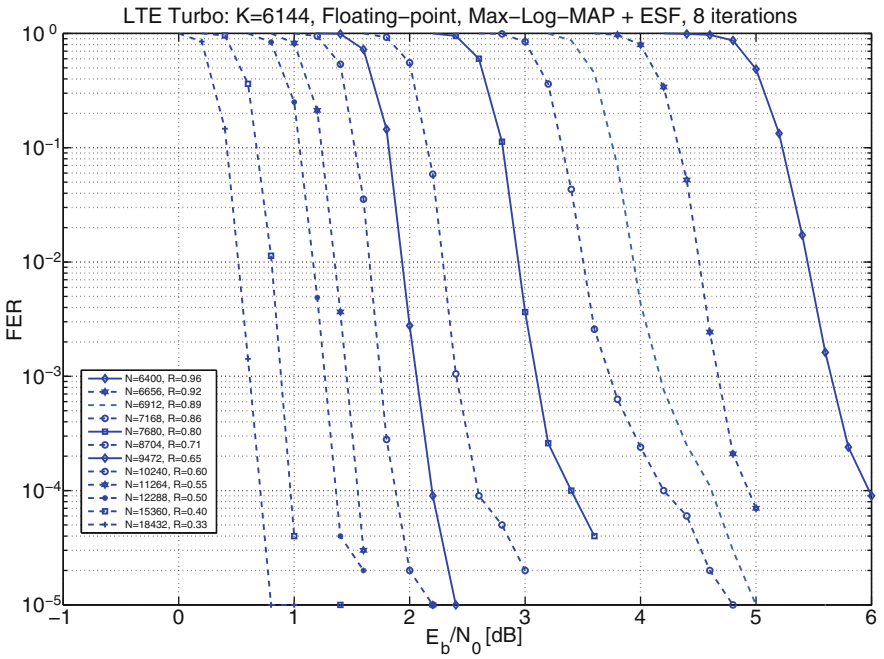


Fig. 2.5 FER vs. SNR for different code rates of a LTE turbo code

code rates of $R = 1/3$ to $R = 0.96$, respectively. For each E_b/N_0 and each code rate 100000 frames were simulated, in order to get statistically stable results.

All curves show a characteristic drop of the observed FER for an increased SNR. All code rates have a $FER = 1$ at a very low E_b/N_0 ratio. E.g. at $E_b/N_0 = 0$dB all simulations have at least one bit error per frame after the channel decoder. It can be seen that the lower the code rate, the smaller the SNR level at which a certain FER can be obtained. It is really important that we compare the SNR in terms of E_b/N_0, since only then a comparison of different code rates and the evaluation of the final achieved coding gain is possible. When comparing the coding gain of two coding schemes, here two different code rates, we always need a certain FER as reference. For practical wireless transmission the reference frame error rate is often $FER = 10^{-2}$ or $FER = 10^{-3}$.

2.2 Digital Modulator and Demodulator

The modulator maps groups of Q bit to one symbol, before transmitting these via the channel. Task of this modulation is the increase of sent information per channel usage.

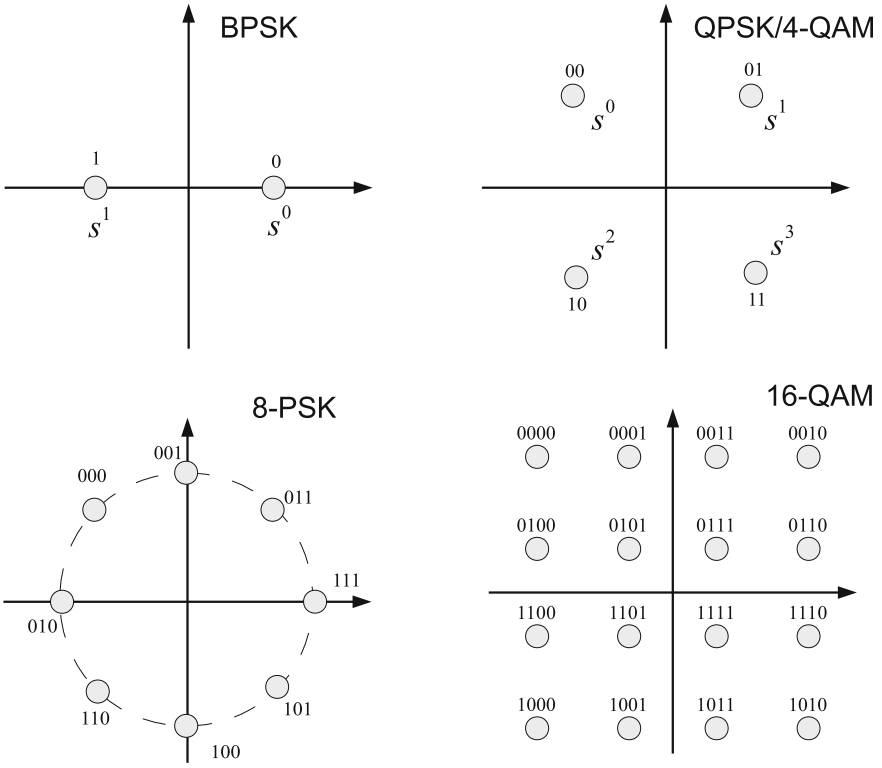


Fig. 2.6 Different modulation types: binary phase shift keying, 4-QAM, 8

Figure 2.6 shows different signal constellations which can be used for the modulation. 2^Q different signal points exist, depending on the number of mapped bits. For the phase-shift keying with 2, 4, or 8 signal points (BPSK, QPSK, 8-PSK) each signal point has the same energy E_S. The figure shows as well a quadrature amplitude modulation (QAM) with 16 signal points. The energy for different signal points may vary for QAM modulation types, while the average signal energy $\bar{E}_S$ will be normalized, with

$$\bar{E}_S = \frac{1}{2^Q} \sum_{k=0}^{2^Q-1} (Re(s_k)^2 + Im(s_k)^2). \tag{2.16}$$

The amplitudes $(Re(s), Im(s))$ of the complex constellation points are chosen accordingly to ensure $\bar{E}_S = 1$. The bit to symbol mapping shown in Fig. 2.6 is a so called Gray mapping. A Gray mapping or Gray coding ensures that two neighboring symbols differ in only one bit. This type of mapping is most often used in communication systems which use the introduced bit-interleaved coded modulation transmission.

The demodulator is the first stage after the channel model, see Fig. 2.1. The input to the demodulator is the sequence of received values $\boldsymbol{y}$, where each value y_i comprises the information of Q bits corrupted by noise.

The task of the demodulator is to calculate for each of these Q bits, an estimate of the originally sent bit. These bit estimates can be given as so-called hard-values or soft-values.

- **Hard-values**: These values are represented by just one bit per transmitted bit. For the BPSK example in Fig. 2.2 this would be the sign bit of the received information, which can by extracted in this case by a simple threshold decision.
- **Soft-values**: These values include a probability information for each bit. Thus we pass a sign value and an associated confidence information, which is derived from the received sample.

The advantage of the soft-values is an increased possible coding gain of the channel decoder. This gain is up to 3 dB when comparing channel decoding with hard-input or soft-input information, respectively [1]. Nearly all modern channel decoders require soft-values as input information.

In order to obtain soft-values we have to calculate the a posteriori probability (APP) that the bit x_i (or symbol s_j) was transmitted under the condition that y_j was received. Also possible to calculate a symbol probability $P(s_j|y_j)$ it is often required to calculated directly the corresponding bit probability $P(x_i|y_j)$. This is due to the BICM system at which the bit information is interleaved to ensure independent probabilities at the channel decoders input. For a simple modulation scheme with one bit mapped to one symbol we will calculate one a posteriori probability. In the case of a higher mapping we have to extract Q APP values, one for each mapped bit. Thus, the demodulator has to calculate for one received symbol y_j:

$$P(x_i|y_j), ..., P(x_{i+Q-1}|y_j) \tag{2.17}$$

We can not directly calculate this probability, but instead we have to use Baye's theorem, which then results in:

$$P(x_i|y_j) = P(y_j|x_i) \cdot P(x_i)/P(y_j) \tag{2.18}$$

$P(x_i|y_j)$ is denoted as a posteriori probability, $P(y_j|x_i)$ as conditional probability, and $P(x_i)$ as a priori information. The calculation of $P(y_j|x_i)$ and $P(y_j)$ depends on the channel characteristics, while $P(x_i)$ reflects an a priori information about the source characteristics. For the demodulation process we do not calculate the probability of $P(x_i|y_j)$ directly; rather we operate in the so called log-likelihood domain, which calculates

$$\lambda(x_i|y_j) = ln\frac{P(y_j|x_i = 0) \cdot P(x_i = 0)/P(y_j)}{P(y_j|x_i = 1) \cdot P(x_i = 1)/P(y_j)} \tag{2.19}$$

$P(y_j)$ depends only on the channel characteristics and is a constant for a certain received y_j. The term cancels out while the calculation can be split into two parts.

$$\lambda(x_i|y_j) = ln\frac{P(y_j|x_i=0)}{P(y_j|x_i=1)} + ln\frac{P(x_i=0)}{P(x_i=1)} = \lambda(y_j|x_i) + \lambda(x_i) \tag{2.20}$$

For binary sources without a priori information, i.e., with an assumed equal distribution of 1s and 0s, the probabilities $P(x_i = 1) = P(x_i = 0)$ are equivalent and thus cancel out (i.e, $\lambda(x_i) = ln\frac{P(x_i=0)}{P(x_i=1)} = 0$).

Log-likelihood ratios (LLR) have several advantages over operating on the probabilities directly. These advantages become evident for evaluating $\lambda(y_j|x_i)$ when taking the characteristics of a channel into account, e.g. an AWGN channel. The mathematical operations for BPSK demodulator is presented in the following. The demodulation for a 4 amplitude modulation is presented in Appendix D, respectively.

BPSK Demodulator

In the case of BPSK modulation only one bit is mapped to every symbol s. Typically the mapping is $x_i = 0 \rightarrow s_i = +1$ and $x_i = 1 \rightarrow s_i = -1$. For each received value one LLR value is calculated, which is here denoted with $\lambda(y|x)$. For the sake of simplicity, we omit the indices $i = j$ for the BPSK demodulation. In order to calculate $\lambda(y|x)$ under the assumption of an AWGN channel we have to evaluate the Gaussian density function with a shift of its mean value according to the amplitude of the transmitted symbol s, thus:

$$p(y|s) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{-\frac{(y-s)^2}{2\sigma^2}\right\}. \tag{2.21}$$

For the BPSK mapping given above the computation of the demodulator results in:

$$\begin{aligned}
\lambda(y|x) &= ln\frac{P(y|s=+1)}{P(y|s=-1)} \\
&= ln\frac{exp\left\{-\frac{1}{2\sigma^2}(y-1)^2\right\}}{exp\left\{-\frac{1}{2\sigma^2}(y+1)^2\right\}} \\
&= \frac{1}{2\sigma^2}(y+1)^2 - \frac{1}{2\sigma^2}(y-1)^2 \\
&= \frac{2}{\sigma^2}y
\end{aligned} \tag{2.22}$$

Thus the channel LLR value $\lambda = \frac{2}{\sigma^2} y$ of the received symbol is just the received sample corrected by the *channel reliability factor* $L_{ch} = \frac{2}{\sigma^2}$.[1] We can summarize the advantage of the log-likelihood logarithm of Eq. 2.20 for this demodulator example: Only the exponents are of interest, while all normalizers of the density function $(1/\sqrt{2\pi\sigma^2})$ cancel out. Furthermore, in the logarithmic domain multiplications and divisions turn into additions, and subtractions, respectively, which is of great advantage for a hardware realization.

2.3 Channel Capacity

As mentioned before, the goal of the channel code is to transmit the information bits reliably through a discrete unrealiable channel at the highest practicable rate. The channel capacity gives us the bound for a reliable (error free) transmission with respect to a given signal-to-noise ratio. For evaluating this question we have to introduce the definition of the information content I. The information content of a discrete event x is defined as:

$$I(x) = \log_2 \frac{1}{P(x)}, \tag{2.23}$$

with $P(x)$ the probability of the single event x, see Appendix C. The logarithm of base two constraints this information measure to the dimension 'bit'. This is the smallest and most appropriate unit, especially when dealing with digital data. Often we are interested in the average information content of a discrete random variable X with probability density function $p(x)$. This is denoted as entropy H and can be calculated via the expectation with respect to I:

$$H(X) = E\left\{\log_2 \frac{1}{p(x)}\right\} = \sum_{x \in \Omega_x} p(x) \log_2 \frac{1}{p(x)}. \tag{2.24}$$

$p(x)$ defines the probability of the random variable X, with Ω_x defining the respective discrete sample space. The entropy of the random variable X is maximized when each element occurs equally likely. Assuming Ω_x has 2^K elements, the entropy results in:

$$H(X) = \sum_{i=0}^{2^K-1} \frac{1}{2^K} \log_2 \frac{1}{\frac{1}{2^K}} = 2^K 2^{-K} K = K \ \ [bit]. \tag{2.25}$$

Assuming two discrete random variable X and Y we can define a joint entropy:

[1] A channel model related to the AWGN channel is the so called fading channel. In this channel model each received value has an additional attenuation term a, and the channel reliability factor turns into $L_{ch} = a\frac{2}{\sigma^2}$, with a being the fading factor.

$$H(X;Y) = E\left\{\log_2 \frac{1}{p(x,y)}\right\} = \sum_{x\in\Omega_x}\sum_{y\in\Omega_y} p(x,y)\log_2\frac{1}{p(x,y)}. \tag{2.26}$$

The joint entropy is very important since we can describe the dependency of the information between two variables. Its relation is closely linked to Bayes theory, see Eq. C.9. However, since the entropy is defined in the logarithm domain, additions instead of multiplications result within the chain rule:

$$H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X) \tag{2.27}$$

The mutual information $I(X;Y)$ gives us the information that two random variables X and Y share. If this shared information is maximized we have as much knowing of one variable to give information (reduce uncertainty) above the other. Figure 2.7 shows the dependencies between the mutual information and the entropies. The illustration shows a possible transmission of variable X with entropy $H(X)$ via a channel. Information will get lost during transmission which is expressed as conditional entropy (equivocation) $H(X|Y)$. In turn irrelevant information will be added $H(Y|X)$. The final received and observable entropy is $H(Y)$. The mutual information of two random variables can be calculated by:

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y). \tag{2.28}$$

We can define the mutual information as well directly via the joint and marginal probabilities

$$I(X;Y) = \sum_{x\in\Omega_x}\sum_{y\in\Omega_y} p(x,y)\log_2\frac{p(x,y)}{p(x)\cdot p(y)}. \tag{2.29}$$

Note that the mutual information of the input X and the output Y of a system can be estimated by Monte Carlo simulations. By tracking the density functions by histogram measurements we can evaluate Eq. 2.29. This is a pragmatic approach especially when the density functions are not entirely known. As mentioned before, the channel capacity gives an upper bound for a reliable (error free) transmission

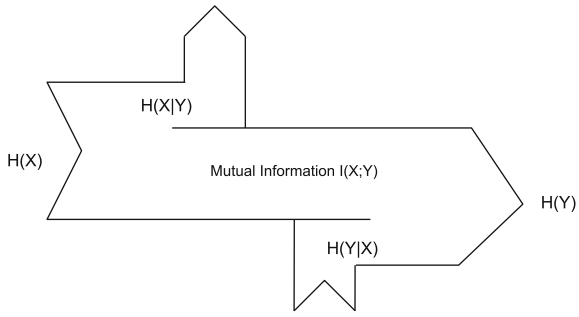


Fig. 2.7 Illustration of the mutual information and its dependencies

with respect to a given signal-to-noise ratio. Assuming a random input variable X and the observation Y, the capacity of the utilized channel is its maximum mutual information

$$C = \sup_{p(x)} I(X;Y). \tag{2.30}$$

Equation 2.30 requires the knowledge of the joint distribution of X and Y and the distribution and the channel. Three assumptions have to be met to achieve the maximum capacity which is denoted as Shannon limit:

- the input variable X requires to be Gaussian distributed,
- the noise has to be additive with Gaussian distribution,
- the length of the input sequence has to be infinite.

In the case of an AWGN channel with a constant noise power of N_0 the channel capacity can be evaluated to [2]:

$$C = \log_2\left(1 + \frac{E_S}{N_0}\right) = \log_2\left(1 + \frac{RE_b}{N_0}\right). \tag{2.31}$$

When using a transmission rate $R < C$ an error probability as low as desired can be achieved. With Eq. 2.31 we can evaluate the bound for the required bit energy to noise power by assuming the code rate to be equal to the capacity

$$\frac{E_b}{N_0} = \frac{2^R - 1}{R}. \tag{2.32}$$

The Shannon limit gives a bound for the minimum bit energy required for reliable transmission assuming an infinite bandwidth, i.e. the code rate approaches zero ($R \to 0$). The absolute minimum bit energy to noise power required for reliable transmission is $E_b/N_0 = -1.59\,\text{dB}$.

The bandwidth B is limited for many communications systems and quite expensive from an economic point of view. The total noise power is given by $N = B \cdot N_0$, the total signal energy $S = R_b E_b$ is defined via the data rate R_b. R_b is the number of bits we can transmit within a certain amount of time. According to the Nyquist criterion this will be $2BT$ symbols within a time unit of T. The channel capacity for a channel with limited bandwidth B can be expressed as

$$C_B = B \cdot \log_2\left(1 + \frac{S}{N}\right) \quad [bits/s]. \tag{2.33}$$

We can see that it is possible to trade off bandwidth for signal-to-noise ratio. The introduced capacity before, without bandwidth, defines the spectral efficiency. This spectral efficiency gives us the number of bits per second per Hertz $[bits/s/Hz]$ which is an important metric since it defines the amount of information that can be transmitted per Hertz of bandwidth.

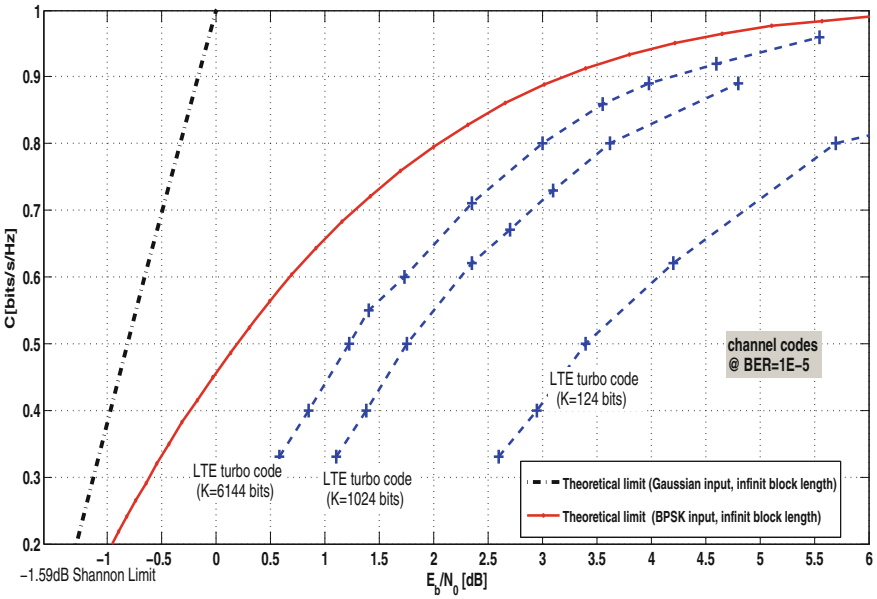


Fig. 2.8 Spectral and power efficiency for Shannon limit, theoretical limit applying BPSK modulation, and turbo codes with different block sizes

Figure 2.8 shows the spectral efficiency versus the signal-to-noise ratio E_b/N_0. Plotted is the optimum theoretical limit which can only be reached for Gaussian distributed values given as input to a channel. By constraining the channel input to be binary (e.g. Binary Phase Shift Keying (BPSK) [3] the resulting capacity is lower than the optimum capacity. The random input variable S has a sample space of $s \in \{-1, +1\}$ while the output is modeled as a continuous random variable $y \in R$. Then the mutual information evaluates to:

$$I(S;Y) = \frac{1}{2} \sum_{s_i \in \{+1,-1\}} \int_{-\infty}^{+\infty} p(y|s_i) \cdot \log_2 \left(\frac{2p(y|s_i)}{p(y|s_i = -1) + p(y|s_i = +1)} \right) dy \tag{2.34}$$

The resulting theoretical limit for BPSK input is shown as well in Fig. 2.8.

The channel capacity is the tight upper bound for a reliable transmission, however, it tells us not how to construct good codes which can approach this limit. Furthermore, the limit is only valid for infinite block sizes. Therefore, Fig. 2.8 shows coding schemes with different block lengths and coding rates. The utilized coding schemes here are turbo codes utilized in the LTE standard.

All codes are plotted at a reference bit error rate of $BER = 10^{-5}$. The results are collected from different simulations over a simple AWGN channel and BPSK modulation, with K the number of information bits ($K = 124$, $K = 1024$, $K = 6144$).

The different code rates are obtained by a so called puncturing scheme defined in the communication standard. The simulated LTE turbo codes are introduce in Chap. 6. One important aspects can be seen in this figure. The larger the block length the smaller the gap to the theoretical limit. For block sizes of $K > 100k$ bits the gap to the theoretical limit is less than 0.2 dB.

2.4 Multiple Antenna Systems

Multiple antenna system is a general expression for a transceiver system which relies on multiple antennas at the receiver and/or at the transmitter side. The major goal of using a multiple antenna system is to improve the system performance. This could either mean the transmission of more information for a given amount of energy, or to achieve a better quality of service for a fixed amount of energy [4]. By using multiple antennas different type of gains can be obtained.

- Array gain: The array gain can increase the signal-to-noise ratio by a coherent combination of the received signal. Achieving the coherent detection we need the knowledge of the channel to align the phases of signals. The average SNR grows proportionally to the number of received signals, i.e., multiple receive antennas collect more signal energy compared to a single receive antenna.
- Diversity gain: Diversity gain is obtained when transmitting information multiple times via different transmission channels. Different channels may underly a different fading, which means, the attenuation of the channel path will change. The assumption is that different paths do not fade concurrently. The receiver will gather the diversified information and has then the possibility to mitigate the fading effects. The higher the number of paths, the higher the probability to mitigate occurring fading effects. There exist different sources of diversity: diversity in time (also called temporal diversity), diversity in frequency, and diversity in space. Temporal diversity can be introduced in an artificial way by the transmitter, by transmitting the same information at different points in time. Frequency diversity can be utilized when transmitting at different frequencies. Spatial diversity uses the distance between receive or transmit antennas. The goal of diversity is always to improve the reliability of communication. The maximum diversity gain we can obtain is of $d = M_T M_R$th order, i.e. each individual path can be exploited. M_T defines the number of transmit antennas, M_R defines the number of receive antennas, respectively. The diversity d is defined as the slope of the average error probability at high signal-to-noise ratio, i.e., the average error probability can be made to decay like $1/SNR^d$ [5].
- Multiplexing gain: Maximum gain in spatial multiplexing is obtained when at each time step and each antenna a new symbol is transmitted. The MIMO detector on the receiver side has the task to separate the symbols which are interfered and corrupted by noise. MIMO channels offer a linear increase $r = min(M_T, M_R)$ in capacity [5]. It was shown in the high SNR regime that the rate of the system (capacity)

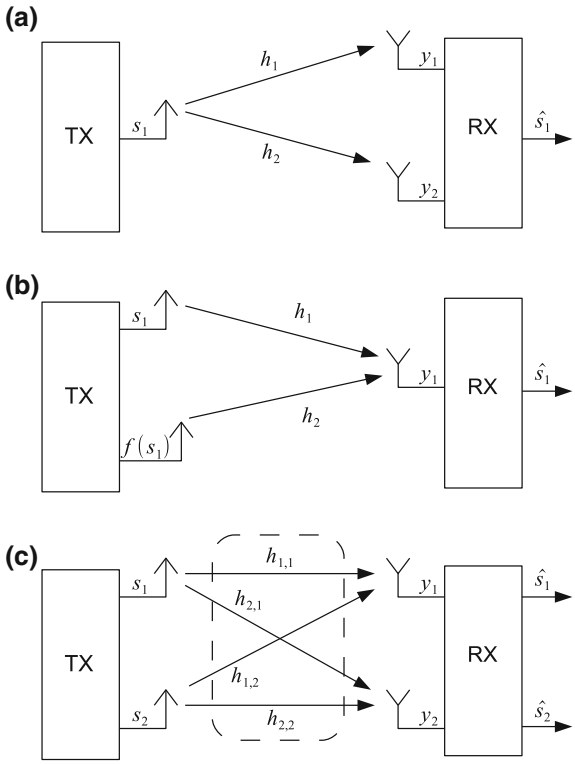


Fig. 2.9 Three different multiple antenna systems

can grow with $r\ log(SNR)$. The factor r is denoted as spatial multiplexing gain and defines the number of degrees of freedom of a system (bit/s/Hz). There exists a trade off between gain in diversity and gain in spatial multiplexing [5].

Three major scenarios exist to set up a multiple antenna system which are shown in Fig. 2.9. All three possibilities are used in the LTE communication standard, a good overview can be found in [6]. Here, only the obtained diversity or multiplexing gain is highlighted with respect to the different antenna constellations.

Single-Input Multiple-Output Transmission

The most straight forward system using multiple antennas is the single-input multiple-output (SIMO) system which is shown in Fig. 2.9a. The receiver collects two versions of the sent symbol s_1 which is denoted as receive diversity. The channel coefficients h_1 and h_2 define the different attenuations of both paths. The highest degree of diversity is obtained when the channel coefficients are uncorrelated. Assuming M_R receive antennas we can obtain diversity of order M_R. The SIMO system reflects a typical point-to-point transmission from a mobile device to a base station (uplink).

Multiple-Input Single-Output Transmission

Multiple antennas can be applied as well at the transmitter which the goal to obtain transmit diversity, see Fig. 2.9b. Again, to maximize the diversity the channel coefficients have to be uncorrelated. This can be achieved by an appropriate distance between transmit antennas. Diversity can be in addition introduced by applying a function to the sent information, denoted here as $f(s_1)$. The function indicates the introduction of a time or frequency offset. One prominent example is the so called space-time coding [7] which is as well introduced in the LTE communication standard. The point-to-point multiple-input single-output (MISO) system can be efficiently exploited in the downlink of wireless transmission systems.

Multiple-Input Multiple-Output Transmission

MIMO systems can achieve either a diversity gain and/or a multiplexing gain. Especially the multiplexing gain is of large interest. For achieving the highest multiplexing gain the symbol stream is demultiplexed to multiple transmit antennas. At each transmit antenna and in each time slot we transmit independent symbols. The receiver side collects the superposed and noise disturbed samples from multiple receive antennas. This spatial multiplexing yields in a linear increase of system capacity, i.e. the spectral efficiency is increased by a factor of $min(M_T, M_R)$ for a given transmit power. LTE defines a spatial multiplexing scheme using 4 transmit and 4 receive antennas (4×4) while applying a 64-QAM modulation. The number of pits per second per Hertz would be 24 [bits/s/Hz] without channel coding. However, spatial multiplexing should be combined with a channel coding scheme which is often done according to a BICM system as shown in Fig. 2.1.

Ergodic Channel and Quasi-Static Channel

Figure 2.9c shows a two-by-two antenna system. The transmission of a data and the received vector can be modeled in general as:

$$\boldsymbol{y}_t = \boldsymbol{H}_t \cdot \boldsymbol{s}_t + \boldsymbol{n}_t, \tag{2.35}$$

with $\boldsymbol{H}_t$ the channel matrix of dimension $M_R \times M_T$ and $\boldsymbol{n}_t$ the noise vector of dimension M_R. The entries in $\boldsymbol{H}_t$ can be modeled as independent, complex, zero-mean, unit variance, Gaussian random variables [8]. The presented channel model is again a time discrete model. In this model fading coefficients are represented by the matrix $\boldsymbol{H}_t$ which is in contrast to the already introduced AWGN channel in which no fading coefficients are present. The average transmit power E_S of each antenna is normalized to one, i.e. $E\{\boldsymbol{s}_t \boldsymbol{s}_t^H\} = \boldsymbol{I}$ [2]. We assume an additive Gaussian noise at each

[2] $\boldsymbol{I}$ is the identity matrix and $()^{\boldsymbol{H}}$ denotes the Hermitian transpose.

receive antenna, with $E\{\boldsymbol{n}_t \boldsymbol{n}_t^H\} = N_0 \boldsymbol{I}$. The signal-to-noise power at each receive antenna can be expressed as $SNR = \frac{M_T E_S}{N_0}$. We have to distinguish two major cases for simulating the MIMO channel:

- Quasi-static channel: $\boldsymbol{H}_t$ remains constant for a longer time period and is assumed to be static for one entire codeword of the channel code.
- Ergodic channel: $\boldsymbol{H}_t$ changes in each time slot and successive time slots are statistically independent.

Assuming a quasi-static channel we cannot guarantee an error free transmission since we cannot average across multiple channel realizations. The fading coefficients of $\boldsymbol{H}_t$ could by chance prohibit a possible correction of the sent signal. Another definition of the channel capacity has to be used which is called outage capacity [9]. The outage capacity is associated with an outage probability P_{out} which defines the probability that the current channel capacity is smaller than the transmission rate R.

$$P_{out} = P(C(\rho) < R) \tag{2.36}$$

$C(\rho)$ defines the capacity of a certain channel realization with the signal-to-noise ρ of the current fading coefficients. The outage probability can be tracked via Monte-Carlo simulations. The capacity C of a given channel model differs for a quasi-static or ergodic channel.

For ergodic channels the capacity is higher as shown in [9]. The capacity C for an ergodic channel can be calculated as expectation via the single channel realizations:

$$C = E\left\{\log_2 \det\left(\boldsymbol{I} + \frac{SNR}{M_T} \cdot \boldsymbol{H_t} \boldsymbol{H_t^H}\right)\right\} \quad [bits/s/Hz]. \tag{2.37}$$

SNR is the physically measured signal-to-noise ratio at each receive antenna. Assuming an ergodic channel the channel capacity increases linearly with $min(M_T, M_R)$. Transmission rates near MIMO channel capacity can be achieved by a bit-interleaved coded MIMO system. The resulting BICM-MIMO systems and the challenges for designing an appropriate MIMO detector are shown in Chap. 8. A comprehensive overview of MIMO communications can be found in [8].

References

1. Bossert, M.: Kanalcodierung, 2nd edn. B. G. Teubner, Stuttgart (1998)
2. Moon, T.K.: Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, Hoboken (2005)
3. Proakis, J.G.: Digital Communications, 3rd edn. McGraw-Hill, Inc., Boston (1995)
4. Boelcskei, H.: Fundamental tradeoffs in MIMO wireless systems. In: Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless, Communication, vol. 1, pp. 1–10 (2004). doi:10.1109/CASSET.2004.1322893
5. Zheng, L., Tse, D.N.C.: Diversity and multiplexing: a fundamental tradeoff in multiple-antenna channels. IEEE Trans. Inf. Theory **49**(5), 1073–1096 (2003). doi:10.1109/TIT.2003.810646

6. Dahlman, E., Parkvall, S., Skoeld, J., Beming, P.: 3G Evolution. Elsevier, HSPA and LTE for Mobile Broadband, Oxford (2008)
7. Alamouti, S.M.: A simple transmit diversity technique for wireless communications. IEEE J. Sel. Areas Commun. **16**(8), 1451–1458 (1998)
8. Paulraj, A.J., Gore, D.A., Nabar, R.U., Bölcskei, H.: An overview of MIMO communications—a key to gigabit wireless. Proc. IEEE **92**(2), 198–218 (2004). doi:10.1109/JPROC.2003.821915
9. Telatar, I.: Capacity of multi-antenna Gaussian channels. Eur. Trans. Telecommun. **10**, 585–595 (1999)

Chapter 3
Channel Coding Basics

This chapter gives a short overview of the basic principle of linear block codes and possible decoding methods. As this treatise only deals with a subset of existing channel codes, only a short primer with small examples are given. For an in depth study many good books are available, like [1] and [2].

When speaking about channel coding we have to distinguish between the code definition itself and the associated decoding algorithm which has to calculate an estimation of the transmitted information based on the sequence of received symbols. The goal of a practical system is to establish a channel coding scheme which has the theoretical capability to approach the Shannon limit. However, just as important is an associated decoding method that can be implemented in hardware.

All described codes here in this chapter and in this manuscript are linear block codes. The basic terms and overview of channel codes are introduced in Sect. 3.1, linear block codes are presented in Sect. 3.2. The general decoding problem -trying to solve the maximum likelihood (ML) criterion—is addressed in Sect. 3.3. For the decoding algorithm we have to distinguish between a soft-input and hard-input information as explained in Sect. 2.2. Soft-input information refers to the LLR values for each received sample, provided by the demodulator. All decoding algorithms presented in this chapter require these LLR values as input, furthermore the decoding algorithms are derived in the logarithm domain which enables an efficient decoder hardware realizations.

Solving the ML criterion is an NP complete problem, thus in practical applications we can often only approximate the ML result by utilizing heuristics. These iterative heuristics are mandatory in the case of turbo and LDPC code decoding. When applying iterative decoding we have to solve the symbol-by-symbol maximum a posteriori (MAP) criterion which is explained in Sect. 3.3.3. One important class of channel codes is the class of convolutional codes. These are introduced in Sect. 3.4, an example of a decoding algorithm which approximates the MAP criterion is derived in Sect. 3.4.2.

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_3, © Springer Science+Business Media New York 2014

3.1 Overview Channel Coding

The goal of a channel code is to transmit information reliably via an unrealiable channel. To achieve this we map an information word $\boldsymbol{u}$ into a codeword $\boldsymbol{x}$. Throughout this manuscript we assume binary information for the elements $u_i, x_i \in [0, 1]$. Table 3.1 shows an exemplary binary code with $K = 3$ information bits which are mapped onto $N = 8$ distinct codewords. The code C is given by K basis vectors of length N. The cardinality of this code is $|C| = 2^3$. The process of mapping an information word to a code word is called encoding. A channel code defines only the mapping from $\boldsymbol{u}$ to $\boldsymbol{x}$, it does not define a possible decoding procedure.

One important measure for the quality of the code is its minimum (Hamming) distance. The Hamming distance is defined as the distance between two binary vectors, i.e.

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=0}^{N-1} |x_i - y_i|. \tag{3.1}$$

The minimum distance is an important measure for the quality of a code. The presented channel code in this example is a linear code, i.e., a channel code is linear if the addition of two codewords yields again a valid codeword. The minimum distance of a linear code is the minimum weight of a codeword:

$$d_{min} = \min_{\forall \boldsymbol{x} \in C} \sum_{i=0}^{N-1} x_i. \tag{3.2}$$

The code of Table 3.1 has a minimum Hamming distance of $d_{min} = 3$.

During transmitting of the codeword $\boldsymbol{x}$ errors may occur. Suppose we have transmitted one codeword of Table 3.1 and we receive a (hard) bit vector $\boldsymbol{y} = [111100]$. By using nearest neighbor decoding we can measure the distance of the received (corrupted) vector to the possibly sent codewords. The distances for all 8 possible codeword are: $d(\boldsymbol{x_0}, \boldsymbol{y}) = 4$, $d(\boldsymbol{x_1}, \boldsymbol{y}) = 1$, $d(\boldsymbol{x_2}, \boldsymbol{y}) = 3$, $d(\boldsymbol{x_3}, \boldsymbol{y}) = 2$, $d(\boldsymbol{x_4}, \boldsymbol{y}) = 2$, $d(\boldsymbol{x_5}, \boldsymbol{y}) = 3$, $d(\boldsymbol{x_6}, \boldsymbol{y}) = 5$, $d(\boldsymbol{x_7}, \boldsymbol{y}) = 4$. The codeword $\boldsymbol{x_1}$ has the smallest distance

Table 3.1 A binary code with $K = 3$ information bits and $N = 6$ codeword bits

Information word	Codeword
$\boldsymbol{u_0}$=[0 0 0]	$\boldsymbol{x_0}$=[0 0 0 0 0 0]
$\boldsymbol{u_1}$=[0 0 1]	$\boldsymbol{x_1}$=[1 1 0 1 0 0]
$\boldsymbol{u_2}$=[0 1 0]	$\boldsymbol{x_2}$=[0 1 1 0 1 0]
$\boldsymbol{u_3}$=[0 1 1]	$\boldsymbol{x_3}$=[1 0 1 1 1 0]
$\boldsymbol{u_4}$=[1 0 0]	$\boldsymbol{x_4}$=[1 1 1 0 0 1]
$\boldsymbol{u_5}$=[1 0 1]	$\boldsymbol{x_5}$=[0 0 1 1 0 1]
$\boldsymbol{u_6}$=[1 1 0]	$\boldsymbol{x_6}$=[1 0 0 0 1 1]
$\boldsymbol{u_7}$=[1 1 1]	$\boldsymbol{x_7}$=[0 1 0 1 1 1]

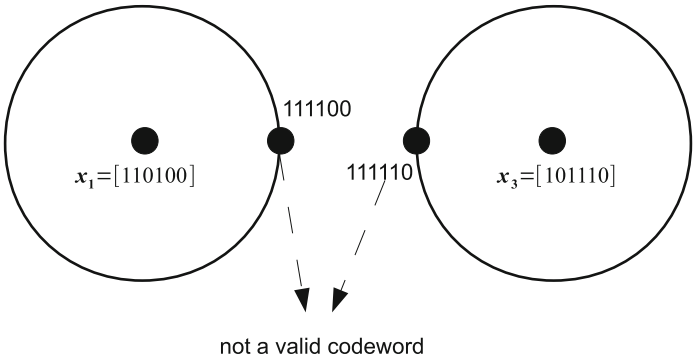


Fig. 3.1 Hamming distance between two valid codewords

to the received information $\boldsymbol{y}$ and is thus the codeword that was most likely sent by the transmitter.

The importance of a large minimum distance is shown in Fig. 3.1. It shows two codewords $\boldsymbol{x_1}$ and $\boldsymbol{x_3}$ with the Hamming distance of $d(\boldsymbol{x_1}, \boldsymbol{x_3}) = 3$. The non-overlapping spheres around the codewords indicate the decision region for a particular codeword, i.e. vectors inside this region can be clearly associated to a codeword. For a code with a minimum distance d_{min} we have a guaranteed error correction capability of:

$$t = \left\lfloor \frac{d_{min}}{2} \right\rfloor, \tag{3.3}$$

with t the number of errors occurred in the received vector. $\lfloor a \rfloor$ defines the next integer which is lower than or equivalent to a. The error detection capability of the code is $d_{min} - 1$, i.e., it can detect any invalid sequence as long as this sequence is not a codeword. Thus, the worst thing that can happen during transmission is an error sequence which results in another valid codeword, i.e. $\boldsymbol{y} = \boldsymbol{x_i} + \boldsymbol{e} = \boldsymbol{x_j}$, with $\boldsymbol{x_j} \in C$. The error sequence is represented here by an error vector $\boldsymbol{e}$ with $e_i \in \{0, 1\}$. An one entry indicates that the corresponding bit position is flipped. One goal when designing channel codes is to ensure a minimum distance as large as possible which will enable a high error correction capability.

3.2 Linear Block Codes

A binary linear block code with cardinality $|C| = 2^K$ and block length N is a K dimensional subspace of the vector space $\{0, 1\}^N$ defined over the binary field ($GF(2)$). It is possible to define a linear block code as well over larger field, e.g. extension fields $GF(2^m)$. However, here we only address binary codes. The operations with respect

Table 3.2 GF(2) addition (XOR)

+	0	1
0	0	1
1	1	1

Table 3.3 GF(2) multiplication (AND)

*	0	1
0	0	0
1	0	1

to $GF(2)$ are shown in Tables 3.2 and 3.3. Additions and multiplications result in simple *XOR* and *AND* operations.

The linear code C is given by K basis vectors of length N which are represented by a $K \times N$ matrix $\mathbf{G}$ (generator matrix). The encoding process can be described by a multiplication with the generator matrix G. Thus the encoder evaluates:

$$\mathbf{x} = \mathbf{uG} \tag{3.4}$$

The output of the encoder is the *codeword* $\boldsymbol{x}$. Most of the practically codes utilized in communication standards are linear codes. Wireless communications systems utilize packet based transmission techniques, thus linear block codes that are defined on vectors fits well to theses systems.

Equivalently, a code C can be described by a parity check matrix $\boldsymbol{H} \in \{0, 1\}^{M \times N}$ where $M = N - K$. We thus have $\boldsymbol{x} \in C$, i.e., $\boldsymbol{x}$ is a codeword, if

$$\boldsymbol{x}\boldsymbol{H}^T = \mathbf{0}. \tag{3.5}$$

The scalar product of each row $\boldsymbol{x} \cdot \boldsymbol{h_i}^T$ has to be zero. All operations are performed in the binary domain. We denote the i^{th} row and j^{th} column of $\boldsymbol{H}$ by $H_{i,.}, H_{.,j}$ respectively. $\boldsymbol{x}H_{i,.}^T = 0$ in $GF(2)$ is defined as the i^{th} parity check constraint.

The parity check constraints are used within decoding algorithms to check for a valid codeword. Assuming a vector $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{e}$ with remaining errors, the resulting codeword check will evaluated to

$$\boldsymbol{y}\boldsymbol{H}^T = \boldsymbol{x}\boldsymbol{H}^T + \boldsymbol{e}\boldsymbol{H}^T = \boldsymbol{s}. \tag{3.6}$$

$\boldsymbol{s}$ is denoted as syndrome and contains information about the current error vector. This property can be as well utilized within a decoding algorithm [2].

Since the codeword $\boldsymbol{x}$ may be derived via $\boldsymbol{x} = \boldsymbol{uG}$ one can see the important relation between generator matrix and parity check matrix which is:

$$\boldsymbol{G} \cdot \boldsymbol{H}^T = 0 \tag{3.7}$$

If the information word $\boldsymbol{u}$ is part of the codeword the code is called systematic, e.g.:

$$\boldsymbol{x} = \boldsymbol{uG} = [x_0 x_1 \ldots x_{N-1}] = [\boldsymbol{up}] = [\underbrace{u_0\, u_1 \ldots u_{K-1}}_{\text{systematic bits}}\ \underbrace{p_0 \ldots p_{M-1}}_{\text{parity bits}}] \tag{3.8}$$

The code is called non-systematic if the information vector is not part of the codeword. A linear systematic generator matrix is specified by $\boldsymbol{G} = [\boldsymbol{IP}]$, with $\boldsymbol{I}$ an identity matrix of size $K \times K$.

$$\boldsymbol{G} = \left[\begin{array}{cccc|ccc} 1 & 0 & \ldots & 0 & p_{0,0} & p_{0,1} & \cdots\ p_{0,M-1} \\ 0 & 1 & \ldots & 0 & p_{1,0} & p_{1,1} & \cdots\ p_{1,M-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & p_{K-1,0} & p_{K-1,1} & \cdots\ p_{K-1,M-1} \end{array}\right] \tag{3.9}$$

The number of parity bits M is defined by $N - K = M$. The matrix part which defines the parity constraints $\boldsymbol{P}$ is transpose within in the corresponding parity check matrix.

$$\boldsymbol{H} = [\boldsymbol{P^T I}] = \left[\begin{array}{ccc|ccc} p_{0,0} & p_{1,0} & \cdots\ p_{K-1,0} & 1 & 0 & \ldots\ 0 \\ p_{0,1} & p_{1,1} & \cdots\ p_{K-1,1} & 0 & 1 & \ldots\ 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ p_{0,M-1} & p_{1,M-1} & \cdots\ p_{K-1,M-1} & 0 & 0 & \ldots\ 1 \end{array}\right] \tag{3.10}$$

One of the first and best known linear block codes is the Hamming code.

$$\boldsymbol{G} = \begin{bmatrix} 1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} 1\,0\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1 \end{bmatrix}$$

The generator matrix enables a systematic encoding which can be seen by the identity matrix I at the left part of the matrix $G = [IP]$. With P the parity check equations for generating the corresponding parity bits. As mentioned mentioned the scalar product of each row $\boldsymbol{x} \cdot \boldsymbol{h_i}$ has to be zero, thus each code word has to fulfill the following parity check equations.

$$\begin{aligned} x_0 + x_2 + x_3 + x_4 &= 0 \\ x_0 + x_1 + x_3 + x_5 &= 0 \\ x_1 + x_2 + x_3 + x_6 &= 0 \end{aligned}$$

When ever all these parity check equations are fulfilled a valid codeword is found. This is utilized during the decoding process. This (7,4) Hamming code has a minimum distance of $d_{min} = 3$.

3.3 General Decoding Problem

The general decoding task is to detect or even correct errors which may have occurred during transmission. For deriving corresponding decoding criteria we initially ignore the result of the demodulator of Sect. 2.2 and derive the decoding task directly on the received vector $\boldsymbol{y}$. Later we will comment on the important separation of demodulator and decoding algorithm. We can distinguish two different decoding principles, which either optimizes results on the entire codeword or on each individual bit.

- Maximum likelihood criterion: solving the maximum likelihood criterion optimizes the so called codeword probability which means ML decoding picks a codeword $\hat{\boldsymbol{x}}$ which maximizes the condition probability:

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x} \in C} P\,(\boldsymbol{x} \quad \text{sent} | \boldsymbol{y} \quad \text{received}) \tag{3.11}$$

  The receiver has no knowledge about the true sent codeword $\boldsymbol{x}$. Thus the receiver algorithm for decoding may check all possible sent codewords $\boldsymbol{x} \in C$ and decides for the codeword $\hat{\boldsymbol{x}}$ which was most likely sent.
- Symbol-by-symbol maximum a posteriori criterion: solving the symbol-by-symbol MAP criterion optimizes the bit probability which means MAP decoding decides for a single bit x_i

$$\hat{x}_i = \arg\max P\,(x_i \quad \text{sent} | \boldsymbol{y} \quad \text{received}) \tag{3.12}$$

  The resulting bit estimations for the entire codeword $\hat{x}_i$ for $i \in \{1, \ldots, N\}$ will result in the optimum bit error rate. The result will not necessarily be a valid codeword. The MAP probability will be $\arg\max P\,(x_i | \boldsymbol{y})$ with $x_i \in \{0, 1\}$.

3.3.1 Maximum Likelihood (ML) Decoding

For many block based transmissions the frame error rate is of importance, thus when ever possible we should try to solve th ML criterion. In the following we will derive the ML criterion in a more evident form. Applying Bayes's rule to entire vectors results for the ML criterion in:

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x} \in C} \left(P\,(\boldsymbol{y} | \boldsymbol{x}) \cdot \frac{P(\boldsymbol{x})}{P(\boldsymbol{y})} \right) \tag{3.13}$$

$P(\boldsymbol{y})$ is the vector of channel related probability. The vector elements are constant for each received information, it will thus have no influence on the final decision. Furthermore we assume that each codeword was equally likely sent. With this assumption, the term $P(\boldsymbol{x})$ can be omitted as well. Under the assumption that each received sample is independent and with the transformation to the logarithm domain

the maximum likelihood (ML) criterion turns into

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}\in C} P(\boldsymbol{y}|\boldsymbol{x}) \tag{3.14}$$

$$= \arg\max_{\boldsymbol{x}\in C} \left(\prod_{i=0}^{N-1} P(y_i|x_i)\right) \tag{3.15}$$

$$= \underset{\boldsymbol{x}}{\in} C \arg\min \left(-ln\prod_{i=0}^{N-1} P(y_i|x_i)\right) \tag{3.16}$$

$$= \arg\min_{\boldsymbol{x}\in C} \left(-\sum_{i=0}^{N-1} lnP(y_i|x_i)\right) \tag{3.17}$$

The vector problem can be decomposed in the product form. This is only possible if the received values are independent of each other. The next two steps are the transformation in the logarithm domain and the transformation in a minimization problem which can be simply done by -1 multiplication. Note, that $P(y_j|x_i)$ is exactly the result of our demodulator in the probability domain, attention here we assumed that for each received symbol one bit exists.

It is more convenient to express the ML criterion with log likelihood values. If we add the constant $\sum_{i=0}^{N-1} lnP(y_i|0)$, i.e. the weight of the zero codeword, the problem results in

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in C} \left(\sum_{i=0}^{N-1} (lnP(y_i|x_i = 0) - lnP(y_i|x_i))\right) \tag{3.18}$$

When ever the codeword has a zero at the corresponding position $x_i = 0$ no weight will be added to the entire cost function. If the corresponding position equals $x_i = 1$ the already introduced LLR term results with $\lambda_i = ln\frac{P(y_i|x_i=0)}{P(y_i|x_i=1)}$. Now, the ML criterion can be expressed in a more elegant form with the scalar product of the LLR vector obtained of the demodulator and a possible codeword $\boldsymbol{x} \in C$. The maximum likelihood criterion is thus a minimum search via a linear cost function of the demodulator output and a possible code word

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}\in C} \left(\sum_{i=0}^{N-1} \lambda_i x_i\right). \tag{3.19}$$

The maximum likelihood criterion derived here has the cost function of $\sum_{i=0}^{N-1} \lambda_i x_i$. Each possible codeword $\boldsymbol{x} \in C$, is associated with its cost or weight w_c, which can be expressed as the scalar product of $w_c =< \boldsymbol{\lambda}, \boldsymbol{x}_c >$, with c the index of one particular codeword. We decide in favor of the codeword with the minimum resulting weight. Each λ_i value represents the derived LLR result of Sect. 2.2 on bit level, defined in a general form $\lambda_i = ln\frac{P(y_j|x_i=0)}{P(y_j|x_i=1)}$. Depending on the channel statistics this calculation may differ, however the derived ML criterion is still valid if λ_i is represented on

bit level. A general framework to solve the ML criterion for linear block codes is presented in the next section.

3.3.2 ML Decoding by Solving IP Problems

The ML decoding problem can be formulated as an integer program (IP). In literature mainly two different possibilities can be found for the IP modeling [3, 4]. One formulation relies on the parity check equation, the other one relies on the syndrome and the error vector $\boldsymbol{e}$. Both formulations model the ML decoding problem exactly and are thus equivalent.

$$\min \sum_{i=0}^{N-1} \lambda_i x_i \tag{3.20}$$

$$\text{s.t.} \quad \boldsymbol{H}\boldsymbol{x} - 2\boldsymbol{z} = \boldsymbol{0}$$

$$\boldsymbol{x} \in \{0, 1\}^N$$

$$\boldsymbol{z} \in \mathbb{Z}^{N-K}, \; z_i \geq 0 \tag{3.21}$$

The variables $\boldsymbol{x}$ model bits of the codeword, while $\boldsymbol{z}$ is a vector of variables used by the IP formulation to account for the binary modulo 2 arithmetic. The cost function to be minimized is identical to that derived in previous section.

For the syndrome based formulation the cost function changes and is based on the error vector $\boldsymbol{e}$.

$$\min \sum_{i=0}^{N-1} |\lambda_i| e_i \tag{3.22}$$

$$\text{s.t.} \quad \boldsymbol{H}\boldsymbol{e} - 2\boldsymbol{z} = \boldsymbol{s}$$

$$\boldsymbol{e} \in \{0, 1\}^N$$

$$\boldsymbol{z} \in \mathbb{Z}^{N-K}, \; z_i \geq 0 \tag{3.23}$$

The constraints are based on the syndrome vector $\boldsymbol{s}$ while again the $\boldsymbol{z}$ variables ensure the $GF(2)$ operations.

Interestingly both formulations are efficient enough such that a general purpose IP solver can tackle the problem for codes of practical interest. At least for smaller block sizes we can perform Monte-Carlo simulations to obtain the FER performance. The IP formulation has to be solved many times, once for every simulated frame.

Since the formulation are only based on the parity check matrix all types of linear block codes can be simulated with this IP formulation. Furthermore, the IP can be enhanced to comprise as well higher oder modulation types. This was shown in [5].

The general framework of integer programming models is shortly introduced together with some solution strategies. Let $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be given.

Let $\boldsymbol{x} \in \mathbb{R}^n$ denote the vector of variables. Integer programming is the mathematical discipline which deals with the optimization of a linear *objective function* $c^T x$ over the feasible set, i.e. the space of all candidate solutions. A general linear integer programming problem can be written as

$$\begin{aligned} \min \text{ or } \max \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{Z}^n. \end{aligned}$$

Without loss of generality, we may consider minimization problems only. In contrast to linear programming problems, solutions are required to be integral. General IP problems—as well as many special cases—are NP-hard. However, due to extensive studies of theoretical properties, the development of sophisticated methods, as well as increasing computational capabilities, IP proved to be very useful to model and solve many real-world optimization problems (e.g. production planning, scheduling, routing problems,…).

IP problems can be solved in a brute force way by explicitly enumerating all possible values of the variable vector $\boldsymbol{x}$ and choosing the one yielding the minimal objective function value. Though correct, this procedure is guaranteed to be exponential in the number of components of $\boldsymbol{x}$. To avoid excessive computational effort, a theory inducing more efficient algorithms was developed which relies on techniques like implicit enumeration, relaxation and bounds, and the usage of problem-specific structural properties. In the following, we will highlight some basic concepts. An in-depth exposition to this field can be found in [6].

Branch and bound is a wide-spread technique realizing the concept of divide-and-conquer in the context of IP: For each $i \in \{0, \ldots, n-1\}$ and $v \in \mathbb{Z}$, an optimal solution $\boldsymbol{x}^*$ either satisfies $x_i^* \leq v$ or $x_i^* \geq v + 1$.

Using these two constraints two sub problems can be created from the original problem formulation by adding either one or the other constraint to the original set of constraints. This can be seen as branches of a tree, while each branch posses a smaller feasible set. At least one branch results in the optimal solution $\boldsymbol{x}^*$. Iterative application of this branching step yields IP problems of manageable size. For each (sub)problem, primal and dual bounds are obtained by relaxation techniques and heuristics. They allow to prune branches of the search tree, thus reducing the search area (implicit enumeration). Branch and bound techniques are often used within algorithms for communications systems, e.g., the sphere decoding which is presented in Chap. 8.

For any IP problem a linear programming (LP) problem can be derived, called the LP relaxation. This can be done by taking the same objective function and same constraints but with the requirement that the integer variables are replaced by appropriate continuous constraints. Cutting plane algorithms rely on the idea of solving the IP problem as the LP problem

$$\min\{\boldsymbol{c}^T \boldsymbol{x} : \boldsymbol{x} \in \operatorname{conv}(P_I)\}.$$

However, the convex hull of the feasible set (P_I) of the IP problem is in general hard to compute explicitly. Therefore, approaches are developed which iteratively solve the LP relaxation of the IP and compute a cutting plane, i.e., a valid inequality separating the feasible set P_I from the optimal solution of the LP relaxation. These cuts are added to the formulation in all subsequent iteration. Important questions address the convergence of this procedure, as well as the generation of strong valid inequalities. In case of channel code decoding this technique is applied in [4].

A mixture of strategies such as branch-and-cut and the utilization of problem intrinsic structures might lead to enhanced solution strategies. Implementing an efficient IP problem solver is certainly a demanding task. For a special purpose solver, the problem has to be thoroughly understood, a suitable algorithm has to be chosen and efficiently realized. But there also exists a bandwidth of all-purpose solvers, both open source (like GLPK [7]) and commercial (e.g. CPLEX [8]), which may be sufficient to solve various different IP problems such as the ones mentioned for ML decoding. Several ML decoding results are presented within this treatise, all are based on solving the corresponding IP formulations as described in this section. In [9, 10] further IP formulations are presented, that cover different aspects of code analysis. All formulations are general and can therefore be applied to arbitrary linear block codes.

ML Decoding Examples

In the following two simple channel codes are introduced together with their ML decoding procedure. Since the code examples are very small it is possible to explicitly enumerate all solutions. For both examples we assume an input LLR vector $\boldsymbol{\lambda}$ of length $N = 4$. Of course, for decoding the receiver needs of course the knowledge of utilized encoding scheme. One example uses a so called repetition code for encoding, the second example a so called single parity check code.

Example: Repetition Code

A repetition code is a very simple code that simply repeats a single bit of information (K = 1) N times. The generator matrix and the parity check matrix of a repetition code for $N = 4$ is the following:

$$\boldsymbol{G} = [1111] \; \boldsymbol{H} = \begin{bmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \end{bmatrix}$$

In this example one bit is repeated four times which results in a code rate of $R = 1/4$. For this type of a code it is obvious that only two possible codewords exist, since $K = 1$: either the all zero codeword or the all one codeword.

Imagine we receive the LLRs $\boldsymbol{\lambda} = \begin{bmatrix} 0.4 & 1.1 & -0.1 & -0.3 \end{bmatrix}$. The ML decoder determines, which codeword $\hat{\boldsymbol{x}}$ was the one that most likely led to the observation of $\boldsymbol{\lambda}$ solving:

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in C}{\arg\min} \left(\sum_{i=0}^{N-1} \lambda_i x_i \right).$$

As already mentioned the code space C for a repetition code has just two possible codewords. Evaluating the cost function for these two codewords results in:

$$C := \begin{cases} [0\ 0\ 0\ 0] \\ [1\ 1\ 1\ 1] \end{cases} \quad \Rightarrow \quad \sum_{i=0}^{N-1} \lambda_i x_i = \begin{cases} 0 \\ 1.1 \end{cases}$$

When looking at the result of the cost function it can be seen that $\sum_{i=0}^{3} \lambda_i x_i = 0$ is the result for the all zero codeword while for the all one codeword the weight is $\sum_{i=0}^{3} \lambda_i x_i = 1.1$. The ML decoding solution is thus $\hat{\boldsymbol{x}} = [0\ 0\ 0\ 0]$. The minimum distance of this code is $d_{min} = 4$, since all four bits are different. When looking at the input values of $\boldsymbol{\lambda}$ one can see that a decoding with just the sign information $sign(\boldsymbol{\lambda}) = \begin{bmatrix} 1.0 & 1.0 & -1.0 & -1.0 \end{bmatrix}$ would result in cost values which can not be distinguished. Thus we could only detect an error and not correct it. This is one intuitive example which shows that soft information input provides a better error correction capability than hard-input values.

Example: Single Parity Check Code

A second very simple code is the so called single parity check code with $K = 3$ information bits. The corresponding generator and parity check matrices for the resulting block length of N=4 are:

$$\boldsymbol{G} = \begin{bmatrix} 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 1 \\ 0\ 0\ 1\ 1 \end{bmatrix} \boldsymbol{H} = [1111]$$

Again the codeword is generated by the multiplication of the information vector with the generator matrix $\boldsymbol{x} = \boldsymbol{u}G$. The code space has now $2^K = 8$ possible codewords, since the information vector $\boldsymbol{u}$ has now three bits. The resulting eight codewords and thus the code space C is:

$$\boldsymbol{u} = \begin{cases} [0\,0\,0] \\ [0\,0\,1] \\ [0\,1\,0] \\ [0\,1\,1] \\ [1\,0\,0] \\ [1\,0\,1] \\ [1\,1\,0] \\ [1\,1\,1] \end{cases} \Rightarrow [u_0 u_1 u_2] \begin{bmatrix} 1\,0\,0\,1 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \end{bmatrix} = \boldsymbol{x} \qquad C := \begin{cases} [0\,0\,0\,0] \\ [0\,0\,1\,1] \\ [0\,1\,0\,1] \\ [0\,1\,1\,0] \\ [1\,0\,0\,1] \\ [1\,0\,1\,0] \\ [1\,1\,0\,0] \\ [1\,1\,1\,1] \end{cases}$$

In the following we will calculate the ML estimation $\hat{\boldsymbol{x}}$ when receiving $\boldsymbol{\lambda} = \begin{bmatrix} 0.4 & -1.1 & 0.1 & 0.3 \end{bmatrix}$. The cost function for all possible codewords can be evaluated to:

$$C := \begin{cases} [0\,0\,0\,0] \\ [0\,0\,1\,1] \\ [0\,1\,0\,1] \\ [0\,1\,1\,0] \\ [1\,0\,0\,1] \\ [1\,0\,1\,0] \\ [1\,1\,0\,0] \\ [1\,1\,1\,1] \end{cases} \Rightarrow \sum_{i=0}^{N-1} \lambda_i x_i = \begin{cases} 0 \\ 0.4 \\ -0.8 \\ -1.0 \\ 0.7 \\ 0.5 \\ -0.7 \\ -0.3 \end{cases}$$

The smallest value is -1.0 and thus the codeword that most likely equals the originally transmitted codeword is $\hat{\boldsymbol{x}} = [0\,1\,1\,0]$.

Solving the maximum likelihood criterion is the goal of a channel decoder. However, maximum likelihood decoding is an NP-hard problem [11]. The presented examples solve the ML decoding problem by evaluating all possible codes in a brute force manner. In practical examples this is not a feasible solution since the cardinality of the code space is $|C| = 2^K$, with $K > 1000$ for many applications. This is why some channel codes, e.g. convolutional codes, have special properties which enable an efficient realization of an ML decoding algorithm. For example the well known Viterbi algorithm is one algorithm which utilizes a special structure of the code to solve the ML criterion—in this case the so called trellis structure. The trellis structure is an elegant way to represent all possible codewords generated by a convolutional code (CC), see Sect. 3.4. The Viterbi algorithm checks all possible realizations of the codeword $\boldsymbol{x}$ and decides in favor of that codeword with the minimum cost function $\sum_{i=0}^{N-1} \lambda_i x_i$.

3.3.3 Symbol-by-Symbol MAP Decoding

For modern codes (turbo codes, LDPC codes) the maximum likelihood decoding is too complex for practical applications. Thus, heuristics are used which tries to approximate the ML solution. The typical procedure is to divide the code C in smaller component codes

$$C = \{C_0 \cap C_1 \cap \ldots \cap C_j\} \tag{3.24}$$

Typically, the cardinality $|C_k|$ is identically for each sub-code, while a sub-code has a smaller cardinality as the original code $|C_k| \leq |C|$. Each sub-code is a linear code itself which can be defined by a parity check matrix $\boldsymbol{H}_{C_k}$. $\boldsymbol{H}_{C_k}$ is in turn a part of the original parity check matrix. As an example the Hamming code is decomposed in three sub-codes.

$$\boldsymbol{H} = \begin{bmatrix} 1\,0\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1 \end{bmatrix} \quad \begin{aligned} \boldsymbol{H}_{C_0} &= [1\,0\,1\,1\,1\,0\,0] \\ \boldsymbol{H}_{C_1} &= [1\,1\,0\,1\,0\,1\,0] \\ \boldsymbol{H}_{C_2} &= [0\,1\,1\,1\,0\,0\,1] \end{aligned}$$

For decoding, each component code can be solved independently. The result of each component decoder is then passed to the others component code decoders and serves as an additional support information (a priori information). For 3GPP turbo codes the utilized decoding algorithm exchanges information between two component codes $C = \{C_1 \cap C_2\}$. LDPC codes are typically divided in M component codes $C = \{C_0 \cap C_1 \cap \ldots \cap C_{M-1}\}$, which means we decompose the original code, row-by-row, in M single parity check codes.

Each sub-code is decoded by utilizing a so called symbol-by-symbol MAP criterion or in short MAP algorithm. The MAP algorithm provides a posteriori probabilities for each symbol or bit. For each decoded bit x_i, the probability is calculated that this bit was either 0 or 1, given the received sequence $\boldsymbol{\lambda}$.

The MAP probability is derived by an example using a single parity check code. The same received sequence is the starting point, i.e. $\boldsymbol{\lambda} = \begin{bmatrix} 0.4 & -1.1 & 0.1 & 0.3 \end{bmatrix}$.

One possible first step is to evaluate the already explained cost function for all possible codewords.

$$C := \begin{cases} \boldsymbol{x}_0 = [0\,0\,0\,0] \\ \boldsymbol{x}_1 = [0\,0\,1\,1] \\ \boldsymbol{x}_2 = [0\,1\,0\,1] \\ \boldsymbol{x}_3 = [0\,1\,1\,0] \\ \boldsymbol{x}_4 = [1\,0\,0\,1] \\ \boldsymbol{x}_5 = [1\,0\,1\,0] \\ \boldsymbol{x}_6 = [1\,1\,0\,0] \\ \boldsymbol{x}_7 = [1\,1\,1\,1] \end{cases} \quad \Rightarrow \quad \sum_{i=0}^{N-1} \lambda_i x_i = \begin{cases} w_0 = 0 \\ w_1 = 0.4 \\ w_2 = -0.8 \\ w_3 = -1.0 \\ w_4 = 0.7 \\ w_5 = 0.5 \\ w_6 = -0.7 \\ w_7 = -0.3 \end{cases} \tag{3.25}$$

A labeling for each weight $w_c =< \boldsymbol{\lambda}, \boldsymbol{x}_c >$ is now introduced with an associated codeword numbering. $\boldsymbol{x}_c$ is one codeword of the codeword space C with the corresponding cost value w_c. The set of all possible weights is denoted as $\boldsymbol{w}$. The weight which associated to the ML codeword will be denoted as w_{ML}. Now we would like to evaluate the MAP probability for the first bit. Each cost w_c gives us a relative measure of a certain code word $\boldsymbol{x}_c$ with respect to a received $\boldsymbol{\lambda}$.

When we derived the cost function we have subtracted the all zero codeword, see Eq. 3.18. Thus each weight w_c reflects the log likelihood ratio of a certain codeword $\boldsymbol{x}_c$ with respect to the all zero codeword $\boldsymbol{x}_0$.

$$w_c = ln\left(\frac{P(\boldsymbol{x}_c|\boldsymbol{y})}{P(\boldsymbol{x}_0|\boldsymbol{y})}\right) \tag{3.26}$$

with $\boldsymbol{x}_c$ one codeword of the codeword space C. The MAP probability for the first bit position $P(x_0 = 0|\boldsymbol{y})$ gives the probability that the first bit position was sent as a zero, i.e., that either $\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2$ or $\boldsymbol{x}_3$ have been sent. These four codewords have all a zero at the first position. When looking on Eq. 3.26 we can evaluate this question by

$$P(x_0 = 0|\boldsymbol{y}) = \frac{e^{w_0} + e^{w_1} + e^{w_2} + e^{w_3}}{\text{normalizer}}. \tag{3.27}$$

The normalizer has to ensure that the probability over all codewords is 1. Again the normalizer can be omitted when building the likelihood ratio, or here directly the log-likelihood ratio.

$$\Lambda(x_0|\boldsymbol{y}) = ln\left(\frac{P(x_0 = 0|\boldsymbol{y})}{P(x_0 = 1|\boldsymbol{y})}\right) = ln\frac{e^{w_0} + e^{w_1} + e^{w_2} + e^{w_3}}{e^{w_4} + e^{w_5} + e^{w_6} + e^{w_7}} \tag{3.28}$$

This is the symbol-by-symbol MAP result in terms of log-likelihood ratios, since $\Lambda(x_0|\boldsymbol{y})$ is conditioned on an entire receive vector it denotes with a capital lambda. For the second bit position x_1 we have to evaluate

$$\Lambda(x_1|\boldsymbol{y}) = ln\left(\frac{P(x_1 = 0|\boldsymbol{y})}{P(x_1 = 1|\boldsymbol{y})}\right) = ln\frac{e^{w_0} + e^{w_1} + e^{w_4} + e^{w_5}}{e^{w_2} + e^{w_3} + e^{w_6} + e^{w_7}}. \tag{3.29}$$

A general expression for each bit position results in

$$\Lambda(x_i|\boldsymbol{y}) = ln\frac{\sum\limits_{\boldsymbol{x}|x_i=0} e^{w_c}}{\sum\limits_{\boldsymbol{x}|x_i=1} e^{w_c}} \tag{3.30}$$

$\boldsymbol{x}|x_i = 1$ is a codeword $\boldsymbol{x} \in C$ with the is bit set to 1. Equation 3.30 is one possible description to calculate the symbol-by-symbol MAP probability for any type of linear block code. The formulation is based on the weights of LLR values and can be used as starting point for implementation. The basic procedure to solve this equation is to

exploit the code properties to enable an intelligent search with respect to the current constraint codeword $\boldsymbol{x}|x_i$.

3.3.4 Max-Log-MAP Approximation

Equation 3.30 delivers the correct MAP result in the log-likelihood domain and is typically denoted as Log-MAP solution. The logarithm over the sum of exponential functions can expressed by the Jacobian logarithm either in its positive form with

$$\ln\left(e^{\delta_1} + e^{\delta_2}\right) = \max^*(\delta_1, \delta_2) = \max(\delta_1, \delta_2) + ln(1 + e^{-|\delta_1 - \delta_2|}), \tag{3.31}$$

or in its negative form with

$$-\ln\left(e^{-\delta_1} + e^{-\delta_2}\right) = \min^*(\delta_1, \delta_2) = \min(\delta_1, \delta_2) - ln(1 + e^{-|\delta_1 - \delta_2|}). \tag{3.32}$$

By utilizing the Jacobian logarithm and ignoring the correction term $ln(1+e^{-|\delta_1-\delta_2|})$ yields the so called Max-Log-MAP approximation which will be:

$$\begin{aligned} \Lambda(x_i|\boldsymbol{y}) &= ln\left(\frac{P(x_i = 0|\boldsymbol{y})}{P(x_i = 1|\boldsymbol{y})}\right) \\ &\approx + \max_{\boldsymbol{x}|x_i=0}\{\boldsymbol{w}\} - \max_{\boldsymbol{x}|x_i=1}\{\boldsymbol{w}\} \end{aligned} \tag{3.33}$$

Or we can express this function as well utilizing the minimum search by just multiplying it with -1.0; The resulting Max-Log-MAP expression results in:

$$\begin{aligned} \Lambda(x_i|\boldsymbol{y}) &= ln\left(\frac{P(x_i = 0|\boldsymbol{y})}{P(x_i = 1|\boldsymbol{y})}\right) \\ &\approx - \min_{\boldsymbol{x}|x_i=0}\left\{\sum_{i=0}^{N-1}\lambda_i x_i\right\} + \min_{\boldsymbol{x}|x_i=1}\left\{\sum_{i=0}^{N-1}\lambda_i x_i\right\} \\ &\approx - \min_{\boldsymbol{x}|x_i=0}\{\boldsymbol{w}\} + \min_{\boldsymbol{x}|x_i=1}\{\boldsymbol{w}\} \end{aligned} \tag{3.34}$$

Equation 3.34 searches through the weights $\boldsymbol{w}$, always with respect to one position i. Note that the weight of the ML solution w_{ML} will always part of the solution.

The sub-optimal Max-Log-MAP calculation becomes clearer when evaluating again the our single parity check example with the received sequence $\boldsymbol{\lambda} = \begin{bmatrix} 0.4 & -1.1 & 0.1 & 0.3 \end{bmatrix}$. The corresponding weights and codewords are shown in Eq. 3.25, the approximated Max-Log-MAP expression evaluates to:

$$\begin{aligned}
\Lambda(x_0|\boldsymbol{y}) &= -min\{0, 0.4, -0.8, -1.0\} + min\{0.7, 0.5, -0.7, -0.3\} = \quad 0.3 \\
\Lambda(x_1|\boldsymbol{y}) &= -min\{0, 0.4, 0.7, 0.5\} + min\{-0.8, -1.0, -0.7, -0.3\} = -1.0 \\
\Lambda(x_2|\boldsymbol{y}) &= -min\{0, -0.8, 0.7, -0.7\} + min\{0.4, -1.0, 0.5, -0.3\} = -0.2 \\
\Lambda(x_3|\boldsymbol{y}) &= -min\{0, -1.0, 0.5, -0.7\} + min\{0.4, -0.8, 0.7, -0.3\} = \quad 0.2
\end{aligned}$$

In each result of $\Lambda(x_i|\boldsymbol{y})$ the weight of the ML result $w_{ML} = -1.0$ is used. The sign bit of each LLR value calculated by the Max-Log-MAP approximation correspond to the maximum likelihood estimation $\hat{\boldsymbol{x}}$.

In hardware realizations we can either use the Max-Log-MAP approximation or the Log-MAP realization which mainly depends on the type of application.

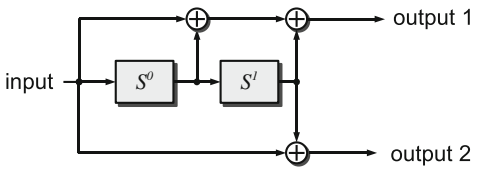
- A Max-Log-MAP implementation is chosen when the communications performance result of a Max-Log-MAP approximation is close to that of a Log-MAP realization. The difference with respect to communications performance is shown for turbo codes in Sect. 6.2.2.
- The computational complexity of the approximation term can be quite significant. However, sometime an approximation towards the Log-MAP solution becomes mandatory due to communications performance reasons. Then in hardware an approximation of the correction term is realized, one possible approximation is shown in Sect. 5.2.
- One major advantage of a MAX-Log-MAP realization is its independence of the signal-to-noise ratio, i.e. the algorithm is robust with respect to linear scaling of the input values provided by the demodulator. One example with respect to fixed-point realization and analysis of the robustness of an algorithm is shown in Sect. 6.2.3.

3.4 Convolutional Codes

Convolutional codes (CC) were already introduced in 1954 [12] and are still parts of many communication systems. This section just gives a pragmatic introduction to this type of codes. The theory for designing a convolutional codes and its mathematical description mandatory for in-depth analysis is omitted. An excellent in depth analysis is found in [1].

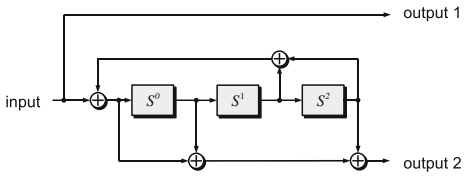
The encoder of a convolutional code is either a recursive or a non recursive filter, operating on bit level. It is composed of M memory elements, as shown for $M = 2$ in Fig. 3.2 and for $M = 3$ in Fig. 3.3 respectively.

Figure 3.2 shows a non-recursive encoder for a non-systematic convolutional (NSC) code. One input bit produces always two output bits. Thus this is a $R = \frac{1}{2}$ code. Depending on the values stored at any given moment in the storage elements S^0 and S^1 and on the input different output bits are produced. Since two storage devices are present $M = 2$, 4 different internal states can be reached (2^M) The so called transition table is shown at the right which shows the current state, input, next state, output1 and output2 values. Figure 3.3 shows a recursive systematic convolutional (RSC) code, again with transition table. Recursive means that there exist a feedback



current state	input	next state	out1-out2
00	0	00	00
00	1	10	11
01	0	00	11
01	1	10	00
10	0	01	10
10	1	11	01
11	0	01	01
11	1	11	10

Fig. 3.2 4-state NSC code with state transition table



current state	input	next state	out1-out2
000	0	000	00
000	1	100	11
001	0	100	00
001	1	000	11
010	0	101	01
010	1	001	10
011	0	001	01
011	1	101	10
100	0	010	01
100	1	110	10
101	0	110	01
101	1	010	10
110	0	111	00
110	1	011	11
111	0	011	00
111	1	111	11

Fig. 3.3 8-state RSC code with state transition table. This code is used in 3GPP turbo encoding

from the state information to the input bit. The plus boxes indicates an XOR of the participating bits.

The transition table describes a Mealy automaton, where the output depends on the state and the input information. For the 4-state NSC code the automaton is shown in Fig. 3.4. The finite state diagram does not contain any information in time, but unrolled over time steps results in the trellis diagram. For each time step k one new trellis step exists, which is shown for three time steps in Fig. 3.4. The trellis is one graphical representation for all possible output sequences.

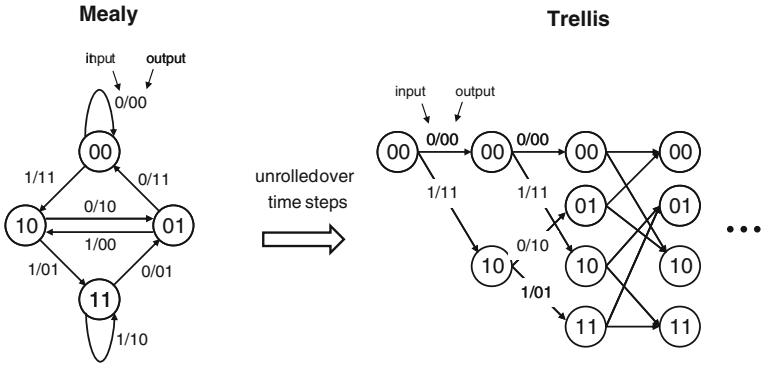


Fig. 3.4 State diagram unrolled over the time results in the trellis diagram

As described by the Mealy automaton we have in this example always two possibilities to reach a certain state. In the case of the encoding procedure of Fig. 3.4 this reflects two possible input/output sequences resulting in the same state.

In Fig. 3.5 the trellis is shown for two possible paths which converge again, e.g. there exist two paths which start from the same state and end up at the same state. Since a path is associated to an input bit/output bit combination one can see that the input sequence $[u_0\ u_1\ u_2] = [0\ 0\ 0]$ and $[u_0\ u_1\ u_2] = [1\ 0\ 0]$ have paths which merges again. The corresponding output sequence and thus part of the codeword are $[x_0\ x_1\ x_2\ x_3\ x_4\ x_5] = [0\ 0\ 0\ 0\ 0\ 0]$ and $[x_0\ x_1\ x_2\ x_3\ x_4\ x_5] = [1\ 1\ 1\ 0\ 1\ 1]$, respectively. This 6 bits are just a part of the codeword. Thus we have at least a minimum distance of the two codewords of $d_{min} = 5$.

As well convolutional codes are utilized in packet based transmission systems. At the end of the encoding of one block, typically the trellis is terminated in the zero state. This is enforced by adding tail bits to the sequence. The tail bits are chosen depending on the last encoding stage. This is shown for the 4-state NSC code of our example in Fig. 3.6. For a 2^M-state NSC code a sequence of M zeros can be used as input to enforce the termination in the zero state.

A formal method for describing a convolutional code is to give its generator matrix G in the D-transform notation, where D represents a delay operator. With respect to the encoding process of Fig. 3.2 we can say that the current input bit at time step $D^0 = 1$ is influenced by the bit of the previous time step D^1 and the bit at the input

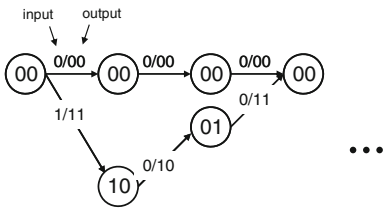


Fig. 3.5 Merging paths in a trellis

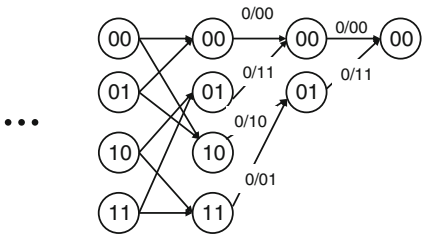


Fig. 3.6 Tail bits to enforce the final zero state

two time steps before D^2. The linear map of the input bits onto the coded bits can be written as a multiplication of the input sequence and a generator:

$$\boldsymbol{x}(D) = \boldsymbol{u}(D)G(D) \tag{3.35}$$

For the 4-state NSC encoder (Fig. 3.2) and the presented 8-state RSC encoder these generators are:

$$G(D) = [1 + D + D^2 \quad 1 + D^2] \qquad G(D) = \left[1 \quad \frac{1 + D^2 + D^3}{1 + D + D^3}\right] \tag{3.36}$$

In communication standards these generators are given typically in octal form which directly shows the structure for the encoding process. For the 4-state the so called forward polynomial for output one is $G_0 = 111_2 = 7_8$, and for output two is $G_1 = 101_2 = 5_8$. The binary form describes directly the taps of the NSC filter. Thus, the octal representation groups 3 taps positions per digit. For the recursive structure a standard typically defines a forward polynomial, here for the 8-state $G_0 = 13_8$ and the feedback polynomial as $G_{FB} = 15_8$. For different code rates more forward polynomials may exist, which are labeled by G_i.

Since the encoding scheme and as well the decoding scheme is quite simple many communication standards utilize convolution codes. Table 3.4 shows a selection of different convolutional codes utilized in various standards. The list shows the number of states, the code rate, and the defined polynomials. In a communication standards like defined by 3GPP there exist various different codes for e.g. different control channels or data channels. It quite surprising how many different encoder structures can be found when reading through the documentation of the standards.

3.4.1 ML Decoding of Convolutional Codes

For decoding we ideally would like to solve the ML criterion which is defined as:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in C} \left(\sum_{i=0}^{N-1} \lambda_i x_i \right)$$

Table 3.4 Selection of standards featuring convolutional codes, with the code type, number of states, and polynomials

Standard	Codes	States	Rate	Polynomials
	NSC	16	1/2	$G_0 = 23_8, G_1 = 33_8$
GSM/EDGE	NSC	16	1/3	$G_0 = 33_8, G_1 = 25_8, G_2 = 37_8$
	RSC	16	1/2	$G_0 = 33_8, G_{FB} = 23_8$
	RSC	16	1/3	$G_0 = 33_8, G_1 = 25_8, G_{FB} = 37_8$
GSM/EDGE	NSC	64	1/2	$G_0 = 123_8, G_1 = 171_8$
	NSC	64	1/3	$G_0 = 123_8, G_1 = 145_8, G_3 = 171_8$ or $G_3 = 175_8$
	NSC	64	1/4	$G_0 = 123_8, G_1 = 145_8, G_2 = 175_8, G_3 = 171_8$
GSM/EDGE	RSC	64	1/4	$G_0 = 123_8, G_1 = 145_8, G_2 = 175_8, G_{FB} = 171_8$
	RSC	64	1/4	$G_0 = 123_8, G_1 = 145_8, G_2 = 171_8, G_{FB} = 175_8$
UMTS	NSC	256	1/2	$G_0 = 561_8, G_1 = 753_8$
	NSC	256	1/3	$G_0 = 557, G_1 = 663_8, G_2 = 711_8$
DVB-H	CC	64	1/2	$G_0 = 171_8, G_1 = 133_8$
IEEE802.11a/g/n	CC	64	1/2	$G_0 = 133_8, G_1 = 171_8$
IEEE802.16e	CC	64	1/2	$G_0 = 133_8, G_1 = 171_8$

As seen in Sect. 3.3.2 a brute force check of all possible code words is quite cumber sum or even impossible for larger block sizes. The trellis however as introduced in the previous chapter is a graph structure which was obtained direct from the encoding process and represents all possible bit patterns. Solving the ML solution by using a trellis is equivalent to solving a shortest path problem and is denoted as Viterbi algorithm in the field of communications [13].

This is now demonstrated for a code Rate of $R = 1/2$. The sum of the cost function can be decomposed in partial sums:

$$\left(\sum_{i=0}^{N-1} \lambda_i x_i\right) = \left(\sum_{i=0}^{2k-1} \lambda_i x_i\right) + \underbrace{\left(\sum_{i=2k}^{2k+1} \lambda_i x_i\right)}_{partial \sum = branch\ metric} + \left(\sum_{i=2k+2}^{N-1} \lambda_i x_i\right) \tag{3.37}$$

The middle part, which reflects the partial sum of one encoding step, is of special interest. This partial sum is labeled with $\gamma_k^{x_i x_{i+1}}$ and are denoted as branch metrics. Four different branch metrics are possible, depending on the bit combinations of x_i and x_{i+1}.

$$\begin{aligned} \gamma_k^{00} &= 0 &&\rightarrow [x_i\ x_{i+1}] = [0\ 0] \\ \gamma_k^{01} &= \lambda_{i+1} &&\rightarrow [x_i\ x_{i+1}] = [0\ 1] \\ \gamma_k^{10} &= \lambda_i &&\rightarrow [x_i\ x_{i+1}] = [1\ 0] \\ \gamma_k^{11} &= \lambda_i + \lambda_{i+1} &&\rightarrow [x_i\ x_{i+1}] = [1\ 1] \end{aligned} \tag{3.38}$$

As already mentioned the trellis defines all possible bit combinations. A valid path from the beginning to end of the trellis defines a valid codeword. For each trellis step

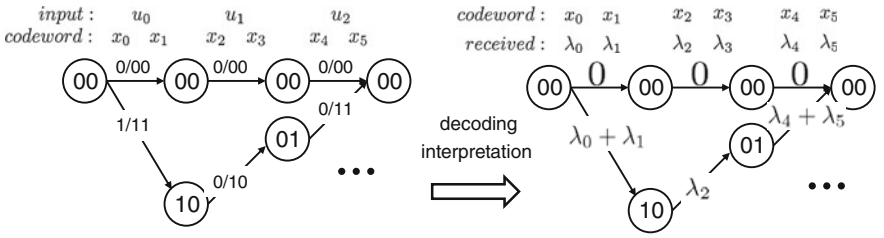


Fig. 3.7 Merging paths in a trellis

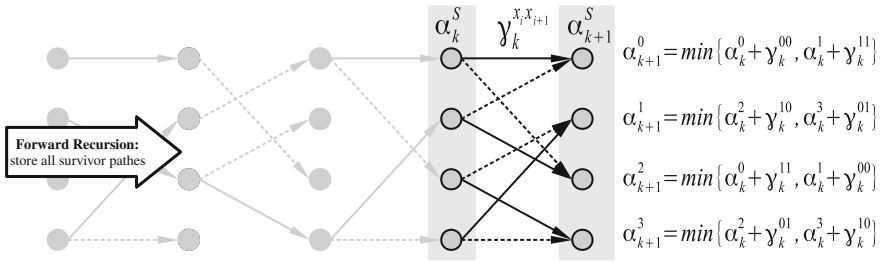


Fig. 3.8 Viterbi processing with the resulting surviver paths

k we can associate one intermediate sum value of Eq. 3.38 to one state transition, while i defines the position of the bit x_i within the codeword. The relation between trellis step k and codeword position i is $2 * k = i$, which results due to the code rate of $R = \frac{1}{2}$ within the example. One branch metric associated to an edge is denoted as:

$$\gamma_k^{x_i x_{i+1}}(S_k^m, S_{k+1}^{m'}) \tag{3.39}$$

which determines the mapping of a branch metric to a state transition or edge in the trellis, with S_k^m and $S_{k+1}^{m'}$ the connected states at trellis step k to $k+1$ with $m, m' \in \{0 \ldots 2^M - 1\}$. Fig. 3.7 shows 3 trellis steps which corresponds to 6 bits of the codeword. On the right side of Fig. 3.7 the paths are labeled with the corresponding $\sum \lambda_i x_i$ values. The labeling of the edges are exactly one branch metric of Eq. 3.38. The partial sum $(\lambda_0 + \lambda_1) + (\lambda_2) + (\lambda_4 + \lambda_5)$ comprises 3 encoding steps and is part of a valid cost function. However, the all zero path with the partial sum weight $0 + 0 + 0$ could be one part as well. Thus it makes sense only to follow the survivor of a minimum search

$$\alpha_{k=3}^0 = \min\{0 + 0 + 0\ ,\ (\lambda_0 + \lambda_1) + (\lambda_2) + (\lambda_4 + \lambda_5)\} \tag{3.40}$$

With $\alpha_{k=3}^0$ the so called state metric or current survivor sum at trellis step $k = 3$ and state S^0. The calculation has to be done for each state and each trellis step. We recursively calculate new state metrics α_{k+1}^S, with S the state number which is in this example for possible states with $S \in 0, 1, 2, 3$.

Figure 3.8 shows the basic sketch of the Viterbi processing at processing step k. Processing step and trellis step are identically in this example. The Viterbi algorithm which calculates the ML codeword can be divided in 3 major steps

- branch metric allocation: according to the state transition tables, allocate one of the corresponding branch metrics of Eq. 3.38 to an each edge in the trellis, according to Eq. 3.39.
- forward recursion: calculate at each time step and thus trellis step k the corresponding survivor path metric α_{k+1}^{m} for each state $m \in \{0 \ldots 2^M - 1\}$. We have to store these results for each state of the current processing step k. The α_{k+1}^{m} state metric hold an intermediate sum of the overall cost function of Eq. 3.37 to reach this particular state. It is the smallest sum which can be evaluated to reach this state, starting form the very first state α_0^0. In addition to the state metrics we have to store an indication about the surviver edge, i.e. where did I come from. The resulting surviver paths are indicated in Fig. 3.8 to the left of the current processing step.
- trace back: When reaching the final state at trellis step $k = N/2$ we have to extract the path which connects α_0^0 and $\alpha_{N/2}^0$. When we reached the last terminated state we obtained the final cost value of the entire code word $\alpha_{N/2}^0 = \sum_{i=0}^{N-1} \lambda_i x_i$. Obtaining the final cost value is, however, secondary to the objective of finding the ML path $\hat{\boldsymbol{x}}$. The way to obtain this path is quite elegant. For each time step we have stored the already indicated survivor at each trellis step. When reaching the final state we now use a back tracing of the path information to obtain $\hat{\boldsymbol{x}}$ which is thus obtained in a reversed order.

3.4.2 Max-Log-MAP Decoding of Convolutional Codes

The symbol-by-symbol maximum a posteriori algorithm, short MAP algorithm, provides a posteriori probabilities for each symbol or bit. The original algorithm in probability domain for trellis codes was already published in 1974 [14] and named after their inventors BCJR algorithm. However, the algorithm was not of practical importance since it is more complex than the Viterbi algorithm and has no advantage when decoding only convolutional codes. The Viterbi algorithm find the optimum sequence while the MAP algorithm provides additional reliability information for each position. First the invention of iterative decodable codes turned the focus again on the MAP algorithm. In iterative decoding the 'soft' information plays an important role.

In hardware the processing is always done in the so already mentioned log-likelihood domain. For each decoded bit x_i, the Log-Likelihood Ratio (LLR) is calculated that this bit was 0 or 1, given the received symbol sequence.

$$\Lambda(x_i|\boldsymbol{y}) = ln\left(\frac{P(x_i = 0|\boldsymbol{y})}{P(x_i = 1|\boldsymbol{y})}\right) \tag{3.41}$$

Solving the Log-MAP or Max-Log-MAP criterion in a brute force manner was already shown in Sect. 3.3.3. Here we show the procedure with respect to a trellis representation. The MAP processing on a trellis is a so called forward-backward algorithm and is shown for trellis step k in Fig. 3.9. The processing can be decomposed in 4 steps.

- branch metric allocation: according to the state transition tables, allocate one of the corresponding $\gamma_k^{x_i x_{i+1}}$ values of Eq. 3.38 to each edge in the trellis. This part is identical to the Viterbi algorithm
- forward recursion: calculate at each time step and thus trellis step k the corresponding path metric α_{k+1}^S for every state S. We have to store these results for each state S of all processing step up to trellis step k. Only the current state metrics at a given trellis step are utilized for the next time step. For the Max-Log-MAP algorithm exactly the same α or forward recursion is utilized which can be expressed in a more general form as:

$$\alpha_{k+1}^{m'} = \min_{\forall(m->m')} \left(\alpha_k^m + \gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'}) \right) \tag{3.42}$$

  We have find the minimum over all possibilities which connect states at the previous time step to $S_{k+1}^{m'}$.
- backward recursion: the backward recursion has exactly the same recursive functionality, however starting with the calculation from the last state in the trellis.

$$\beta_k^m = \min_{\forall(m'->m)} \left(\beta_{k+1}^{m'} + \gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'}) \right) \tag{3.43}$$

  Again we store all obtained results for each trellis step.
- soft-output calculation: we would like to calculate the symbol-by-symbol Max-Log MAP result. For this we need the results of the forward recursion, backward recursion and the current branch metric, which is:

$$\begin{aligned} \Lambda(x_i, \boldsymbol{y}) = & \min_{\forall\gamma|x_i=0} \left(\gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'}) + \alpha_k^m + \beta_{k+1}^{m'} \right) \\ & - \min_{\forall\gamma|x_i=1} \left(\gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'}) + \alpha_k^m + \beta_{k+1}^{m'} \right). \end{aligned} \tag{3.44}$$

  We are searching the minimum over all sum of weights which has at trellis step k under the condition that the corresponding codeword bit is either zero or one.

The individual processing steps of the Max-Log-MAP algorithm are explained now by a small example. Note, that for practical hardware implementation for turbo decoding only this algorithm is implemented, more details about this in the turbo decoder chapter. The utilized channel code in the following example is a simple 2-state RSC with $G_0 = 1$ and $G_{FB} = 3_8$, its mealy automaton ins displayed next to its trellis representation, see Fig. 3.10.

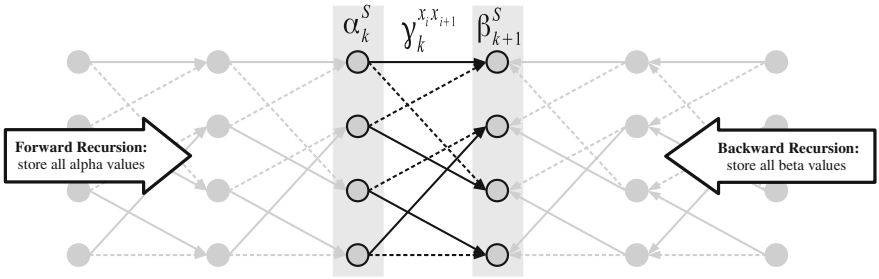


Fig. 3.9 MAP processing with the forward and backward recursion

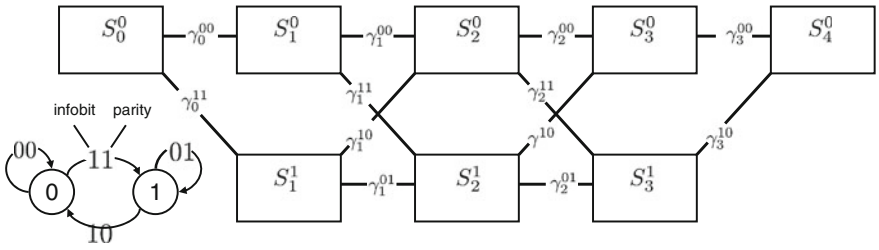


Fig. 3.10 *Step one: set up the corresponding trellis and the labeling*. The two state trellis with 4 trellis steps is shown. For each trellis step i and each edge a branch metric γ exists. Depending on the possible systematic, parity bit combination $x_i^s x_i^p$ a different branch metric has to be allocated

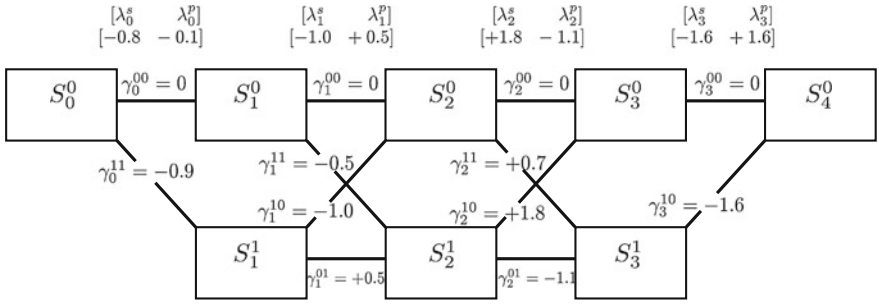


Fig. 3.11 *Step two: allocate the corresponding branch values*. The corresponding λ-values for each trellis step are shown at the top. The branch metrics with the corresponding values are now updated (the trellis step label k is omitted here). Each branch metric is calculated according to Eq. 3.38

The information word used for demonstration has just 4 bits, with $\boldsymbol{u} = [0101]$. The resulting codeword after encoding is the concatenation of the systematic part $\boldsymbol{x^s}$ and the parity part $\boldsymbol{x^p}$. The codeword is modulated via a binary phase shift keying (BPSK), which mapping $0 \rightarrow +1$ and $1 \rightarrow -1$ respectively. After demodulation we receive the LLRs. The values of the resulting codeword, the sent symbols, and the noise corrupted decoder input are the following:

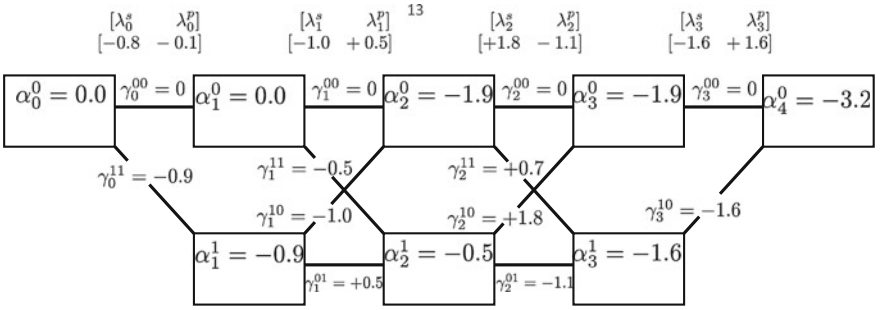


Fig. 3.12 *Step three: calculate the forward state metrics* α: In this example only two possible transitions exist between the previous states and the state we would like to calculate. Since only two states exist in this example, only two equations have to be solved for α_{k+1}^0 and α_{k+1}^1 respectively

$$\alpha_{k+1}^0 = \min\{\alpha_k^0 + \gamma_k^{00}, \alpha_k^1 + \gamma_k^{10}\}$$
$$\alpha_{k+1}^1 = \min\{\alpha_i^0 + \gamma_k^{11}, \alpha_k^1 + \gamma_k^{01}\}$$

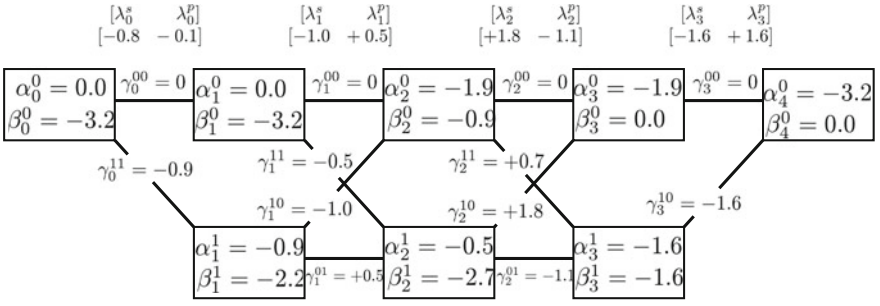


Fig. 3.13 *Step four: calculate the backward state metrics* β: The calculation is identical to that of the forward recursion, again only two equations have to be solved at each trellis step:

$$\beta_k^0 = \min\{\beta_{k+1}^0 + \gamma_k^{00}, \beta_{k+1}^1 + \gamma_k^{11}\}$$
$$\beta_k^1 = \min\{\beta_{k+1}^0 + \gamma_k^{10}, \beta_{k+1}^1 + \gamma_k^{01}\}$$

codeword:
$\boldsymbol{x} = [\boldsymbol{x^s}\ \boldsymbol{x^p}]$ = [[0 1 0 1] [0 1 1 0]]
sent symbol:
$\boldsymbol{s}$ = [+1 −1 +1 −1 +1 −1 −1 +1]
received:
$\boldsymbol{\lambda} = [\boldsymbol{\lambda^s}\ \boldsymbol{\lambda^p}]$ = [[−0.8 −1.0 +1.8 −1.6] [−0.1 +0.5 −1.1 +1.6]]

For the decoding algorithm we several steps have to be done, see Figs. 3.10, 3.11, 3.12, 3.13 and 3.14.

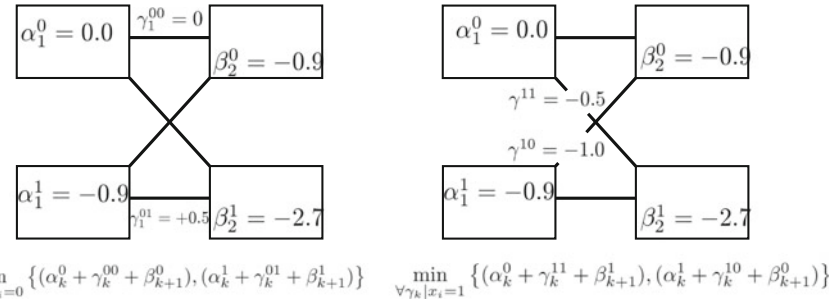


Fig. 3.14 *Step five: calculate the output Max-Log-MAP approximation:* We calculate now the output result by evaluating: $\Lambda(x_i|\boldsymbol{y}) \approx -\min_{\gamma_k|x_i=0}\{\ldots\} + \min_{\gamma_k|x_i=1}\{\ldots\}$. The figure shows the participating α, β, and γ values to calculate trellis step $k = 1$. The left side for all possible $\gamma_k|x_i = 0$ transitions and the right side of the figure shows the equation for the $\gamma_k|x_i = 0$ transition. $\alpha_0^1 = inf$ and $\beta_4^1 = inf$ are both initialized with an infinite value, since these states can not be reached. Zero values are not explicitly stated within the sum terms. For each position we have to evaluate these equations which are shown in the following:

$$
\begin{aligned}
\Lambda(x_0|\boldsymbol{y}) &\approx -\min_{\gamma_0|x_0=0}\{(-3.2), \inf\} \\
&\quad + \min_{\gamma_0|x_0=1}\{(-0.9 - 2.2), \inf\} && = 0.1 \\
\Lambda(x_1|\boldsymbol{y}) &\approx -\min_{\gamma_1|x_1=0}\{(-0.9), (-0.9 + 0.5 - 2.7)\} \\
&\quad + \min_{\gamma_1|x_1=1}\{(-0.5 - 2.7), (-0.9 - 1.0 - 0.9)\} && = -0.1 \\
\Lambda(x_2|\boldsymbol{y}) &\approx -\min_{\gamma_2|x_2=0}\{(-1.9), (-0.5 - 1.1 - 1.6)\} \\
&\quad + \min_{\gamma_2|x_2=1}\{(-1.9 + 0.7 - 1.6), (-0.5 + 1.8)\} && = 0.4 \\
\Lambda(x_3|\boldsymbol{y}) &\approx -\min_{\gamma_3|x_3=0}\{(-1.9), \inf\} \\
&\quad + \min_{\gamma_3|x_3=1}\{\inf, (-1.6 - 1.6)\} && = -1.3
\end{aligned}
\tag{3.45}
$$

3.5 Soft-Input Soft-Output (SISO) Decoder

The previous example is summarized with the following 4 lines, the sent codeword, sent symbol, received LLR, and output result respectively.

$$
\begin{aligned}
\boldsymbol{x} = [\boldsymbol{x}^s\ \boldsymbol{x}^p] &= [[\ \ 0\ \ 1\ \ 0\ \ 1]\ \ [0\ \ 1\ \ 1\ \ 0\ ]] \\
\boldsymbol{s} &= [\ +1\ -1\ +1\ -1\ +1\ -1\ -1\ +1\ ] \\
\boldsymbol{\lambda} = [\boldsymbol{\lambda}^s\ \boldsymbol{\lambda}^p] &= [[-0.8\ -1.0\ +1.8\ -1.6]\ [-0.1\ +0.5\ -1.1\ +1.6\ ]] \\
\boldsymbol{\Lambda}(x_i|\boldsymbol{y}) &= [[\ 0.1\ -0.1\ \ 0.4\ -1.3]\ \ [x\ \ \ x\ \ \ x\ \ \ x\ ]]
\end{aligned}
$$

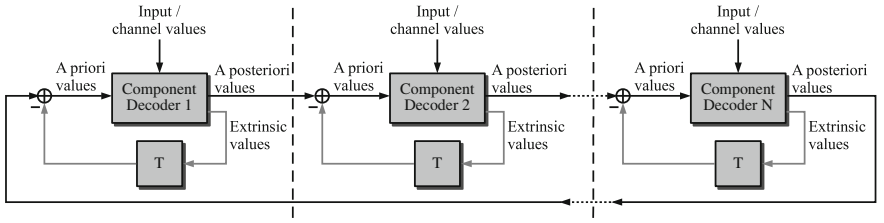


Fig. 3.15 Concatonation of component decoders for soft-in soft-out decoding

As mentioned the result of the symbol-by-symbol MAP algorithm is denoted as well as soft-output information. Typically the decoder utilizes all soft-input information to calculate this soft-output information. The gain we obtained by the soft-input soft-output algorithm is denoted as extrinsic information. It is calculated by taking the difference between the obtained information of the MAP algorithm and the input information.

$$\text{Extrinsic LLR } \boldsymbol{L}^e = \boldsymbol{\Lambda} - \boldsymbol{\lambda} = [\ 0.9\ 0.9\ -1.4\ 0.3\ ]$$

The **extrinsic information** of a symbol or bit is the additional information a (MAP) decoder calculates. The decoder calculates the a posteriori information taking all input information into account. The extrinsic information is thus the a posteriori information excluding the entire input information.

Typically, for decoding we have two types of soft-input information. The channel information with here denoted as $\boldsymbol{\lambda}$ and an additional information which is called **a priori information** ($\boldsymbol{L}^a$) which is in this example zero. The a priori information of symbol or bit is an additional information known before decoding. This information may come from a source independent of the received sequence.

The efficient usage of a priori and extrinsic information turned in to focus when Berrou and Glavieux presented the turbo codes (TC) [15]. These channel codes are decoded in an iterative manner and are already applied in many communication standards [16, 17]. Note that already low-density parity-check (LDPC)codes, introduce in 1963 utilized the iterative decoding.

The success of the iterative decoding process is the efficient usage of extrinsic and a priori information. The major principle of **all** iterative channel code decoders is the exchange of reliability information, in terms of LLRs, between 2 or more component decoders. The basic code structure of TCs is the random concatenation of 2 component codes. In the case of LDPC codes many simple component codes are concatenated. Both code types and the implementation of the decoding algorithm are explained in detail in the next chapters.

The concatenation of different component codes for decoding is shown in Fig. 3.15. For each component code a corresponding component decoder exist. A component decoder calculates a local MAP information of the bits to be decoded. These information is re-sorted according to the concatenation of the component

codes and then passed to the connected component codes. The re-sorting process is denoted as interleaving in the following, see Sect. 4.3. One iteration is finished if each component code has updated the information ones.

The outstanding communications performance of concatenated codes can only be obtained when each component decoder is utilizing soft information at the input and calculates new soft information at the output which is denoted as SISO decoder and does not really tell something about the utilized decoding algorithm like Log-MAP, Max-Log-MAP, or other algorithms.

The output a posteriori probabilities ($\boldsymbol{\Lambda}$) can be decomposed in three parts: the channel input information, the additional gain (extrinsic information), and the a priori information:

$$\boldsymbol{\Lambda} = \boldsymbol{\lambda} + \boldsymbol{L}^a + \boldsymbol{L}^e \tag{3.46}$$

Only the additional gain ($\boldsymbol{L}^e$) is passed to the other component decoders with respect to the connectivity structure. Which means, we subtract the input LLRs as well as the input a priori information:

$$\boldsymbol{L}^e = \boldsymbol{\Lambda} - \boldsymbol{\lambda} - \boldsymbol{L}^a \tag{3.47}$$

This additional information serves now as a priori information for an other component code decoder and is besides the channel information one part of the soft-input information.

Without subtracting the old input information a SISO decoder (Eq. 3.47) would just confirm its decision of prior iterations.

References

1. Lin, S., Costello, D.J. Jr.: Error Control Coding, 2nd edn. Prentice Hall PTR, Upper Saddle River (2004)
2. Bossert, M.: Kanalcodierung, 2nd edn. B.G. Teubner, Stuttgart (1998)
3. Breitbach, M., Bossert, M., Lucas, R., Kempter, C.: Soft-decision decoding of linear block codes as optimization problem. Eur. Trans. Telecommun. **9**(3), 289–293. doi:10.1002/ett.4460090308. http://dx.doi.org/10.1002/ett.4460090308 (1998)
4. Tanatmis, A., Ruzika, S., Hamacher, H.W., Punekar, M., Kienle, F., Wehn, N.: A separation algorithm for improved LP-decoding of linear block codes. IEEE Trans. Inf. Theor. **56**(7), 3277–3289 (2010). doi:10.1109/TIT.2010.2048489
5. Scholl, S., Kienle, F., Helmling, M., Ruzika, S.: ML vs. MP decoding of binary and non-binary LDPC codes. In: Proceedings of the 7th International Symposium on Turbo Codes and Iterative Information Processing (2012)
6. Wolsey, L.A., Nemhauser, G.L.: Integer and Combinatorial Optimization. Wiley-Interscience, New York (1999)
7. Free Software Foundation Inc.: GLPK—GNU Linear Programming Kit. http://www.gnu.org/software/glpk/. Accessed Apr 2012
8. IBM: IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/. Accessed Apr 2012

9. Punekar, M., Kienle, F., Wehn, N., Tanatmis, A., Ruzika, S., Hamacher, H.W.: Calculating the minimum distance of linear block codes via integer programming. In: Proceedings of the 6th International Turbo Codes and Iterative Information Processing (ISTC) Symposium, pp. 329–333 (2010). doi:10.1109/ISTC.2010.5613894
10. Scholl, S., Kienle, F., Helmling, M., Ruzika, S.: Integer programming as a tool for code analysis. In: 9th International ITG Conference on Systems, Communications and Coding (2013) (Accepted)
11. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theor. **24**(3), 384–386 (1978). doi:10.1109/TIT.1978.1055873
12. Elias, P.: Error-free coding. IEEE Trans. Inf. Theor. **4**(4), 29–39 (1954)
13. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inf. Theor. **13**(2), 260–269 (1967)
14. Bahl, L., Cocke, J., Jelinek, F., Raviv, J.: Optimal decoding of linear codes for minimizing symbol error rate. IEEE Trans. Inf. Theor. IT-20, 284–287 (1974)
15. Berrou, C., Glavieux, A., Thitimajshima, P.: Near shannon limit error-correcting coding and decoding: turbo-codes. In: Proceedings of the 1993 International Conference on Communications (ICC '93), pp. 1064–1070. Geneva, Switzerland (1993)
16. Third Generation Partnership Project: 3GPP TS 25.212 V1.0.0; 3rd Generation Partnership Project (3GPP);Technical Specification Group (TSG) Radio Access Network (RAN); Working Group 1 (WG1); Multiplexing and channel coding (FDD). http://www.3gpp.org (1999)
17. Third Generation Partnership Project 2: 3GPP2 C.S0002-A. http://www.3gpp2.org (2000)

Chapter 4
Hardware Design of Individual Components

The previous chapters described the basic principles for the application channel coding. Before proceeding to advanced channel coding techniques and its possible hardware realization we will introduce in this chapter the basic steps for a hardware design. An entire receiver is large system and comprises many different functionalities. Combining all of them on a single die yields a so called System-on-a-Chip (SoC). The SoC design requires the knowledge from system specification down to hardware partitioning and refinement. However, every SoC is partitioned in smaller functional blocks which can then be developed individually on component level. This is especially true for the channel decoder which is just one single component in a larger system. The same hold for e.g. demodulator, source encoder or decoder and so on. The advantage of designing components individually is that typically the functionality is restricted and can be well described. In this chapter we first revise (Sect. 4.1) the design flow for a single component and show the different design constraints which are posed either by the communications domain or the hardware domain. Note, that the design flow shown here is no general hardware design flow. It is restricted to communications specific constraints with respect to the introduced base band processing components. Memories are an extremely important part for the entire SoC and for each individual component as well.

For every hardware designer an understanding of the data access patterns and their impact on the choice of the memory architecture, as well as the resulting area and power consumption is mandatory. The basic parameters for the instantiation of memories for the design of individual components are described in Sect. 4.2. The following Sect. 4.3 exemplary shows the design of a simple interleaver component, and how it is heavily influenced by the constraints of instantiated memories.

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_4,

4.1 Design Flow

A generic design flow for individual component design is shown in Fig. 4.1. Every component like the discussed channel decoder in this manuscript is embedded in a larger system. The design flow for a full system is not explained here, the starting point is the isolated functionality like channel coding. Note that many of this small functional blocks exist in a system design which have to be extracted by a system engineer using the divide and conquer method. Every component design flow has input constraints and certain quality measures. Here, the flow starts at the algorithmic design level and ends in a refined model on the so called register transfer level (RTL). This design flow is tailored to components which are embedded in communications systems. For other applications different constraints especially for the quality of service will exist.

Typically we can distinguish between an algorithmic design space exploration and a hardware design space exploration. Different design requirements and quality assessments exist for different levels of the design.

4.1.1 Algorithmic Design Space Exploration

Quality of Service

The quality of service (QoS) is the expected error rate with respect to a given signal-to-noise ratio. This QoS is given by a communications standard or has to be defined by a system engineer. A defined error rate could be: at most 3 bits are erroneous out of 10,000 decoded bits. For every utilized algorithm we have to track the communications performance.

Communications Performance

At each level during the design process we have to ensure that the achieved communications performance meets the given QoS design requirements. The communications performance as an assessment of the quality means e.g. the measured frame error rate with respect to a certain noise level of a channel. Typically the communications performance cannot be evaluated analytically. Thus so called Monte Carlo simulations are performed: the entire transmission chain is model (e.g., using Matlab or C++) and the transmission of information is simulated until the observed communications performance is statistically stable. For every algorithmic transformations the impact on the communications performance has to be checked.

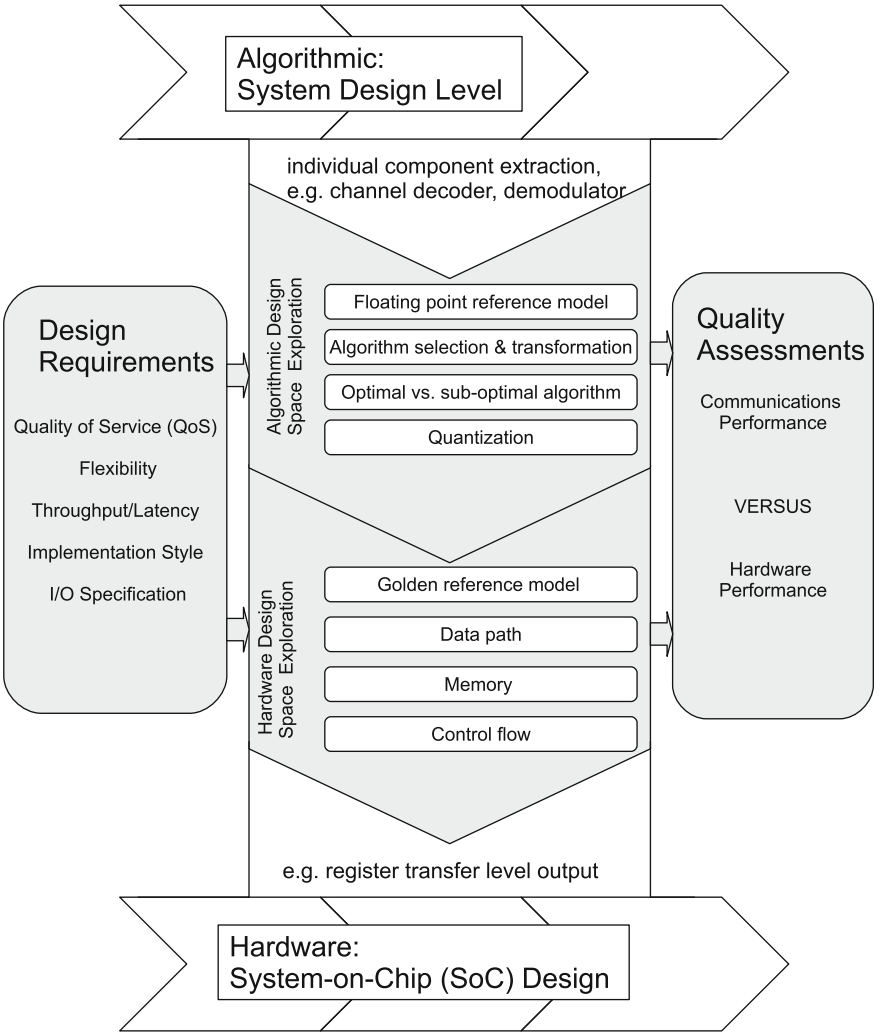


Fig. 4.1 Generic design flow: From component algorithmic extraction to register transfer level

Floating Point Reference Model

The floating point reference model is the time discrete model of the algorithm. It is not important in which language this model exist, e.g. C, C++, Matlab. Each value is represented by sufficient bits (e.g. 32 or 64 bits) to achieve the best achievable communications performance. In many cases this perfect model is too complex for an efficient hardware realization. Thus, sub-optimal algorithms have to be derived which can be implemented in hardware. The degradation of each algorithmic transformation has to be checked with respect to an optimal floating point reference model.

Algorithm Selection & Transformation

The floating point reference model defines one correct realization, often different basic algorithms exist to achieve the perfect performance as well. For example to decode a given channel code we can implement the algorithm in probability domain or in log-likelihood domain or we can change the basic flow of the algorithm e.g. depth first or breadth first algorithm. The algorithm type selection is the first important step towards a hardware realization. This decision step is for sure not a simple one, since a wrong decision at this point may lead to complex or cumbersome hardware realization. At this point the communications performance has to match that of the floating point reference model.

Optimal Versus Sub-optimal Algorithm

After defining the algorithm type the first approximations within the algorithms are introduced. Goal of this approximation is already to limit the complexity of the algorithm. This is a very important design step and requires already experience about a possible hardware complexity. This step is pretty often combined with the floating point to fixed point conversion.

Quantization

During the floating point to fixed point conversion many algorithmic manipulations take place. The communications performance has to be tracked whether it still fulfills the QoS requirements. This step is not a classical quantization step where the quantization noise can be measured, rather it in-cooperates many transformations with respect to sub-optimality of the algorithm. The overall effects can only be measured by Monte Carlo simulations by comparing the final performance degradation with respect to the floating point reference model.

4.1.2 Hardware Design Space Exploration

The fixed point model defines all bit widths of quantized data: input bits, output bits, and all internal values. However, the fixed point model does not define further implementation details. Many further exploration steps have to be performed until a register transfer model will be obtained. A hardware design can be decomposed into three major blocks the data path, control logic, and the memory structure. Different design requirements exist as well for the hardware design space exploration which are shortly discussed in the following.

Implementation Style

As mentioned with the term 'implementation styles' we refer to the choice of different hardware platforms and thus different possibilities with respect to flexibility, power consumption, design time, re-usability, and costs. Different possibilities for the implementation style were already shown in Fig. 1.2.

The choice of the implementation style is a trade-off between performance, power, programmability, and costs. It is typically made even before the beginning of the entire component design flow, since the platform will influence each design decision significantly. Furthermore, the clock frequency is given, or at least, which clock frequencies are available within a larger SoC.

Throughput and Latency Specification

The throughput and latency numbers for the entire communication system is defined in the communications standard. The latency of an individual component has to be derived from the overall system. Each component a dedicated respond time has to be allocated. Starting from the latency and a given clock frequency one can derive the required parallelism of an architecture. The goal is to meet the required latency constraints and the throughput design requirements. However, the interface has to be taken into account as well, as this can influence the final architecture.

Input and Output Interface Specification

The input and output (I/O) interface specification pose further constraints for the hardware designer. It is very important to define the interfaces early during the design space exploration since latency and throughput may be effected by the interface. The interface should be specified in the very beginning of the hardware design. However, in practice due to new requirements this definition changes frequently and thus posses a time consuming challenge for the component designer.

Golden Reference Model

The golden reference model is way more than a simple fixed point implementation. The fixed point model checks the statistical correctness with respect to e.g. communications performance. The golden reference model has to be statistically correct, however, it has to provide a bit accurate behavior with respect to a hardware model. Each internal variable should be modeled as well the correct bit width. Furthermore, often a so called cycle accurate model is required which gives detailed timing information, especially for the interface timing. The timing behavior is important for the system validation and for hardware debugging.

Data Path

The data path is the heart of the processing kernel at which most of the arithmetic operations are performed. Serial or parallel data paths may co-exist within a design. Their basic task is the evaluation of a function $output = f(input)$, where the input and output data may be scalar data or vectors. The parallelism of the data path depends on the required latency/throughput definition. For the design of the data path one has to take care to meet the target clock frequency constraint.

Memory Structure

Memories are important building block for a designer. Typically for component design only static random access memories are utilized which have fast access times. From system point of view, in communications systems up to 70 % of the overall power consumption is due to the memories. Thus, it is of increasing importance that algorithm design takes the memory hierarchy into account. Memory hierarchy means the grouping and organizing of smaller blocks of memory. The memory access, i.e. reading and writing, has to be organized by a controller. Memories are explained in more detail in Sect. 4.2.

Control Flow

Passing the correct input data at the correct time to the data path is one task of a controller. Furthermore it has to organize the message transfer between different data paths and storage devices. The overall control flow of an individual component defines the sequence of individual task which are mandatory to obtain the desired functionality. The control can be either as a software running on a small micro-controller (CPU) or as a dedicated controller instance, depending on the complexity and the task of the controller.

Hardware Performance

The typical performance measures of a certain task with respect to its hardware realization are area and power. Area and power consumption have to be linked with the achieved throughput and with the achieved quality of service. Furthermore the flexibility of the implemented algorithm has to be taken into account. Thus, it is a multi-dimension performance measure with conflicting goals, e.g. lowest power consumption versus highest flexibility. The difficult task of communications performance versus VLSI performance will be addressed in Chap. 9.

4.2 SRAM Memories

Memories are important building blocks. The task of the memory is to store and provide information for different agents which may interpret or manipulate the information. There exist many different types of memories, and each type has different characteristics regarding functionality, accessibility, area efficiency, power efficiency, and of course implementation cost. We can distinguish two major types of memories, volatile memory and non-volatile memories. Volatile memories lose their information when the power is switched off, non-volatile not. Non-volatile memories like read only memories (ROM) or Flash memories are utilized to store long-term persistent data. Flash storage devices cannot be utilized for high performance applications with rapidly changing data content.

In every system-on-chip volatile memories are instantiated with its major storage type called random access memories (RAM). Random access means we can either store or read a data in any order, typically within a given maximum, deterministic time frame. There exist two major classes of random access memories. The static random access memories (SRAM) and the dynamic random access memory (DRAM). Table 4.1 compares the major differences between these two type of memories.

In summary of the table we can extract one major reasons why only SRAMs are possible for the design of an outer receiver. The mandatory bandwidth and a 'truly random' data access can only provided by this type of RAMs. Thus, in the following we only consider design aspects with respect to SRAMs.

Table 4.1 Major differences of SRAMs and DRAMs

	SRAM	DRAM
Technology process	Identical CMOS process with respect to logic and thus can fabricated on the same die	Dedicated process for DRAMs
Access time	Fast and constant access time, independent of the access pattern	Access time heavily depend on the history of access, thus only a worst case time can be specified
Access protocol	Simple access protocol, which means that no special controller is needed	Special controller mandatory
Area and size	SRAMs have typically a limited size ($\sim 8Mbyte$) due to cost and fabrication reasons. One cell to store a bit is typically composed of 6 to 10 transistors, depending on the specific SRAM type	One cell to store a bit is realized by utilizing the capacity of one transistor. Thus, the cell size is up to $\sim$10x smaller compared to that of a SRAM cell. The bit storage to area ratio of DRAMs are very efficient, at least for larger storage demands.
Power	SRAMs have a higher leakage current and dynamic power consumption due to logic technology compared to DRAMS	Very power efficient when large junks of data can be read

Typically we are only interested here in so called synchronous SRAMs, which means everything is triggered by a clock event. The opposite would be asynchronous SRAM where data input and output are triggered by the transition of the addresses.

Figure 4.2 shows the high level view of a SRAM which is mainly composed of a cell matrix, address port, data port, and command port (cmd) together with multiplexers and registers. The cell matrix is composed of individual bits in cells, while one storage cell is typically composed of 6 transistors building a bistable latching circuit. The bit cells are organized in an array, where the word width defines the cells in one row, and word depth defines the number of rows in this cell array. This high level view is one simplistic version of a so called single ported SRAM, with one address port and one data port. Typically for single ported SRAMs we can access exactly one word per clock cycle, either read one word or write one word, respectively. Of course there exist many different types of SRAMs which for example can read and write one data within one clock cycle. These SRAMs are called dual ported memories and will be discussed later. The bit pattern at the address port defines the row number in the cell matrix. The demultiplex after the address port is called address decoder which ensures the correct address in bits to row number. Note, that the number of addresses can be quite large, here indicated with 4096 rows. The address decoder and the mandatory wires to the corresponding cells can have

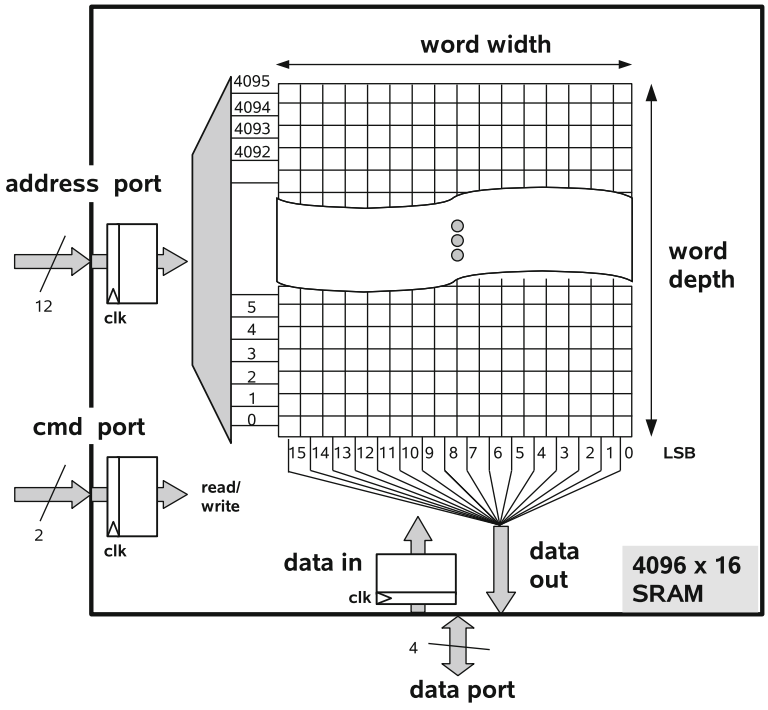


Fig. 4.2 High level view of a SRAM memory

a large influence on power of the overall SRAM which is explained in more detail in the next section. Via the command port we typically define the mode of access, either reading or writing. It is as well possible to put the memory in idle mode which means a reduced static power consumption.

The registers as indicated in Fig. 4.2 can best be understood when looking at the clock diagram for reading and writing the SRAMS. Figure 4.3a shows the timing diagram of a typical read access to a SRAM memory. The address data is passed to the input port of a SRAM memory. This address data is then latched in an internal register with the next rising clock edge. The data which can be read form the output port is then stable at the next clock cycle. We can read one new data at each clock cycle, while the corresponding address has to be passed one clock cycle before to the address port. This behavior is indicated in the high level SRAM view as shown in Fig. 4.2 which shows an input register at the address port but no output register for the data port.

Figure 4.3b shows the timing diagram of a typical SRAM writing access. The address and the corresponding data have to be passed at the same clock cycle to the SRAM ports. The address and data are internally latched and then the data is written, in the mean while the next data and address can be passed to the SRAM ports. This means that the SRAM accepts new data at each clock cycle.

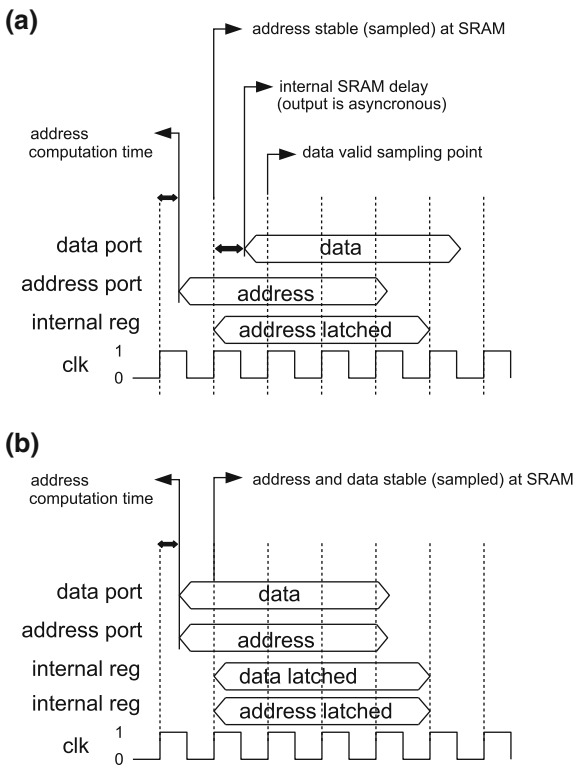


Fig. 4.3 Reading and writing process of a SRAM memory. **a** SRAM read access. **b** SRAM write access

4.2.1 SRAM Design Space Exploration

As already mentioned the SRAM memories are one important building blocks for the design of individual components. The hardware designer has to care about the instantiation of the memory arrays. Many different SRAM can be instantiated with different shapes. The shape defines the word depth and word width. It is obvious that SRAM memories with different shapes result in different area numbers and power numbers. The designer would like to check what is the influence with respect to area and power when instantiating different types of SRAMs. What is the area benefit if my algorithm works with one bit less quantization? What is the expected power consumption if I have to access the memory every clock cycle? What is the difference in e.g. area when instantiating shapes of 512×8 versus 128×32 (both memories store the same amount of overall bits)? What is the maximum design frequency the memory can operate on?

To get answers on these question we have to explore the memory characteristics. Each manufacturer offers another possibility how to analyze SRAM data, which we denote in this manuscript as SRAM memory explorer. **Note that the following data and naming are artificial and not trailered to one specific manufacturer**. The SRAM memory explorer gives you the relevant design information for a given SRAM design, described by its characteristics under specified conditions. It gives here an example of SRAM output characteristics of a SRAM input characteristics request. In the following we will explain the input parameters and output parameters of apossible memory explorer. The input parameters define the SRAM for which we request the design information and its operating condition and are mandatory to narrow the search space.

Even the number of possible input parameters might be very large. For that reason we divide the input request in primary and secondary input parameters. The primary input parameters are mandatory to be defined by the designer, while for setting the secondary parameters a more precise knowledge of the internal memory structure is mandatory.

Primary Input Parameters

- Technology
  Of course the feature size or process type is one of the first parameter with has to be specified. As shown in Fig. 1.1 the size of the technology node changes rapidly. Each technology node is coming typically with different SRAM types like
  - Low power: SRAMs are optimized with respect to low power consumption, typically for mobile applications we assume a so called low leakage technology.
  - High performance: SRAMs are optimized with respect to fast access times and thus high frequencies.
  - Regular: SRAMs which trades off low power and high performance which are of course typically counteracting parameters.

- WordDepth
  The word depth parameter defines the number of words which can be stored in the memory.
- WordWidth
  The word width parameter defines the number of bits per of each word.
- Process
  For the process we have to distinguish between slow and fast. Due to process variation the switching time of transistor differs. Slow process defines e.g. the access time at -3σ of the process, while fast process at $+3\sigma$ respectively. An architecture has to be designed with respect to both conditions, slow and fast, since we do not know the exact resulting switching time of each individual fabricated chip. Thus, the critical path check should always be done on the slow process cycle time, while the so called hold time check has to be done on the fast process. The hold time specifies the time span a data has to be stable after a rising clock edge. This time span is mandatory to latch the data e.g. in a register.
- Voltage
  Each memory is specified with different voltage levels (nominal, worst, best). The design which is done with respect to a nominal voltage e.g. 1. V has to work as well at the respective worst case voltage ($-10\,\%$) of 0.9 V. Thus the cycle time has to checked as well for worst case Voltage since this cycle time may be twice as large. Power is typically checked on nominal voltage level, however again worst case power consumption should be done on $+10\,\%$ voltage level.

Secondary Input Parameters

- Power Off
  Some memory types allow to switch off the core cell array, where we have to specified if we would like to have a pin for such a power off mode or not. One has to remember since the SRAM is a volatile memory a power off mode will always delete the memory content.
- Sleep Mode
  Again an additional pin is generated if a sleep mode or sometime retention mode is desired. A sleep mode switches off the periphery and thus reduces the leakage of the memory. The memory content will be preserved.
- Write single bit
  Some memory types allow to write individual bits. These special property will increase the overall area since additional control logic will be mandatory to enable the addressing of each individual cell.

Primary Output Parameters

- Area
  The area occupied by the specified memory is one of the key numbers, and typically given in mm^2. This number is very accurate even for the final physical design.

Remember that for logic area after synthesis a significant overhead for place and route may occur may occour (e.g. +30 %), due to clock tree or additional design for test structures.

- Power read nominal (Preadnom)
  This number defines the average power of a read under the assumption that half of the address bits are switching. All power numbers of a memory are specified by $\mu W/MHz$, i.e. the specified power consumption is normalized to the system frequency f. The power consumption of a memory results in

$$P = A_{ac} \cdot Preadnom \cdot f. \tag{4.1}$$

  With $A_{ac} \in [0, \ldots, 1]$ the access patter. $A_{ac} = 1$ means that the memory is accessed for a read at each clock cycle.
- Power write nominal (Pwritenom)
  This number defines the average power of a write under the assumption that half of the address bits are switching and half of the data bits are switching. The power consumption of writing a data is slightly larger than that of a reading access ($\leq +10\%$)
- Read access time (T_{acc})
  The read access time specifies the (internal) time a memory requires to read a data. It gives you an indication about the amount of additional logic one can instantiate between the input/output memory port and the next register. Thus the critical path is composed of $T_{cyc} \geq T_{acc} + T_{logic}$.
- Cycle time (T_{cyc})
  The cycle time determines mainly the maximum frequency of the design. For smaller design frequencies one access can be performed at each clock cycle.

Secondary Output Parameters

- Aspect Ratio
  Defines the ratio between the geometries Height/Width.
- ColumnMux:
  A memory array may be folded for good aspect ratio. In Fig. 4.4 one possible example is shown. The memory is instantiated with parameters 4096 × 16 (rows×columns). However the internal storage structure may be folded into a 2048 × 128 memory. However, to select 16 output bits form the 32 columns requires 16 times a 2:1 column multiplexer. Thus for this example the 'Mux' parameter would be ColumnMux = 2.
- Output Capacitance
  This parameter defines the maximum capacitance which the outputs are able to drive.
- Width and Height
  The physical width and height of the SRAM cell in mm.

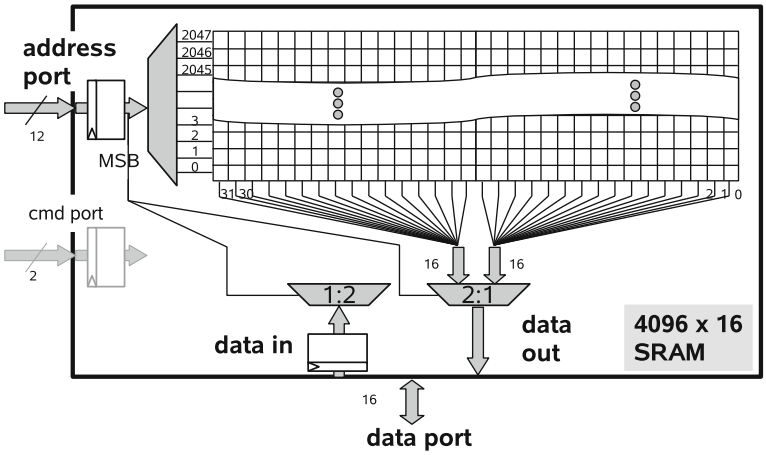


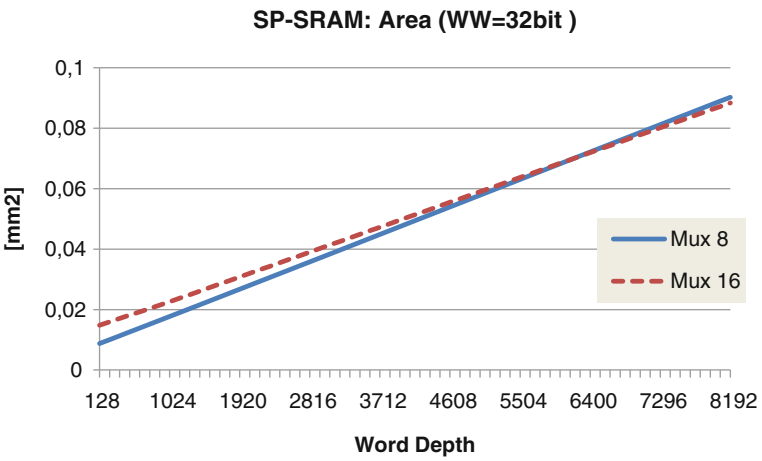
Fig. 4.4 High level view of a SRAM memory with **ColumnMux=2**.

- Density
  The density is defined as Kbits/mm^2 and gives you an indicator about the area efficient of the memory.

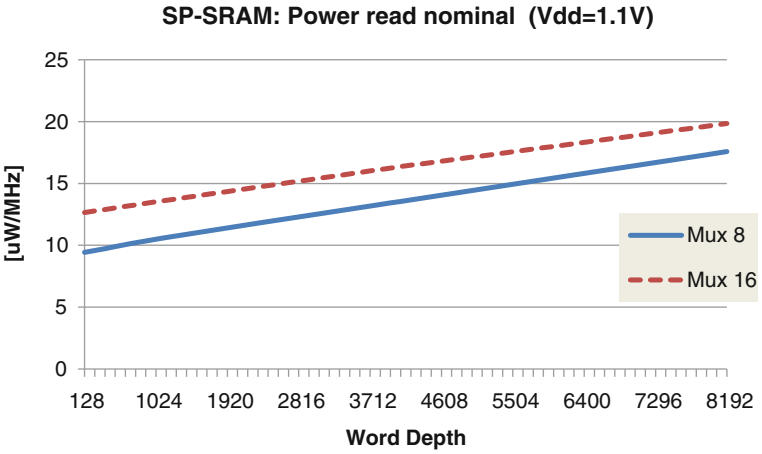
4.2.2 Exemplary SRAM Data: Area, Power, Cycle Time

In this section we show exemplary the results for area, power, and cycle time. The utilized technology for demonstration is a 40 nm low power technology with $V_{dd} = 1.1$ nominal voltage. Note, that the numbers are derived from an existing technology, however, the results are changed with respect to the absolute numbers and adjusted for educational purposes. Shown are different memory types like:

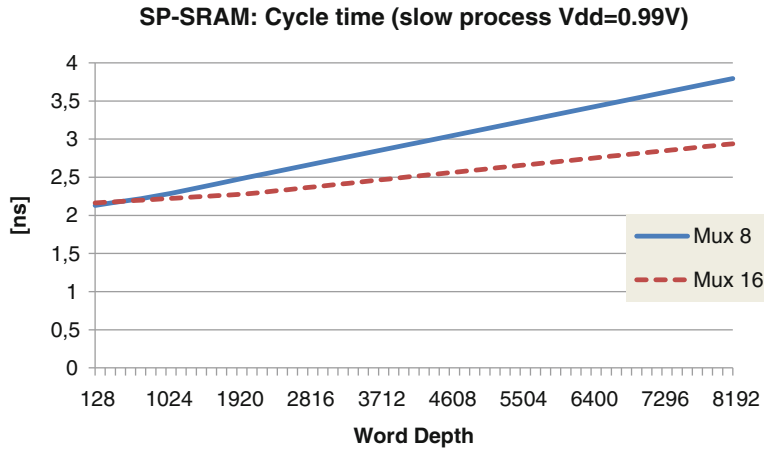
- **SP-SRAM**: single ported SRAM memories.
- **XS-SP-SRAM**: extra small single ported SRAM memories which are designed for a small storage requirement.
- **DP-SRAM**: dual-ported SRAM memories which enable the reading of two values and the possibility to write and read one value within a clock cycle.



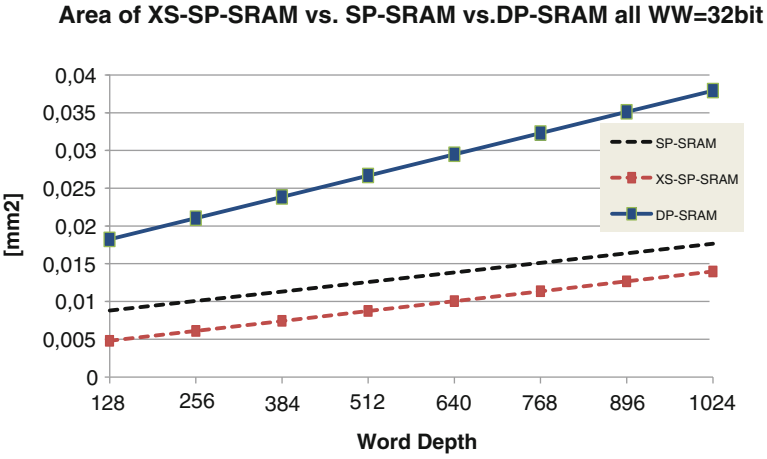
The graph shows the area trends for a *WordWidth* of 32 bits and different *WordDepth* values, ranging from 128 addresses to 8192 addresses respectively. The two lines represents the area for two different *ColumnMux* factors. The area is given in mm^2.



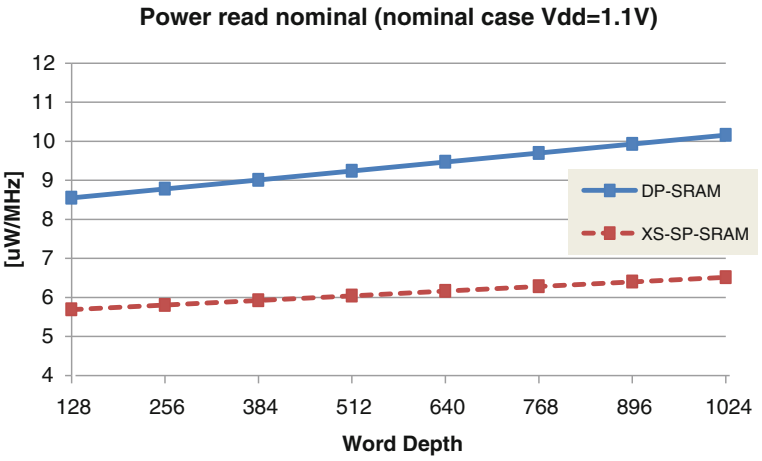
The figure shows the average power read operation (Preadnom) at which half of the addresses are switching. It is important to show the numbers for a nominal voltage case of $V_{dd} = 1.1\,\mathrm{V}$. The two *ColumnMux* factors show at least up to 8192 addresses no break-even point. At least from perspective of power consumption one would decide in favor of $Mux = 8$.



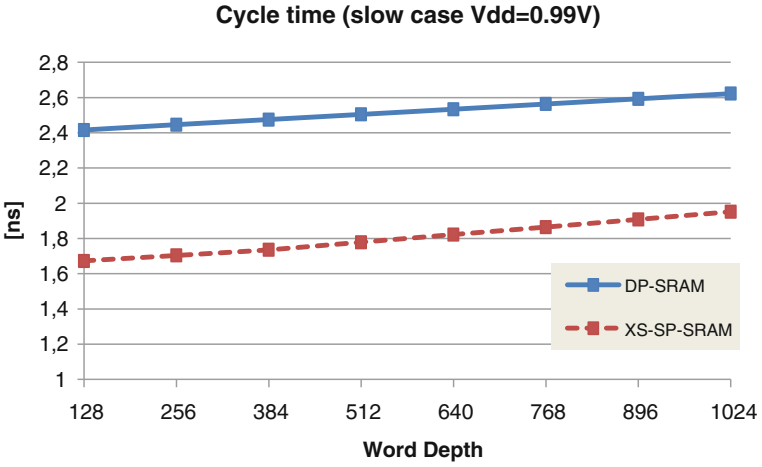
The graph shows the cycle time for the LPHDSPSRAM again with a word width of 32 bits. Attention the cycle time has to be checked for the worst case assumption, which is the slow case of $V_{dd} = 0.99$ V. The two lines represents again the area trends for two different ColumnMux factors. However, this time the larger Mux factor shows a etter cycle time behavior.



The graph shows the area trend lines for three different memory types. The chosen *WordDepth* is quite small with a maximum of 1024. Not surprisingly, the special optimized memory for small storage demands XS-SP-SRAM shows the smallest area. The dual ported memory shows at least in this range a large area compared to both single ported SRAMs.



The figure shows Power read nominal case for the DP-SRAM and XS-SP-SRAM memories (WW = 32 bits, Mux = 4). Again the overhead of the power consumption for an access of a dual ported memory gives a strong argument to avoid dual-ported memories. Note, that these memories here allow a full simultaneous wright/read of two values. Thus the entire control logic, I/Os, and internal routing has to be doubled.



The graph shows the cycle time for the XS-SP-SRAM and DP-SRAMs again with a word width of 32 bits. The Mux factor was chosen to obtain the best possible cycle time for both cases. The cycle time overhead for the dual ported SRAMs can be significant. When using dual-ported SRAMs, special care has to be put on the cycle constraint given by the system constraints.

Summary SRAM Exploration

Figures 4.5 and 4.6 show the area and the power trend for the three different memory types. Both figures show the result for the slow process with $V_{dd} = 0.99\,\text{V}$, thus the best case power consumption. However, the intention of the figures is to show the relative area and power and the area and power trends for different memory types with respect to bit width and word depth. For every memory type three different, typical word width $WW \in \{8, 16, 32\}$ bit are given. One important fact can be seen, that each memory type is designed for a specific range of word depth. Especially the SP-SRAMs are designed for very large number of words. This can be seen when comparing the slopes of the extra small memories and the large single ported memories. Between ~1024 and ~2048 the area slopes will cross. Attention in the figures: no MuxNumbers are given, thus, it is just an indication of sizes and trends. For the designer we can summarize the most important issues for working with the memory explorer or a similar tool.

- Different nominal cases for the memory exist, e.g. $V_{dd}(nom) = 1.0\,\text{V}$ or $V_{dd}(nom) = 1.1\,\text{V}$.
- Derived from the nominal case a slow and fast case exist with $V_{dd}(slow) = 0.9 \cdot V_{dd}(nom)$ and $V_{dd}(fast) = 1.1 \cdot V_{dd}(nom)$.
- The cycle time has to be checked for its worst case assumption, which is $V_{dd}(slow)$.

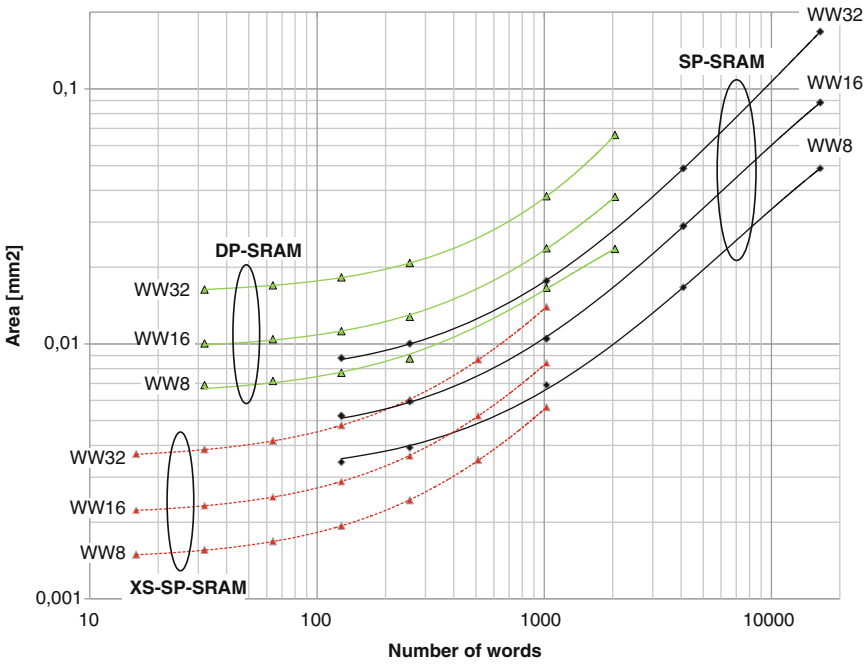


Fig. 4.5 Example of the area for different memories types, all in 40 nm technology

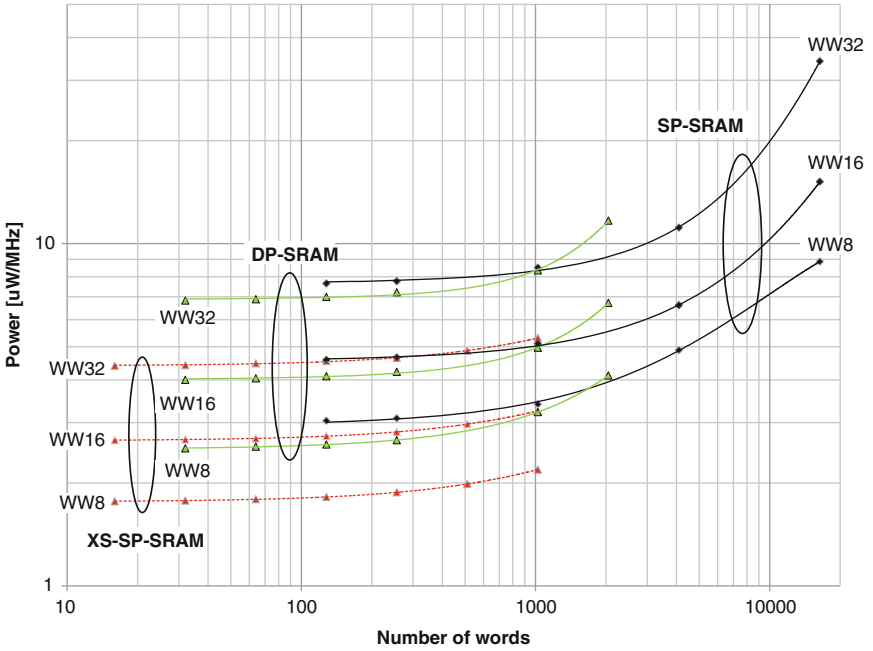


Fig. 4.6 Example of the power for different memories types, all in 40 nm technology

- The power consumption should be investigated for the nominal case $V_{dd}(nom)$.
- The ColumnMux factor defines the internal structure of the memory. Different ColumnMux factors result in different, sometimes contradicting, area, power and cycle time numbers.
- When possible, the instantiation of dual ported memories should be avoided, since area and power are larger than for single-ported memories with identical storage capabilities.

Which SRAM memories are instantiated during the design of a component, and whether it is possible to avoid the instantiation of dual ported memories, depends on the access pattern of the application. Access pattern defines the data access in time and space (location) and depends of the functionality we would like to implement. One example with a difficult access pattern is presented in the next section.

4.2.3 Importance of Memory Exploration

We have seen in the previous section that we can instantiate different types of memories. For the design of a digital baseband receiver exploring the different options gets more and more important. In future designs it is expected that the size of memories with respect to the overall chip area will increase further. For example Fig. 4.7

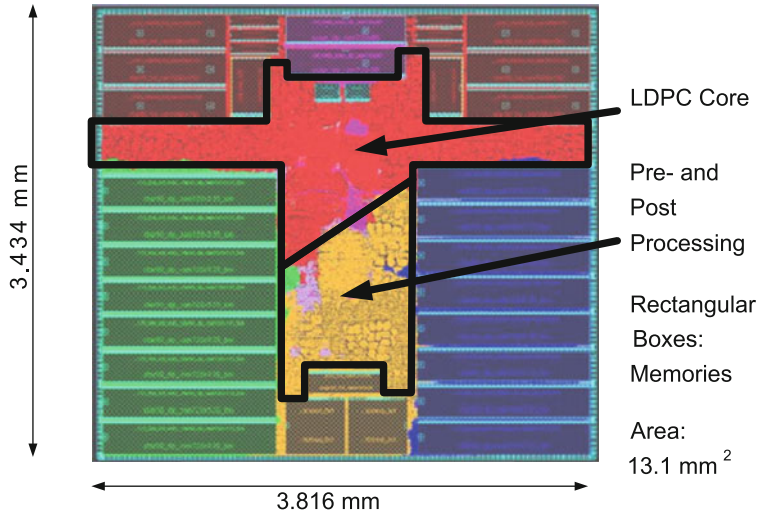


Fig. 4.7 Chip photo after place&route of product DVB-S2 receiver chip, 70 % of the overall area is determined by instantiated memories.

shows the final place and route ($P\&R$) layout of a low-density parity-check (LDPC) decoder which was designed for a DVB-S2 receiver product.

LDPC codes are explained in Chap. 7 while the entire design of this particular decoder is described in [1]. Here, we only consider the memories, which determines 70 % of entire area. The instantiated memories are indicated by the boxes.In this design we have to store in total ∼2Mbits of data. 960 bits are read and written in each clock cycle. An SRAM featuring a bitwidth of 960 bits does not exist as a monolithic building block. Thus, multiple memories have to instantiated to enable the access of 960 bits per clock cycle. The designer now has the possibility to chose a possible fragmentation to achieve the required memory access bandwidth. A so called memory hierarchy is introduced. Note, that the term memory hierarchy is often used in large systems and defines the organization of memories of different types (DRAM, SRAM) in which each storage element may have a different respond time. Here, we use the term memory hierarchy to emulate a large SRAM memory array for an application while the internal structure is fragmented. This is indicated in Fig. 4.8 for two different setup to emulate the required SRAM shape of 2048×960. Either we instantiate 15 SRAMs of shape 2048×64 or we could even instantiate 240 SRAMs of shape 256×32. Both possibilities are shown in Table 4.2. Given are the corresponding data in terms of area and power of a single instance and the overall expected numbers, respectively. In case of a high fragmentation the extra small memories are assumed since these are optimized for the instantiated shape. Assuming the low fragmentation case the data for area and power correspond to standard single ported SRAMs. The table shows clearly the trade-off a designer has to be aware of. The power is optimized for the case of the highly fragmented instantiation, while the area is optimized for the low fragmentation variant.

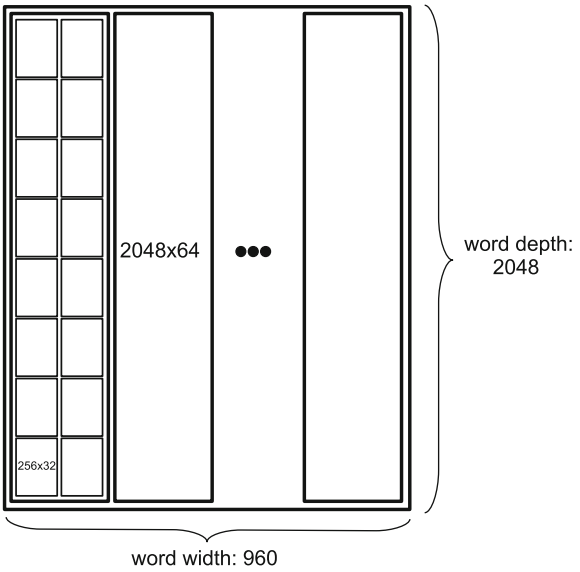


Fig. 4.8 Two different memory hierarchies to enable the access of words with 960 bits

Table 4.2 Two different memory organizations to enable an access of 960 bits per clock cycle.

Case memory type	High fragmentation XS-SP-SRAM	Low fragmentation SP-SRAM
Shape	256×32	2048×64
Single area	$0.006\,mm^2$	$0.05\ mm^2$
Average Power (Preadnom+Preadwrite)/2	$5.8\ \mu W/MHz$	$22\ \mu W/MHz$
Number of instances	240	15
Total number of stored bits	1.96 Mbits	1.96 Mbits
Total area estimate	$1.44\,mm^2$	$0.75\,mm^2$
Total power estimate	52.2 mW	99 mW

The total power estimate assumes here a frequency of $f = 300MHz$ by evaluating Eq. 4.1. The power and area numbers given here are only estimates. $P\&R$ will influence these results again since the data have to be routed to the corresponding memories. Thus, the area and the power consumption will be higher in both cases. The large influence of the $P\&R$ is further highlighted in the design example presented in Sect. 7.5.

In the example here we did not assume any constraints about how to access the 960 bits. It is of course a huge difference if these 960 bits have a regular access pattern or a random access pattern. In all cases the designer has to design the controller to ensure the correct addressing across instances of memories. One example which often requires a high fragmentation due to its difficult access pattern is presented in the next section.

4.3 Individual Component: Interleaver

In this section we will show exemplary the considerations a designer has to make when designing an individual component, in this case an interleaver. We will see that its design, at least for higher throughputs, can be difficult due to requirements to the memory architecture.

4.3.1 Interleaver Types

In communication systems interleavers are used in many different components. For wireless transmission systems, typically, block-based interleavers are used, which change the location of a bit or symbol within a block, where the term block means either a codeword, multiple codewords grouped to a frame, or only a part of a codeword. Thus, we have to distinguish between:

- inter-frame interleaving: interleaving across multiple frames,
- intra-frame interleaving: changes the position of bits within a codeword, as in the case of bit-interleaved coded modulation as shown in Fig. 2.1, and
- channel code interleaving: the channel code interleaving is the bit permutation within a part of the codeword to ensure randomness of the code. This is applied in e.g. turbo codes or LDPC codes, see Chaps. 6 and 7.

The task of any interleaver is to break up dependencies between adjacent positions within a data stream. For example, the inter-frame interleaving ensures that burst errors are spread via multiple frames. Burst errors are errors on consecutive positions within a transmission stream, which may occur when transmitting via fading channels. An interleaver spreads these uncertain locations over a larger distance.

An interleaver uses a bijective function which maps the indices of an input sequence to changed indices of an output sequence. An interleaver table Π is one realization of this index mapping, i.e. it defines the one to one positional mapping from input position to output position of a given interleaver. Typically i defines the index in the interleaved block when we speak about $\Pi(i)$. Thus the interleaved data sequence can be derived by indirect addressing:

$$\mathbf{x}(\Pi(i)) = \mathbf{x}'(i) \tag{4.2}$$

$\mathbf{x}$ refers to the data vector in the original order, the vector $\mathbf{x}'$ is the interleaved vector at the output of the interleaver. It is also possible to define a direct addressing. However, to obtain the same output sequence of $\mathbf{x}'$ the inverse interleaver table has to be derived, which is typically indicated as Π^{-1}.

$$\mathbf{x}(i) = \mathbf{x}'(\Pi(i)^{-1}) \tag{4.3}$$

Many different possibilities exist to generate an interleaver table for a specific block. Which one to use depends mainly on the application. In the following we will present four different methods to generate an interleaver table. Three of these interleavers are actually used in today's communications standards. Figure 4.9 plots the output position over the input position for four different interleavers operating on blocks, all with a block length of 80 bits. The x-axis shows the input position, the y-axis the corresponding output position.

Figure 4.9a reflects the output of a so called random interleaver. The random interleaving shows no special structure and is typically not utilized in communications systems. Figure 4.9b shows the pattern for a classical block interleaver. Note that the name block interleaver denotes just a special type of a block-based interleaver, but

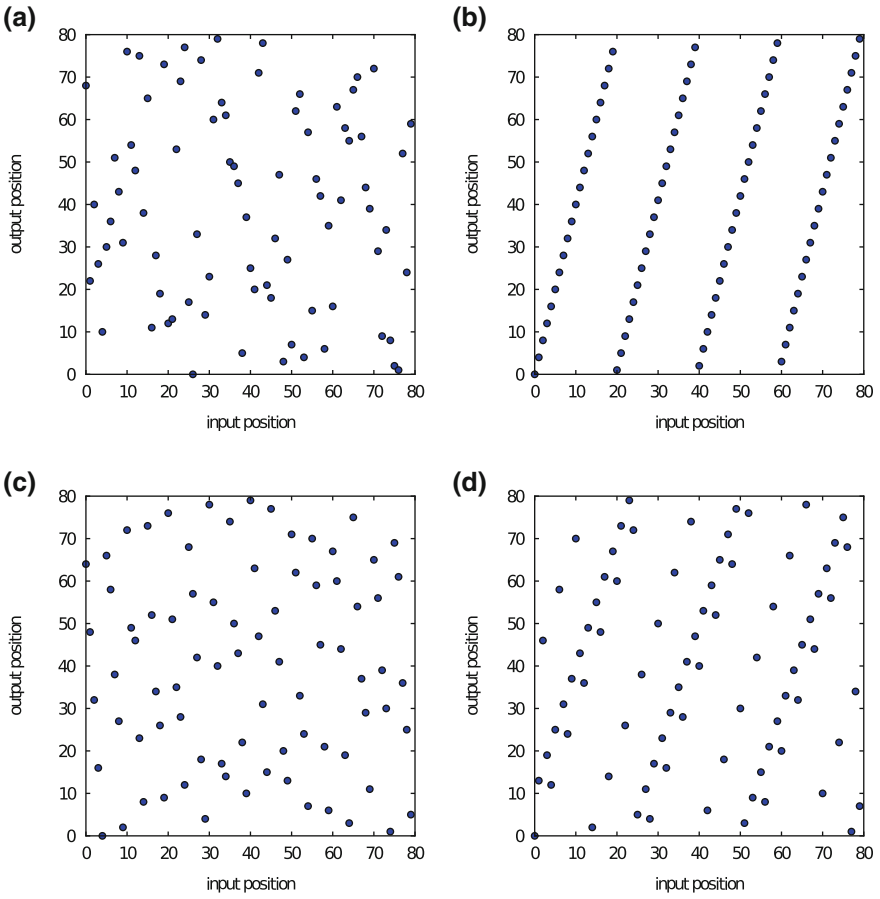


Fig. 4.9 Illustration of the permutation pattern of 4 different interleavers, all with block length of N = 80: **a** random interleaver, **b** block interleaver, **c** UMTS channel code interleaver, and **d** LTE channel code interleaver

these terms are not to be confused. The typical procedure of a block interleaver is to write the data stream in an array in a column by column fashion. The output stream is generated by reading the content row by row. In Fig. 4.9 we utilized an array with $C_1 = 20$ rows and $C_2 = 4$ columns. The generation of the interleaver table Π can be described with

$$\Pi(i) = (i \bmod C_1) \cdot C_2 + \left\lfloor \frac{i}{C_1} \right\rfloor \tag{4.4}$$

C_1 and C_2 are the two dimensions of the block interleaver with the overall block size of $C_1 \cdot C_2$. Block interleavers are used in communication standards very often, since they provide a simple and often effective permutation.

In addition there exist a so called cyclic block interleaver. The cyclic block interleaver needs a further vector for its description, $\boldsymbol{I}^{offset}$. The offset vector has C_2 entries, one for each column. The values define the start index at which each column is filled. For the interleaver plotted in Fig. 4.9b this offset vector would be

$$\boldsymbol{I}^{offset} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}. \tag{4.5}$$

Here, all columns are written starting from the very first position top to bottom.

Plot (c) shows the interleaver which is instantiated within the UMTS channel code encoder (turbo encoder). The interleaver construction is called permuted block interleaving and is based on a classical block interleaver. Again, the input block is written to an array row by row. However, before the reading step, the position of the columns and rows are permuted as well. With this additional row and column permutation stage a quasi random interleaver pattern is obtained, even so, with a deterministic procedure to calculate this interleaver pattern within an application.

In UMTS a different interleaver table has to be generated for each block length ranging from 40 to 5114 bits. Thus, the granularity of this interleaver generator is 1 bit. Despite its deterministic generation procedure, for a hardware realization of the turbo decoder it turned out that the specified procedure to generate the interleaver tables has two disadvantages. Especially for high throughput applications, where spatial parallelism is required two problems have to be solved:

- The generation of the interleaver tables is relatively complicated and may require many clock cycles. Producing multiple indices of the interleaver table in one clock cycle is possible but cumbersome.
- The second problem is more difficult to solve. Parallel processing requires parallel interleaving of data. Memory access conflicts may occur, which have to be resolved, see Sect. 4.3.2.

Due to these two problems a new interleaver type was defined in the newer LTE standard. Figure 4.9d shows the permutation pattern of the LTE channel code interleaver (turbo encoder). It has a very simple construction rule to generate the interleaver table:

$$\Pi(i) = \left(f_1 \cdot i + f_2 \cdot i^2\right) mod K \tag{4.6}$$

Table 4.3 Overview of interleaver types the block size and granularity their characteristics

Standard	Interleaver	Classes	Smallest block	Largest block	Granularity in bits
	TTI frame	Block interleaver		150 k	8
LTE [3]	Sub-block	Block interleaver	120	18 k	8
	Channel code	ARP interleaver [4]	40	6144	8
	TTI frame	Block interleaver		150 k	8
UMTS	Sub-block	Block interleaver	120	15 k	8
[5]	Channel code	Block permuation	40	5114	1
DVB-S2	Intra block	Block interleaver	16.4 k	64.8 k	360
[6]	Channel code	Quasi-cyclic	80 k	240 k	360

f_1 and f_2 are interleaver parameters defined in the standard and K is the block length. The granularity is 4, 8, or even 16 for larger block sizes. This interleaver type is called quadratic polynomial permutation (QPP) interleaver [2].

In summary we can say that different interleaver types are utilized in communications standards. The interleaver has a major impact on the resulting communications performance while different interleaving types within a transmission scheme are mandatory. For modern communication standards the choice of the appropriate interleaver type is often determined by the resulting communications performance and the efficiency of possible hardware realizations of that interleaver.

Table 4.3 shows the different interleaves which are utilized within one communication standard. For example the LTE standard features an interleaver for the transport transmission interval (TTI) frame which interleaves up to 1,50,000 positions. Then, an interleaver stage for the coded codeword is utilized with a maximum block length of 18,000 indices, furthermore the channel coding itself has an interleaver inside. Here, the block sizes for the interleaving ranges from 40 to 6144 bits. The granularity in the table defines the step width between to block sizes. The LTE turbo code interleaver has a granularity of 8 and for larger block sizes even 16, the UMTS turbo code interleaver has a granularity of one. Thus a hardware realization for the UMTS channel code interleaving has to realize all possible block sizes between 40 and 5114 bits. Thus, it is of importance if we can compute the interleaver tables in hardware or whether to compute the corresponding interleaver tables offline. Depending on the throughput requirements different challenges for the hardware realization exists.

4.3.2 Hardware Realization

This section discusses the different possibilities which can be applied to the storage of data in an interleaved order. First we will derive a serial architecture for interleaving, followed by an architecture which interleavers P data in parallel. Assuming a serial architecture, we always store a data block in original order in one memory

while the data with changed indices position is stored in a second memory. Furthermore, we assume that the interleaver tables are stored as well in memories. For the serial interleaver architecture we can identify three major methods to the interleaving/deinterleaving.

- **Interleaving by reading** (Fig. 4.10a)
  The memory which stores the interleaver table is read sequentially. The output data of this memory is used as address for data memory A. The retrieved data is then written to data memory B at consecutive addresses. Thus, we read the content DATA A in an interleaved manner, which is denoted as interleaving by reading.
- **Interleaving by writing** (Fig. 4.10b)
  The data memory A and the deinterleaver memory are read sequentially. The output data of the deinterleaver memory is used as address for data memory B. Thus, the retrieved data is written in an interleaved manner to data memory B. The content of the address memory is different to the first case and is called deintereaver table.
- **Interleaving in two stages** (Fig. 4.11)
  The interleaving is performed in two stages. Stage one writes the data in an interleaved manner to an intermediate data storage (DATA TMP). Interleaver Table 4.1

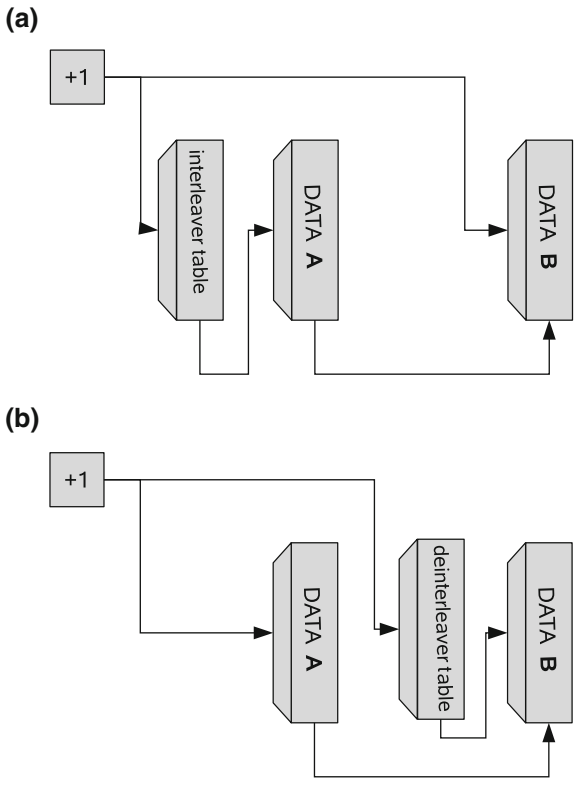


Fig. 4.10 One stage interleaving: Interleaving from memory A to memory B upon read (*top*), upon write (*bottom*)

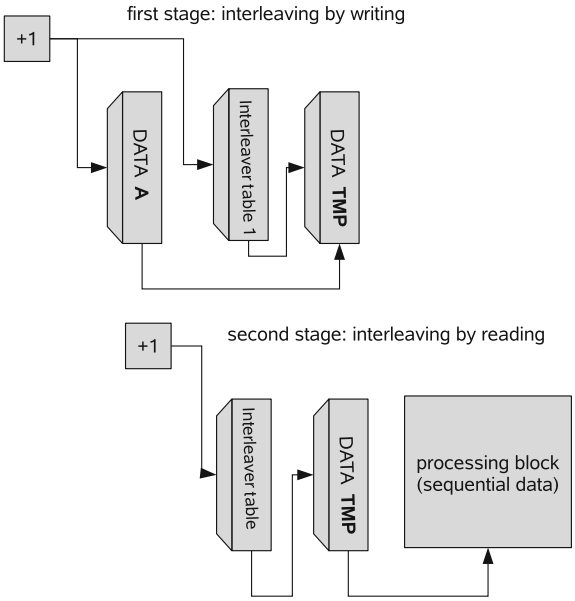


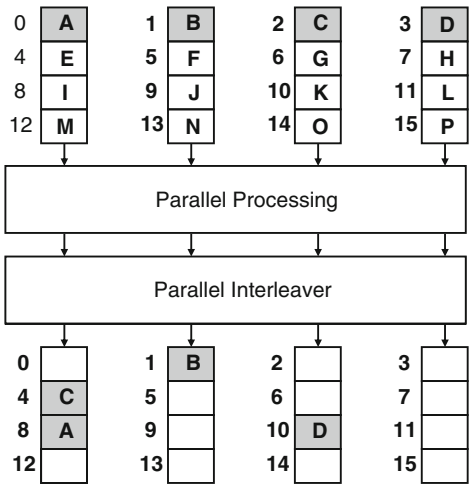
Fig. 4.11 Two stage interleaving: the content of address Tables 4.1 and 4.2 are different.

does not hold the final interleaving table. The final interleaved data sequence is obtained after the second stage, which is here indicated as an input to a processing block. The second stage performs an interleaving by reading. The content of interleaver Tables 4.1 and 4.2 are different and have to be derived from the overall interleaver table.

Within all presented serial architectures we have instantiated memories for the interleaver tables. However, when it is possible to generated the interleaver tables on the fly we can replace these memories by an appropriate logic block. On the fly means we have to provide the correct target address for reading or writing at each clock cycle.

Parallel Interleaving

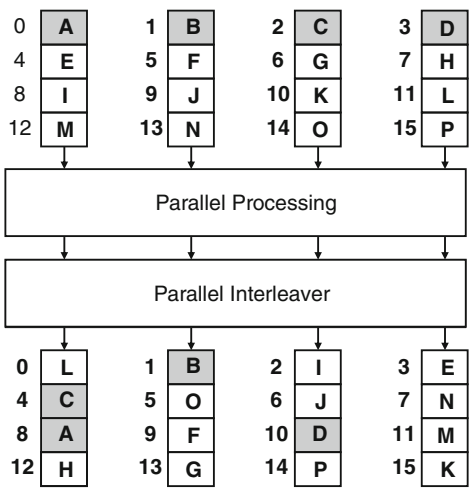
So far we have presented only possibilities for a sequential interleaving, which means we need N clock cycles to interleave N values. Increasing throughput demands requires a parallel interleaving. Multiple data have to processed and, consequently, also interleaved in one clock cycle. Figure 4.12 shows the major problem of parallel interleaving. On the right side the address table with the corresponding interleaving address is shown. The data flow for the architecture is top down. 4 single ported SRAM memories are instantiated in front of a parallel processing unit. The data A,B,C,...,P are stored in the four input memories, while the numbering next to each memory indicates the original address labeling of the interleaver table. The parallel



Addr	Interl. Addr.
0	8
1	1
2	4
3	10
4	3
5	9
6	13
7	12
8	2
9	6
10	15
11	0
12	11
13	7
14	5
15	14

Fig. 4.12 Parallel Interleaving and memory access conflict for the very first processing step

processing unit accepts now four values per clock cycle, i.e., in the first clock cycle it processes data A,B,C,D. After the processing we would like to write the four output data in an interleaved order. However this is not possible within one clock cycle since data A and C have to be written to the same output memory We call this a memory access conflict. Figure 4.13 shows the data after all data are written. In this small example at all four clock cycles a memory access conflict occurs.



Addr	Interl. Addr.
0	8
1	1
2	4
3	10
4	3
5	9
6	13
7	12
8	2
9	6
10	15
11	0
12	11
13	7
14	5
15	14

Fig. 4.13 Parallel Interleaving and memory access after all values are processed

One of the most elegant method to resolve the parallel interleaving problem is two utilize a two stage interleaving process. As shown in Fig. 4.11 the two stage interleaver requires two interleaver tables. Each stage interleaves P data in parallel without memory access conflicts. The trick is the intelligent pre-computation of the two appropriate interleaver tables. In [7] it was shown that it is always possible to find a two stage procedure to interleave P values without memory access problems.

Currently, there is no deterministic algorithm known which can calculate the two address tables on the fly, thus a pre-processing to determine the two interleaver tables has to be performed Typically in practical applications all interleaver patterns are pre-calculated and stored in external memories. Assuming a new block size the corresponding interleaver tables have to be loaded to corresponding interleaver table memories. This parallel interleaving problem occurs also for example in UMTS turbo decoding which features 5000 different block length and thus as many varying interleaver pattern. The storage of the pre-computed interleaver patterns is quite large and exceeds the storage demands of the turbo decoder architecture itself.

Joint Algorithm-Hardware Design

Interleavers are defined in many communication standards. Since the throughput demands of nearly all standards steadily increases we have to implement the interleaving sooner or later as well in a parallel manner. As seen, this posses problems for the implementation. However, it is possible to design an interleaver which provides parallel processing without any memory access conflicts. For that we have to consider jointly constraints from a possible hardware realization and constraints for the algorithm in this case an appropriate permutation pattern.

The idea how to achieve this is quite elegant and often denoted as **joint algorithm-hardware design**. Figure 4.14 shows the first step for designing an interleaver table which allows a conflict free parallel interleaving. We have seen that in the first clock cycle there was an access conflict. We would like to write data A and C to the same memory. The idea now is to replace the original destination address of data C by a new address which allows a conflict free storage. For the first clock cycle in this example we have here 4 possible address which have no conflicts. In theory we could choose any of these for possibilities, nevertheless, since the interleaving has to achieve a certain goal with respect to an application, additional constraints may exist. For example a clustering of addresses is not allowed, see Fig. 4.9a in the case of a random interleaving. A clustering of address indicates that e.g. a burst error would not be spread across the block. The interleaver table is now filled step-by-step, always preventing a possible conflict at each construction step. The entire 'conflict free' interleaver pattern can be obtained by Heuristics as explained, or as well by algebraic methods [4]. The interleaver design method, however, has always a parallelism level in mind, at which memory access conflicts can be prevented. Thus the joint algorithm-hardware design requires knowledge of both domains. The hardware gives us constraints with respect to an initial architecture template and defines also

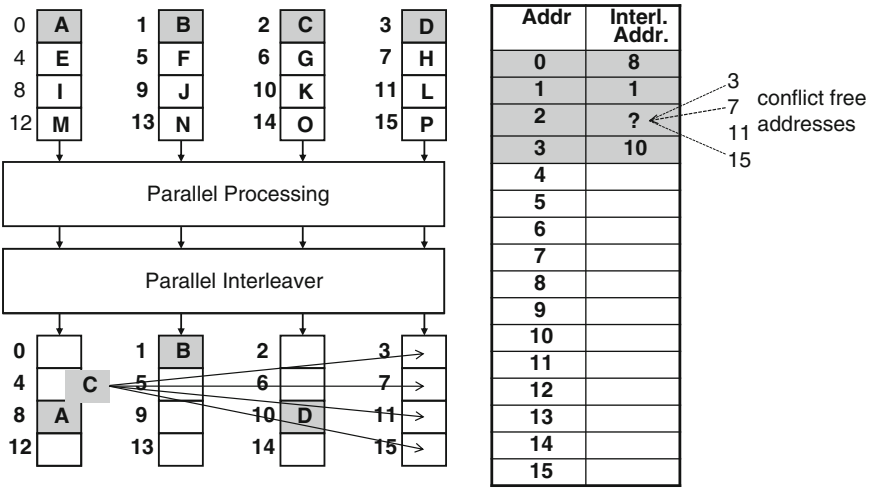


Fig. 4.14 Designing an interleaving table which allows a parallel interleaving without memory access conflicts

as well a target parallelism, while the algorithm or application has constraints with respect to functionality.

The idea of joint algorithm-hardware design was already applied in the design of communications standards. The (turbo code) interleavers of the LTE standard are designed with respect to hardware knowhow. The interleavers can be implemented for parallel processing without occurring memory access conflicts. In the case of LTE turbo code a parallelism of the decoder architecture is considered with $P = 4$, $P = 8$, or $P = 16$ respectively.

References

1. Mller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., Wehn, N.: A novel LDPC decoder for DVB-S2 IP. In: Proc. DATE '09. Design, Automation. Test in Europe Conference. Exhibition, pp. 1308–1313 (2009)
2. Sun, J., Takeshita, O.Y.: Interleavers for turbo codes using permutation polynomials over integer rings. IEEE Trans. Inf. Theory **51**(1), 101–119 (2005). doi:10.1109/TIT.2004.839478
3. Third Generation Partnership Project: 3GPP TS 36.212 V8.5.0; 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8) (2008).www.3gpp.org
4. Nimbalker, A., Blankenship, Y., Classon, B., Blankenship, T.K.: ARP and QPP interleavers for LTE Turbo coding. In: Proceedings of the IEEE Wireless Communications and Networking Conference WCNC 2008, pp. 1032–1037 (2008). doi:10.1109/WCNC.2008.187
5. Third Generation Partnership Project: 3GPP TS 25.212 V1.0.0; 3rd Generation Partnership Project (3GPP);Technical Specification Group (TSG) Radio Access Network (RAN); Working Group 1 (WG1); Multiplexing and channel coding (FDD) (1999). www.3gpp.org

6. European Telecommunications Standards Institude (ETSI): Digital Video Broadcasting (DVB) Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; TM 2860r1 DVBS2-74r8. www.dvb.org
7. Tarable, A., Benedetto, S.: Mapping interleaving laws to parallel turbo decoder architectures. IEEE Commun. Lett. **8**(3), 162–164 (2004)

Chapter 5
Data Path of Individual Components

The data path of an architecture defines the alignment of processing elements. It realizes a certain functionality which fulfills a given throughput constraints. There exit always different possibilities to realize a defined functionality. Here we describe the individual steps to the design of individual components, again tailored for communications systems as shown in Fig. 4.1. Note, there is a difference of a data path for individual components and the data path for processors. The data path of a general purpose processor has the task to be as flexible as possible, thus arithmetics units are used with large bit width to provide a high flexibility, e.g. 32 bit or 64 bit. For dedicated hardware design the data path and thus the computational units are optimized with respect to the given algorithm to be processed. The steps to derive a dedicated data path are:

1. Starting from an algorithm description we have to decompose the algorithm in functional parts. Goal of this task is to identify the mandatory processing. The processing can often be separated in individual processing steps. For the individual processing parts we have to analyze:
   - the final bit width of the data, e.g. what is the influences of the data representation on the communication performance,
   - how to realize processing kernels, e.g. processing in time or frequency domain,
   - the correct or approximative functionality, e.g. whether the communications performance of the approximation is good enough.
2. The next step is to derive a data flow. Deriving a data flow utilizing the complete functionality of the algorithm can sometimes be cumbersome. We have to distinguish between important aspects for the data flow and unimportant ones. The data flow is independent from the processing itself, rather reflects:
   - data dependencies, e.g. which data have to processed first,
   - lifetime analysis of data, e.g. how long do we have to hold information,
   - concurrency, e.g. the choice of consuming one value after the other or processing all at once.

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_5, © Springer Science+Business Media New York 2014

The data flow analysis is very important for the hardware realization since it directly influences the resulting throughput and the overall data handling. The data flow analysis should be done on an simplified or abstracted functional model. An example is shown in next section.

3. Define the constraints to derive a data path. For dedicated hardware design the major constraint is often a time budget to fulfill a certain task. One constraints could be the number of data which should be processed within a certain time budget which is in fact the throughput definition of an application.
4. Define a feasible data path of one possible realization which fulfills the timing budget/throughput constraint. The timing information can be an abstract timing or a detailed cycle based timing. Important is that we allocate the time budget for the processing task.

In the following we derive first an abstract data flow followed by different data path possibilities. We use the example of a Max-Log-MAP processing which was introduced in Sect. 3.3.3. The corresponding functional processing parts are derived afterwards. The data paths derived in the next section are valid for all forward-backward based algorithms, thus independent of a possible Log-MAP or Max-Log-MAP implementation. This is only possible if a clear separation of data flow and functionality is derived. Realizing corresponding processing elements are presented Sect. 5.2. More complicated functions can either be done by look-up tables or by deriving approximations which can be efficiently implemented. Both techniques are explained based on exemplary functions which are often used to realize channel decoders.

5.1 Data Flow to Data Path Example

The MAP decoder has to calculate the maximum a posteriori probability which was introduced in Sect. 3.3.3. We deal here only with the so called Max-Log-MAP algorithm which is typically employed in the hardware realizations. To perform the Max-Log-MAP algorithm on the trellis we have to perform the so called forward-backward processing, this is shown step-by-step at Figs. 3.10–3.14.

To separate the processing and the data flow we first derive the mandatory processing steps in a more general way. The Max-Log-MAP decoding according to Sect. 3.4.2 can be summarized in four processing steps which are repeated here.

- Branch metric computation and allocation on the edges of the trellis.
- Forward recursion: calculate at each time step and thus trellis step k the corresponding path metric α_{k+1}^{S} for every state S. The calculation is done recursively, where the result in step k always depends on the previous results.
- Backward recursion: the backward recursion is based on exactly the same recursive processing, however starting with the calculation from the last state in the trellis.

- Soft-output calculation: to calculate the symbol-by-symbol MAP probability we need the results from the forward recursion, the backward recursion and the branch metrics.

The recursive calculations are always done on multiple values. In the case of trellis processing we calculate at each recursion step $\boldsymbol{S}_{k+1} = f(\boldsymbol{S}_k, \boldsymbol{\gamma}_k)$. In the case of a 4-state trellis $\boldsymbol{S}_{k+1}$ consist of 4 values, while $\boldsymbol{\gamma}_k$, the edge labels, consists as well of 4 branch metrics.

For the data flow we do not concern about the cardinality of $\boldsymbol{S}$ or $\boldsymbol{\gamma}_k$ and we can abstract the recursion function by a simple function which calculates only one value,

$$S_{k+1} = f(S_k, \gamma_k). \tag{5.1}$$

Each branch metric at step k is assumed as well to be one value. For deriving the data flow we use a block of 8 input values, denoted as $\boldsymbol{\gamma} = [a, b, c, d, e, f, g, h]$, with e.g. $\gamma_0 = a$ at step $k = 0$. Furthermore, for the function of Eq. 5.1 we utilize a simple minimum search of two input values, i.e. $\min(x, y)$. The entire forward-backward processing is thus reduced to a simple processing of 8 input values with simple computational stages.

The abstract function for deriving the data flow can be written as a marginalization problem:

$$M(\boldsymbol{x}) = \min_{\sim x_i}\{a, b, c, d, e, f, g, h\} \tag{5.2}$$

$\sim x_i$ defines the values of $\boldsymbol{x}$ without the value at position i. Thus, we have to compute 8 different results, one for each input position.

$$\begin{aligned}
M(x_i = a) &= \min\{b, c, d, e, f, g, h\} & M(x_i = b) &= \min\{a, c, d, e, f, g, h\} \\
M(x_i = c) &= \min\{a, b, d, e, f, g, h\} & M(x_i = d) &= \min\{a, b, c, e, f, g, h\} \\
M(x_i = e) &= \min\{a, b, c, d, f, g, h\} & M(x_i = f) &= \min\{a, b, c, d, e, g, h\} \\
M(x_i = g) &= \min\{a, b, c, d, e, f, h\} & M(x_i = h) &= \min\{a, b, c, d, e, f, g\}
\end{aligned}$$

One possible data flow structure to solve Eq. 5.2 is a tree structure. The tree structure is a typical data flow with respect to a parallel realization.

Parallel means here that we may process all input values $\{a, b, c, d, e, f, g, h\}$ in one step.[1] The tree structure for two functions can be seen in Fig. 5.1. The left tree calculates $M(x_i = a)$ and the right tree $M(x_i = b)$ respectively. It can be seen that in both trees identical sub-results are calculated, indicated by the circles. This gives us an indication about possible hardware reuse in a derived data path.

However, first we define additional constraints on our data flow with respect to data dependencies. The MAP processing is recursive and non commutative, at least when utilizing the trellis structure of convolutional codes. We have to ensure a recursive approach as well to solve Eq. 5.2. This is shown in the new data flow structure of

[1] In a hardware realization this would mean in the same clock cycle.

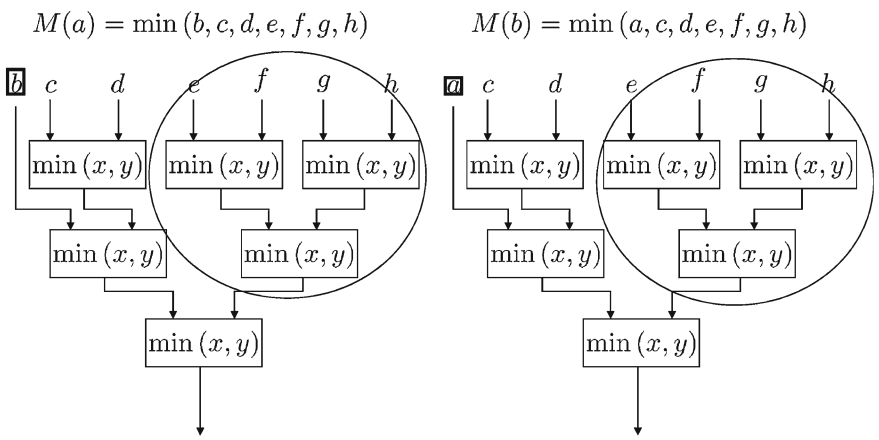


Fig. 5.1 Tree data flow for calculating $M(x_i = a)$ and $M(x_i = b)$

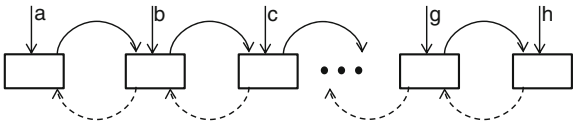


Fig. 5.2 Recursive data flow to calculate $M(x_i = a)$ and $M(x_i = b)$

Fig. 5.2. Two different data flow directions are shown here, the forward direction with solid lines, and the backward direction with dotted lines.

In the following we derive two different data paths which fulfill the constraint of a recursive calculation. The data path contains already timing information and is one major step towards a dedicated hardware architecture. Then, one processing step in the calculation corresponds to one clock cycle.

5.1.1 Serial Data Path: One min *(x, y) Unit*

The serial data path (serial marginalization) has already prerequisites for a final hardware realization. Here in the example, we assume that we can process one input value per clock cycle (processing step). Figures 5.3–5.5 show the data path and the processing steps to solve Eq. 5.2. We write all input information (a, b, d, e, f, h, g) to an array, indicated as box, which may be later instantiated as a memory. The current reading position of the array is indicated as a pointer. In the hardware implementation this pointer will be a reading address for a RAM, see Sect. 4.2. Each processing step is now described step by step.

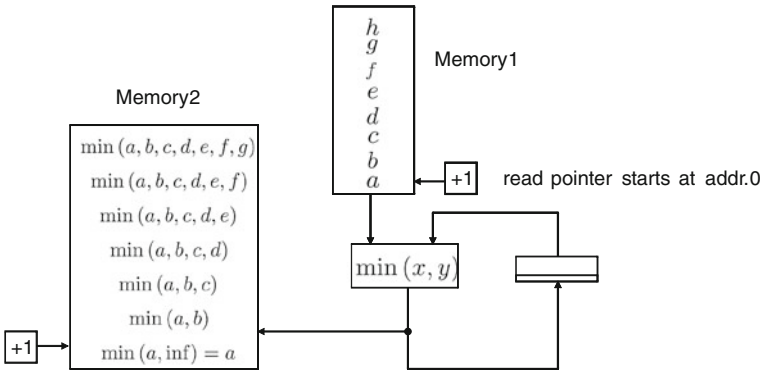


Fig. 5.3 Serial architecture: forward recursion, with the content of memory 2 after the 7th clock cycle

Step 1: Forward Processing (Figure 5.3)

- Initialization:
  The input values $\{a, b, c, d, e, f, g, h\}$ are stored sequentially in memory 1.
  The read pointer (address) for memory 1 is set to 0 (the address where value a is stored).
  The write pointer(address) for memory 2 is set to 0.
  The register is initialized with an 'infinite value', e.g. in hardware this would be the largest number with respect the utilized number representation.
- First clock cycle:
  Read the first value (a) from memory 1.
  $\min(x, y)$ calculates the first result which is $\min(a, \inf) = a$.
  The result is stored in memory 2.
  The register keeps now the value a.
  The read and write pointer are incremented by one.
- Second clock cycle:
  Read the second value (b) from memory 1.
  $\min(x, y)$ calculates the second result which is $\min(a, b)$.
  The result is stored in memory 2.
  The register keeps now the value $\min(a, b)$.
  The read and write pointer are incremented by one.
- 7th clock cycle
  After 7 clock cycles all intermediate results are stored in memory 2.
  The so called **forward processing** or **forward recursion** is finished.

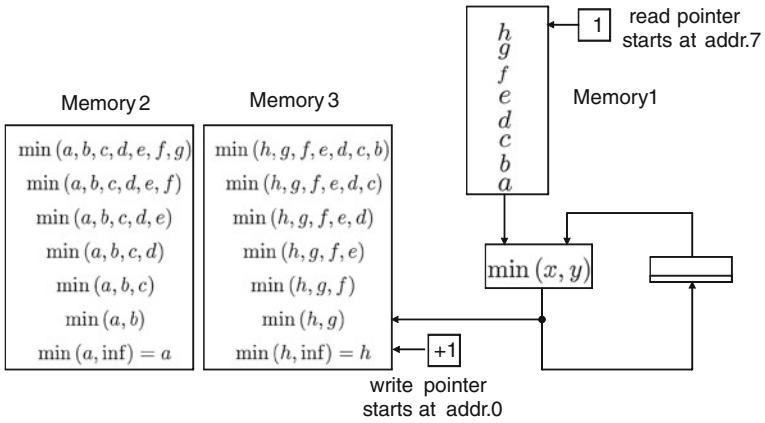


Fig. 5.4 Serial architecture: backward recursion, with the content of memory 3 after the 7th clock cycle

Step 2: Backward Processing (Figure 5.4)

For the backward processing a third memory has to be instantiated, denoted as memory 3.

- Initialization:
  The input values $\{a, b, c, d, e, f, g, h\}$ are again stored sequentially in memory 1.
  The read pointer (address) for memory 1 is set to 7 (the address where value h is stored).
  The write pointer(address) for memory 3 is set to 0.
  The register is initialized with an infinite value.
- First clock cycle to last clock cycle:
  Read the memory 1 starting form address 7, first value is h, last is a.
  The $\min(x, y)$ calculates the results in reversed order.
  The register keeps always the intermediate values.
  The read pointer of memory 1 is decremented by one in each step. The write pointer of memory 3 is incremented by one in each step.

Step 3: Output Processing (Figure 5.5)

For the output processing the data passed to the $min(x, y)$ are multiplexed from memory 2 and memory 3 respectively. The two new introduced multiplexers indicate the required switching.

- Initialization:

  The read pointer (address) for memory 2 is set to 0.
  The read pointer(address) for memory 3 is set to 6.

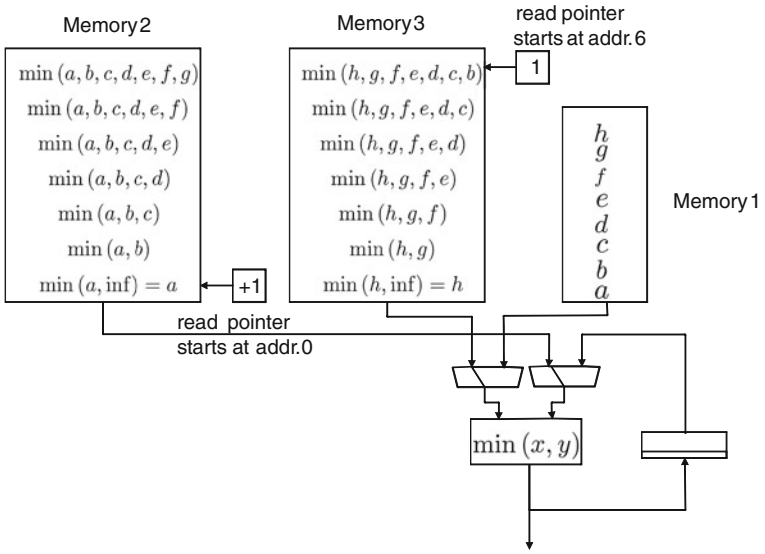


Fig. 5.5 Serial architecture: backward recursion

- First clock cycle:
  Only the value from memory 3 at address 6 is read which is already the first result $M(a) = \min(h, g, f, e, d, c, b)$.
  The pointer of memory 3 is decremented by one.

- Second clock cycle:
  Read address 0 from memory 2.
  Read address 5 from memory 3.
  $min(x, y)$ calculates the next result $M(b)$. The pointer of memory 2 is incremented by one.
  The pointer of memory 3 is decremented by one.

- Last clock cycle:
  Only the value from memory 2 at address 6 is read, this is already $M(h) = \min(a, b, c, d, e, f, g)$.

Summary data path: one min (*x*,*y*) unit

For the processing we instantiate three memories and one processing unit. One memory stores the input values, memory2 and memory 3 store the intermediate results of the forward, and backward processing respectively. The overall number of clock cycles for the processing can be decomposed into three processing steps which are: 7 clock cycles for the forward processing, 7 clock cycles for the backward processing and 8 clock cycles for the output processing. Thus, the overall number

of cycles is 22 to calculate 8 output values. On average, one output value needs $\#cycles/value = 22/8 = 2.75$ cycles. The presented data path utilizes a so called resource sharing of one processing unit, while load of the processing unit is 100 %.

5.1.2 *Serial Data Path: Two* min(*x,y*) *Units*

The section shows a serial data path using one additional $\min(x, y)$ unit. The goal here is to trade off processing units versus memory usage. The basic difference to the first approach is the simultaneous processing of the backward and output processing. The processing is now performed in two processing steps which are again described step-by-step:

Step 1: Forward Processing Figure 5.3

The forward processing remains identical to the description of Fig. 5.3.

Step 2: Backward Processing and Output Processing (Figure 5.6)

- Initialization:
  The input values $\{a, b, c, d, e, f, g, h\}$ are stored sequentially in memory 1.
  The read pointer (address) for memory 1 is set to 7 (the address where value h is stored).
  The write pointer(address) for memory 2 is set to 6 (the address where the last forward result is stored).
  The register is initialized with an infinite value.
- First clock cycle:
  Read the value (h) from memory 1.
  The first min-search unit calculates the first result which is $\min(h, \inf) = h$.
  The result is stored in the register, the register keeps the value h.
  Read the last forward processing result from memory 2, which is min (a, b, c, d, e, f, g).
  The second min-search unit calculates the first output result $M(h)$.
  The read pointer of memory 1 and 2 is decremented by one position.
- Second clock cycle:
  Read the next value (g) from memory 1.
  The first min-search unit calculates the second result which is $\min(h, g)$.
  The result is stored in the register.
  The second min-search unit calculates the second output result $M(g)$.
  The read and write pointer is decremented by one.
- Last clock cycle:
  The last result $M(a)$ is stored in the register after the 7th clock cycle.

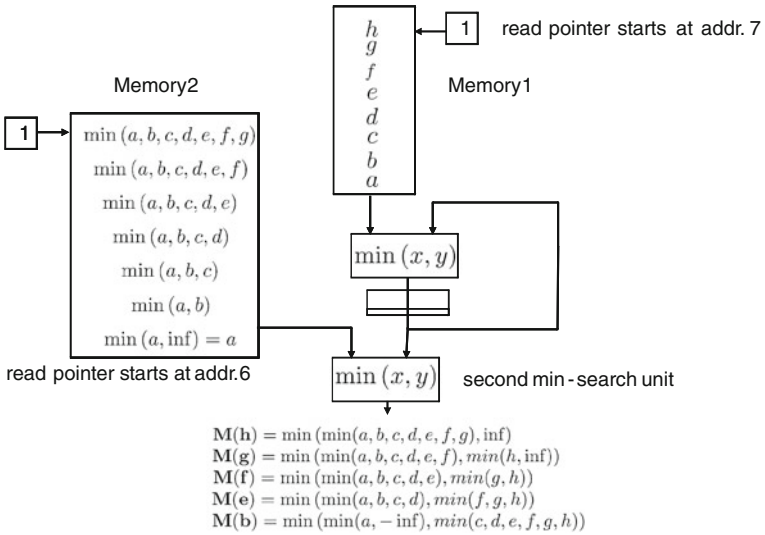


Fig. 5.6 Serial architecture: simultaneous backward and output processing

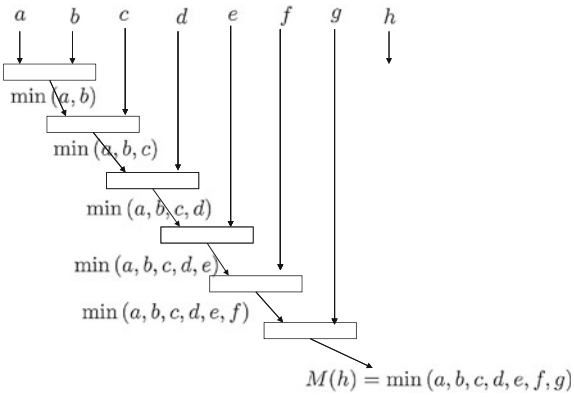
No additional reading from memory 2 is mandatory.
The backward processing including the output processing is finished.

Summary data path: two min (*x*,*y*) units

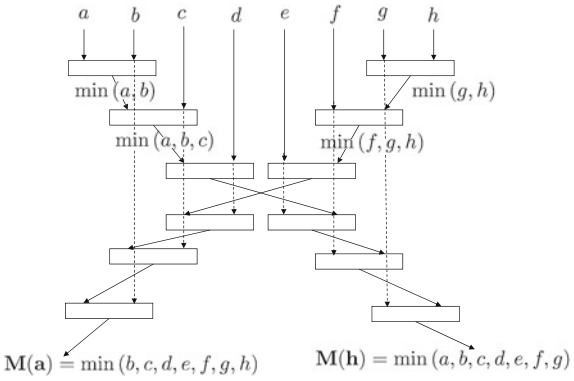
For the processing we instantiate two memories and two processing unit. One memory to keep the input values, and one memory to store the intermediate result of the forward processing. The overall number of clock cycles for the processing can be decomposed into two processing steps which are: 7 clock cycles for the forward processing, and 8 clock cycles for the backward recursion and output processing. Thus, the overall number of cycles is 15 to process 8 output values. On average, one output value needs $\#cycles/value = 15/8 = 1.875$ cycles. The presented serial data path puts focus on a simultaneous processing of different tasks. Two important difference can be seen when comparing the architecture with one and two $\min(x, y)$ unit. First, the result is obtained in a different order, second, the number of required clock cycles is much lower for the architecture according to Fig. 5.6.

5.1.3 Data Path: Parallel Processing

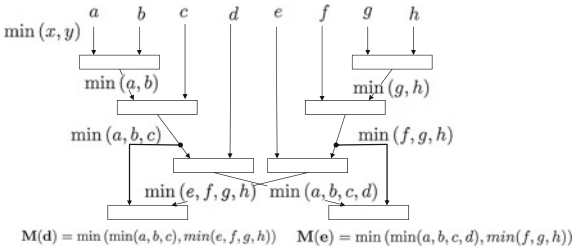
Parallel processing means, that more than one input value is processed in every clock cycle. Since we still have to solve Eq. 5.2 in a recursive manner this is not trivial. The architecture for a possible parallel processing is derived step-by-step.



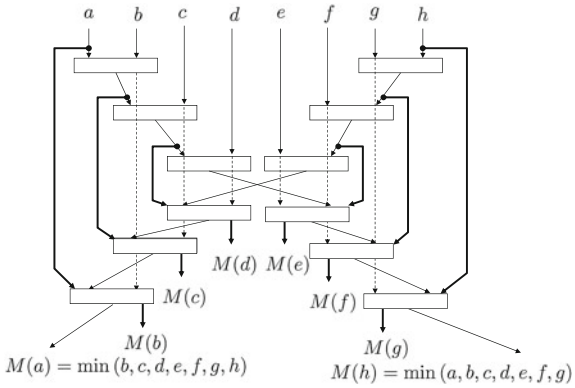
The first step is the so called unrolling of the calculation of $M(h)$. Here, we instantiate six functional units ($\min(x, y)$), which are shown as boxes in this figure Each unit receives the result of the previous unit and one of the input values as input. The intermediate result are written next the outputs of each unit.



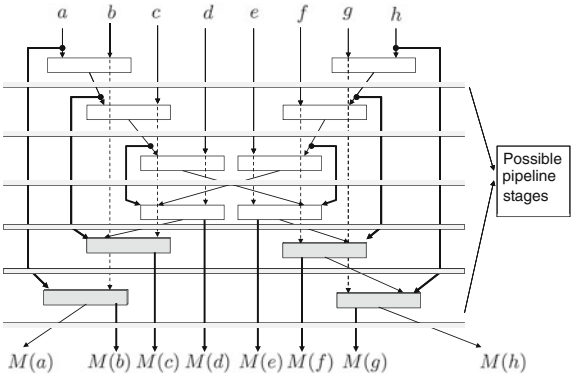
This figure shows the unrolling of the recursions from the beginning and the end. Two results are obtained $M(a)$ and $M(h)$. Every input value is consumed by two $\min(x, y)$ functions. The dotted arrows indicate that a certain value is also an input to a second functional units. For better readability some of the intermediate values are not labeled in this figure.



Here, we use two intermediate results to calculate other $M(x_i)$ values. For example $M(d)$ is obtained by an additional $\min(x, y)$ unit, the inputs x and y are the intermediate result $x = \min(a, b, c)$ which is obtained from the forward recursion, and $y = \min(e, f, g, h)$ which is calculated by the backward recursion respectively.



This figure shows the full parallel processing of all results. Every intermediate result, as well as every input result is used two times. Here, the data dependency restricts the order in which the operations can be performed. Note also that every functional unit in gray features tow instances of $\min(x, y)$ units. One for the output processing and one to pass an intermediate result to the next recursion stage. It is worth noting the similarities to the serial output processing of Fig. 5.6.



Typically we can not process all functions in one clock cycle. Hence, additional registers have to be instantiated which are called pipeline stages. At each pipeline stage we have to store all values which are consumed at a later time step. When ever a line of the derived data flow is crossing the horizontal pipe line stages the corresponding values have to be stored. It can be seen that the number of values we have to store varies for each pipe line stages.

Summary data path: parallel processing

In the following we assume that pipeline stages are instantiated after each processing block.

- The parallel architecture has a latency of 6 clock cycles, before the first output is valid, then in each clock cycle an **entire** block is calculated.
- We need 6 forward and 6 backward recursion units ($\min(x, y)$ units) and 6 output units.
- $(N-1) \cdot (N-1)/2$ intermediate results have to be stored, with N=8.
- In every clock cycle we have to read all 8 input values
- After the pipeline is filled we need one clock cycle to calculate all 8 results, the required cycles to process one value is thus $\#cycles/value = 1/8 = 0.125$ cycles.

The question whether to use a serial or parallel architecture leads to the question of required throughput, data storage problems and data access problems. The requirements for providing the correct input data at the right time is completely different for both architectures. Furthermore one has to analyze the complexity when utilizing the full functional unit of our application. Thus we have now to replace the $(\min(x, y))$ by the correct processing units, which leads to the so called recursion units.

5.2 Deriving Processing Units

In the previous section we have derived the data path for serial and parallel processing. As an example the data flow of a general forward-backward algorithm was used. As mentioned the data path defines the alignment of processing units. These procession

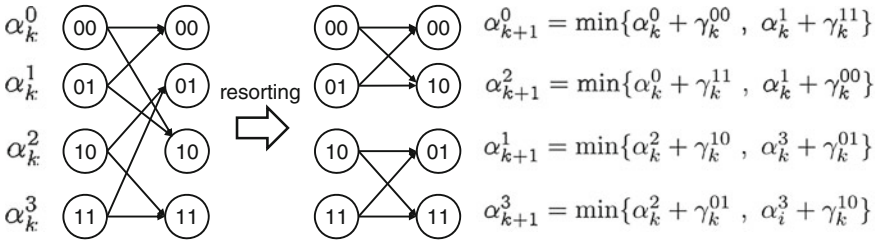


Fig. 5.7 Calculations of one trellis step.

units have to be designed with respect to the required functionality of the application. The functional units for the Max-Log-MAP algorithm are derived in Sect. 5.2.1. Often we have to perform algorithmic manipulations to realize the correct functionality. Either we can use look-up table to realize the mandatory functionality or we can derive good approximations by algorithmic transformation. Both techniques are shown by examples in Sects. 5.2.2 and 5.2.3, respectively.

5.2.1 Recursion Units

In this section we derive the processing units for the Max-Log-MAP algorithm. It is quite obvious that the forward recursion and the backward recursion share identical functionality, which can be seen by comparing the two mandatory equations for α and β processing (Eq. 5.3).

$$\alpha_{k+1}^{m'} = \min_{\forall(m->m')} \left(\alpha_k^m + \gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'})\right)$$
$$\beta_k^{m'} = \min_{\forall(m'->m)} \left(\beta_{k+1}^{m'} + \gamma_k^{x_i,x_i+1}(S_k^m, S_{k+1}^{m'})\right) \tag{5.3}$$

These equations where derived in Sect. 3.4.2. An unit implementing the functionality of this recursion step is called recursion unit (RU). It is always a function of the old state metrics and the input branch metrics. Figure 5.7 shows one example trellis with 4 states, while the trellis on the left reflects one trellis step. Starting form this example we derive a corresponding recursion unit. In trellis step k each state has two possible state transitions to a successor state in trellis step $k+1$. The trellis transition can be mapped to so called butterflies, however, the output has to be resorted. A butterfly represents the processing of two input to two output states, the direction (forward or backward) itself is not of importance. The advantage of the decomposition in butterflies and resorting is the clear separation of a regular processing part and a resorting of the state results.

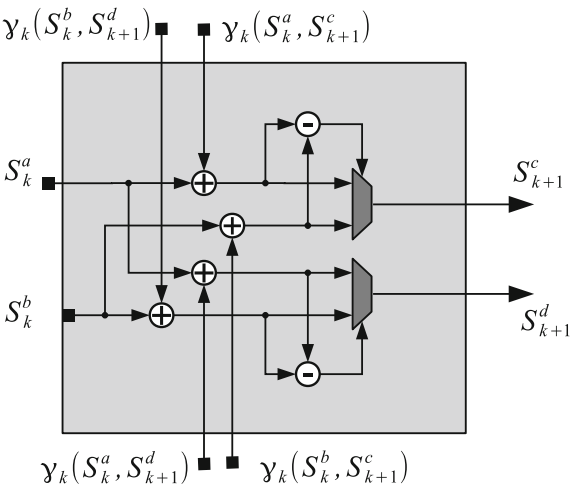


Fig. 5.8 Butterfly architecture for Viterbi decoding or Max-Log-MAP decoding

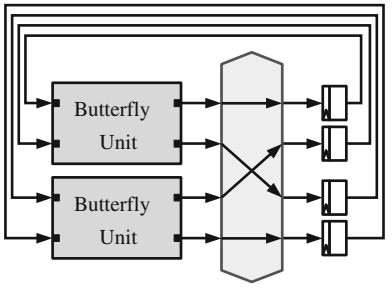


Fig. 5.9 Full recursion unit to process a 4-state trellis. A realization to process a larger amount of states can be done by an appropriate number of instantiated butterfly units.

A butterfly unit for Max-Log-MAP processing or Viterbi decoding is shown in Fig. 5.8. Every Butterfly unit is composed of two $\min(x, y)$ units simply realized as compare select units (CS). The CS part is realized by one subtraction, while the sign of the result is used as the multiplexer control signal.

The inputs for the butterfly units are the old states $S_k^{a,b}$ and the corresponding branch metrics which connect the corresponding states, e.g. $\gamma_k(S_k^a, S_{k+1}^c)$ represents the path metric between state S_k^a and S_{k+1}^c. Since one butterfly unit processes four state transitions we have four branch metrics as input. Note that for a convolutional code of $R = 1/2$ typically only two distinct branch metrics exist, then $\gamma_k(S_k^a, S_{k+1}^c) = \gamma_k(S_k^b, S_{k+1}^d)$ and $\gamma_k(S_k^b, S_{k+1}^c) = \gamma_k(S_k^a, S_{k+1}^d)$ respectively.

The recursion unit itself is implemented as multiple instances of the butterfly units and shuffling unit. The RU unit presented here can process one trellis step in one clock cycle. This is shown for a 4-state code in Fig. 5.9. The old state metrics are read from the registers and the new updated states are written to the registers. This can be done within one clock cycle. The shuffling unit has to ensure that the original trellis connectivity structure is preserved. The derived structure with four states can be extended for more states which is done by increasing the number of butterfly

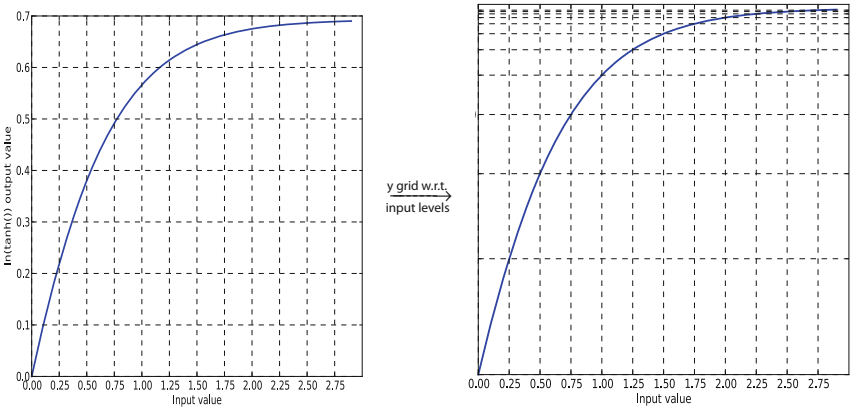


Fig. 5.10 Lookup table analysis and generation. Left shows the plot with a linear grid of the y-axis, right shows the y-axis with a grid with respect to the quantized input values

instances. The merge of the derived functional unit with one derived data path will result in a detailed Max-Log-MAP architecture. One specific instance of a so called serial MAP architecture will be shown in Sect. 6.3.1 in the context of building turbo code decoders.

5.2.2 Look-Up Tables

Sometimes it is difficult to realize a function in hardware. If the input bit width or the output bitwidth is small we can realize the corresponding function by a look-up table. A look-up table stores pre-computed values for each possible input combination. Deriving such a look-up table is shown Fig. 5.10. As an example we derive the function

$$f(x) = ln(tanh(x)). \tag{5.4}$$

This equation is used for processing so called check nodes, as introduced in Chap. 7, here it is used for demonstration purposes only. The left plot shows the function of $ln(tanh(x))$ with the input value on the x-axis, the function value on the y-axis. For the look-up table we first have to evaluate the input quantization. Here, the right plot features an input granularity of 0.25. That means the input values have two bits for the fractional part. Now we have to derive the corresponding output value. Due to the non-linear function the output value requires a higher resolution. For large input values we can barely distinguish the output results. For the look-up table we have to decide on how many bits to represent the corresponding y values. Communications performance simulations are mandatory to see if the chosen granularity will result in a performance degradation.

Look-up tables are simple to realize in hardware, however, the size of the values to be stored may quickly come infeasible for a large input width. Then, more elaborated techniques like an additional compression or the direct approximation of the function may become mandatory.

5.2.3 Deriving an Approximation Function

Approximative functions are used when a direct implementation will not fulfill the throughput constraints, results in a large area, or shows a high power consumption, respectively. For example realizing functions with many input variables quickly increases the number of stored values when realizing these by look-up tables. Thus, we should use algorithmic transformations to see whether it is possible to find a more suitable form for implementation. Often we can only approximate the desired function due to given constraints, however, for many applications this will be good enough. There exist no general rule how to derive a suitable approximation for an efficient hardware realization. Rather, the designer often relies on many different mathematical techniques and its own experience.

In the following one example is presented to derive an approximation suitable for hardware realization. The symbol-by-symbol MAP processing of a single parity check code is used for demonstration. The equation for the mandatory processing was derived in Sect. 3.3.3. Assuming a single parity check code with three bits we have to process:

$$\lambda_q = \ln\left(\frac{e^{\lambda_p} e^{\lambda_r} + 1}{e^{\lambda_p} + e^{\lambda_r}}\right) \tag{5.5}$$

This equation can be derived from Eq. 3.30.[2] It shows the output calculation for bit message q utilizing the two input messages p and r. In the first transformation step we utilize the already introduced Jacobian logarithm in its positive form

$$\ln\left(e^{\delta_1} + e^{\delta_2}\right) = \max^*(\delta_1, \delta_2) = \max(\delta_1, \delta_2) + ln(1 + e^{-|\delta_1 - \delta_2|}).$$

Using the Jacobian logarithm in Eq. 5.5 results in:

$$\lambda_q = \ln\left(\frac{e^{\lambda_p} e^{\lambda_r} + 1}{e^{\lambda_p} + e^{\lambda_r}}\right) = \max^*(\lambda_p + \lambda_r, 0) - \max^*(\lambda_p, \lambda_r) \tag{5.6}$$

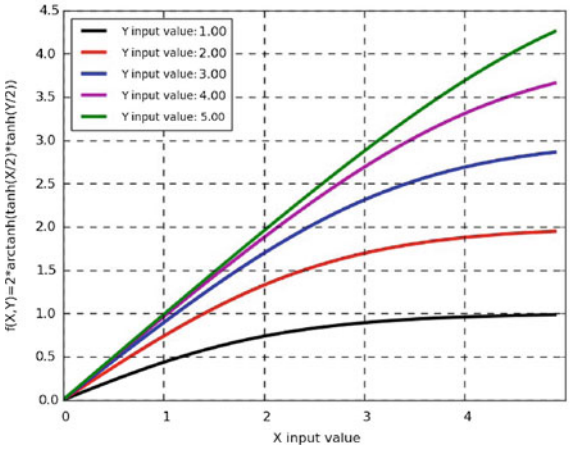
The result can be further decomposed in two maximum searches and two parts determining the correction term.

[2] The function can equivalent expressed as $\lambda_q = 2 \cdot \operatorname{arctanh}\left(\tanh\left(\frac{\lambda_p}{2}\right) \cdot \tanh\left(\frac{\lambda_r}{2}\right)\right)$. This trigonometric expression is sometimes used in literature.

$$\lambda_q = \underbrace{\max(\lambda_p + \lambda_r, 0) - \max(\lambda_p, \lambda_r)}_{approximation\ \tilde{\lambda}_q} + \underbrace{ln(1 + e^{-|\lambda_p + \lambda_r|}) - ln(1 + e^{-|\lambda_p - \lambda_r|})}_{correction\ term\ \delta}$$

This is an important result since we have clearly separated a simple approximation part and an additional correction part. Table 5.1 shows results for λ_q assuming different input values. The true result is decomposed in an approximative output $\tilde{\lambda}_q$ and a correction term δ . The approximation term is always the minimum of the absolute values, while the sign ensures that the single parity check condition is fulfilled. The largest correction term results when both input values are identical.

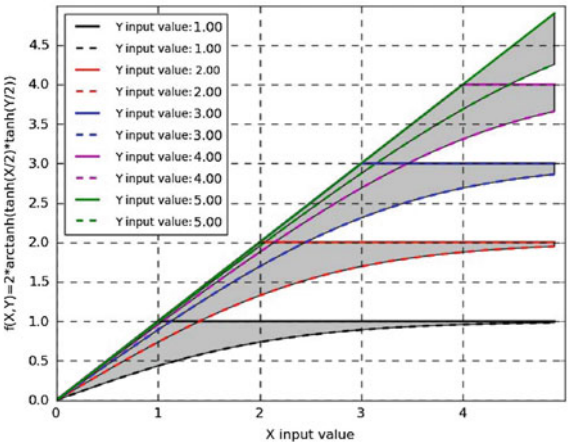
In the following the visual evaluation of Eq. 5.5 is shown, together with a possible realization of the correction term. The visualization often helps to get an idea about the quality of approximations. The used approximation derived within the following four figures was originally presented in [1].



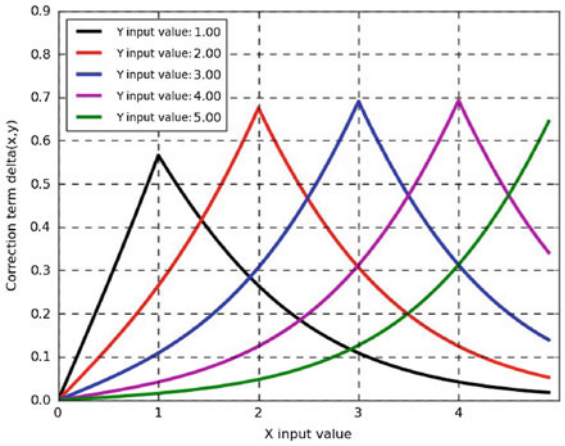
Plotted is Eq. 5.5, here denoted as $f(x, y) = 2 \cdot \text{arctanh}\left(\tanh\left(\frac{x}{2}\right) \cdot \tanh\left(\frac{y}{2}\right)\right)$. The x-scale represents the input value x, while 5 different y values result in 5 different graphs. The larger the y value, the larger the output result (y-axis). Thus, y = 1 will be always the graph at the bottom, y = 5 at the top of the graph. This is true for all following figures.

Table 5.1 λ_q calculations and approximation $\tilde{\lambda}_q$ for different input values

Input 1 λ_p	Input 2 λ_r	Approx. $\tilde{\lambda}_q$	Correction δ	Output λ_q
+5	+2	+2	−0.05	+1.95
+2	+5	+2	−0.05	+1.95
+5	−2	−2	+0.05	−1.95
−2	+5	−2	+0.05	−1.95
−5	−2	+2	−0.05	+1.95
−2	−2	+2	−0.68	+1.32
−5	−5	+5	−0.68	+4.32

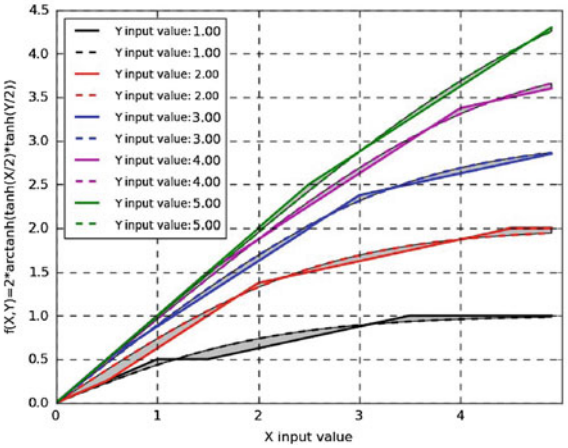


Plotted is the approximation $f(x, y) \sim \min\{|x|, |y|\}$ (dotted lines) and the original function. The shaded area highlights the difference between the true result and the approximation. It can be seen that the approximation error is always the largest for $x = y$.



The difference between the original function and the min approximation is shown. Thus, the correction function is plotted which is $\delta(x, y) = ln(1 + e^{-|x+y|}) - ln(1 + e^{-|x-y|})$ The left graph has $y = 1$ as its input value, while the right most plot has a constant

$y = 5$ value. A hardware approximation of this function can be derived by a Taylor approximation which is the result presented in the next step.



Here the original function and the final utilized approximation is shown. The hardware approximation term utilized here is: $\delta(x, y) \sim \psi(|x + y|) - \psi(|x - y|)$ With a hardware friendly ψ function.

$$
\begin{aligned}
&function\,\psi(a):\\
&\quad d = \frac{5}{8} - \frac{a}{4}\\
&\quad if\,(d > 0.0):\\
&\qquad return\,d\\
&\quad else:\\
&\qquad return\,0.0
\end{aligned}
$$

Comparing the final result one can see that the difference is rather small. This approximation can be applied in a recursive manner for the serial computation of single parity check codes with more bits. For example assuming three bits instead of two, the approximation can be calculated according to $\delta(x, \delta(y, z))$.

The solution derived in this example is only one possible approximation of Eq. 5.5. Providing the accurate functionality with respect to a given frequency constraint is the major challenge when deriving a possible realization. If the cycle budget can not be fulfilled, either we have to find a different approximation, or we have to increase the parallelism of the data path to allow for a reduced frequency constraint.

Final architectures featuring data path and processing units for turbo decoders and LDPC decoders are shown in Chaps. 6 and 7. For both type of decoders the respective design space is described highlighting different algorithmic and architectural possibilities.

Reference

1. Mansour, M.M., Shanbhag, N.R.: High-throughput LDPC decoders. IEEE Trans. Very Large Scale Integr. Syst. **11**(6), 976–996 (2003)

Chapter 6
Turbo Codes

A major step towards the Shannon limit was done in 1993 by introducing the so called turbo code (TC) [1]. Before 1993 a gap of 3 dB existed between practical coding schemes and that what the theory promised. With the introduction of turbo codes this gap was suddenly reduced to less than 1 dB. The new concept was the introduction of iterative decoding. Turbo codes consist of concatenated, simple component codes which calculate their information locally and exchange this information within an iterative loop. The local decoding of component codes also opened the door for practical systems. Due to the superior communications performance and the simple decoding scheme TCs have been rapidly adopted for many communications systems. This chapter introduces turbo codes and the corresponding encoders with respect to communication standards. The iterative decoding process of turbo codes is described and the communications performance is shown, taking into account the quantization effects of fixed-point calculations. The importance of an SNR insensitive algorithm is demonstrated which reflects a more realistic instantiation within a full system. The basic component of a turbo decoder is a maximum a posteriori (MAP) decoder. Its most common realization is shown which is linked to the previously derived data path of a forward-backward algorithm (Chap. 5). The chapter also summarizes the essential questions, which are most commonly used to drive industrial design process for turbo decoders and recap the huge possibilities within the architectural design space exploration.

6.1 Encoder Structure

Figure 6.1 shows a typical generic parallel turbo code encoder. This encoder features two component encoders in parallel which typically are both convolutional codes with identical polynomials.

For all 'parallel' turbo encoders the encoding process is similar. The systematic bit $\boldsymbol{u}$ is a part of the codeword $\boldsymbol{x}^s$. One parity part $\boldsymbol{x}^{p_0}$ is generated by component

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_6,

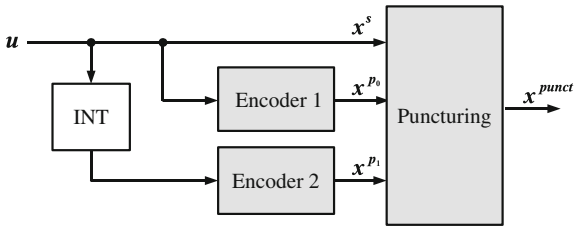


Fig. 6.1 Generic parallel turbo encoder structure with puncturing unit

encoder 1, while the second encoder generates the parity output $\boldsymbol{x}^{\boldsymbol{p_1}}$. The input stream of the second encoder is the interleaved version of the information bits $\boldsymbol{u} = \Pi(\boldsymbol{u})$. Π denotes the permutation operation as described in Sect. 4.3. The resulting code rate of this encoder structure is $R = 1/3$.

For adjusting the code rate the output of the encoder ($\boldsymbol{x} = [\boldsymbol{x}^{\boldsymbol{s}}, \boldsymbol{x}^{\boldsymbol{p1}}, \boldsymbol{x}^{\boldsymbol{p2}}]$) is passed to a puncturing unit. The puncturing unit erases bits from the original codeword $\boldsymbol{x}$ and passes a shorter vector $\boldsymbol{x}^{\boldsymbol{punct}}$ to the next stage. The puncturing unit is often called rate matching unit. Thus the transmitted code rate can be adjusted in a flexible way. In LTE a large range of code rates is specified from base code rate $R = 1/3$ up to very high code rates, e.g. $R = 14/15$. Note that in the high rate case of LTE even information bits are punctured out.

The presented parallel encoder structure is utilized in many communications standards. These are shown in Table 6.1 with their corresponding base parameters. The component codes used in these standards are convolutional codes either with 8 states or 16 states respectively.

The most important standard is the 3GPP initiative (UMTS, LTE). Its turbo code encoder structure is shown in Fig. 6.2. Here, the puncturing unit is omitted. The component encoder are 8-state recursive convolution codes with forward and backward polynomials of $G_0 = 13$ and $G_{FB} = 15$ respectively. The component encoders are the same for UMTS and LTE encoding. However, the interleaver is different for both standards. The newer LTE standard features an interleaver which allows a simpler implementation of parallel decoder architectures. The problem of parallel interleaving was already described in Sect. 4.3.

Table 6.1 Turbo codes in communication standards

Application	Turbo code	States	Forward	Backward	Base code rate
UMTS	Binary	8-state	13	15	1/3
LTE	Binary	8-state	13	15	1/3
CDMA 2000	Binary	8-state	13, 17	15	1/5, 1/3
IEEE 802.16	Duo-binary	8-state	11, 13	15	1/2
CCSDS (deep space)	Binary	16-state	33, 25, 37	23	1/6, 1/4, 1/3,1/2
Immarsat	Binary	16-state	35	23	1/2
DVB-RCS	Duo-binary	8-state	11, 13	15	1/3
Eutelsat	Duo-binary	8-state	13	15	4/5, 6/7

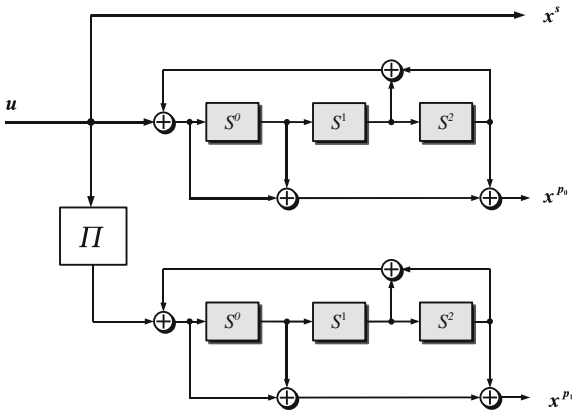


Fig. 6.2 3GPP turbo encoder with 8-state RSC component encoder

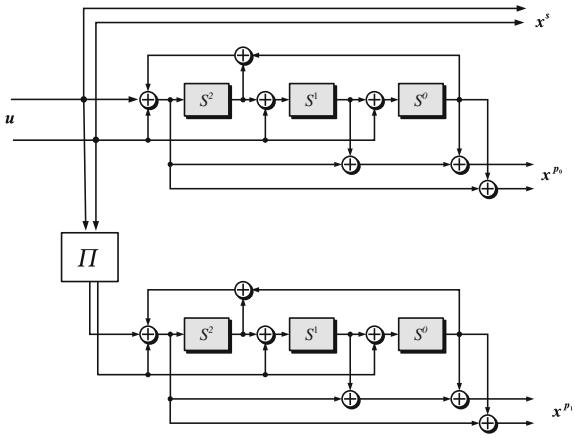


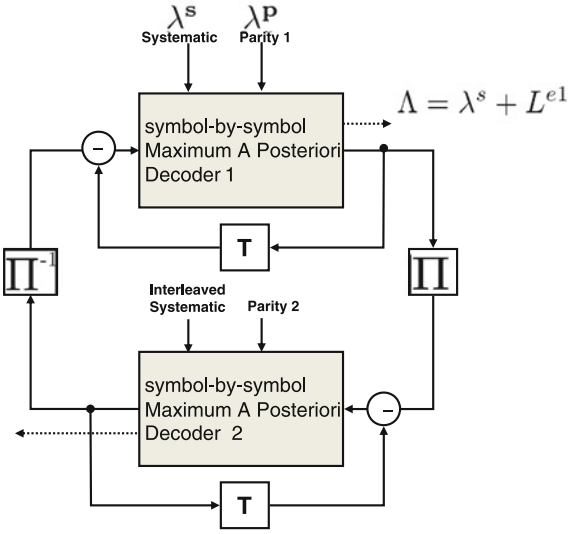
Fig. 6.3 WiMAX duo-binary encoder with 8-state RSC component encoder

Figure 6.3 shows the DVB-RCS duo-binary turbo encoder. The encoder always operates on two bits simultaneously, while these couples are treated together during encoding and decoding. The interleaver permutes the stream with respect to couple positions and does not break up the tight coupling of the 'duo' bit grouping. The resulting base code rate is $R = 1/3$.

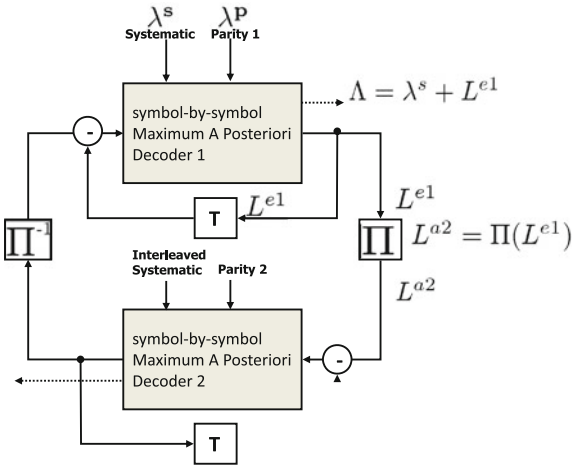
Table 6.1 shows 8 different standards featuring turbo codes. All have a parallel turbo code encoder structure with two component encoders. The turbo code decoding of all theses standards share the same basic iterative exchange of messages. This basic iterative decoding procedure is described next section. Turbo code decoder and turbo decoder are used as synonyms in the following.

6.2 The Iterative Decoding Procedure

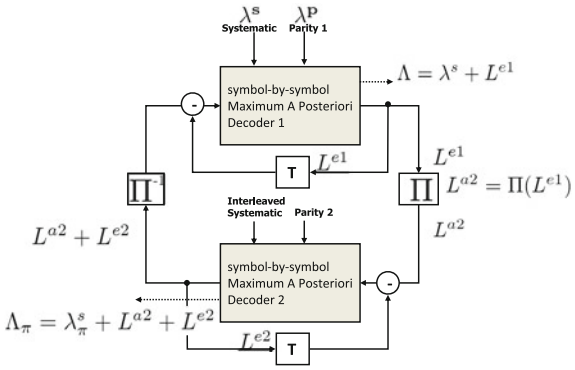
In this section the iterative decoding procedure is described step-by-step. The decoder structure presented here can decode any parallel concatenated turbo code which consist of two component codes. Thus, it fits to all encoders of Table 6.1. This section presents one possible decoder structure. Obviously, more decoder structures exist. All of them have to handle the so called extrinsic information principle which takes care how to exchange messages. One iteration is described in the following.



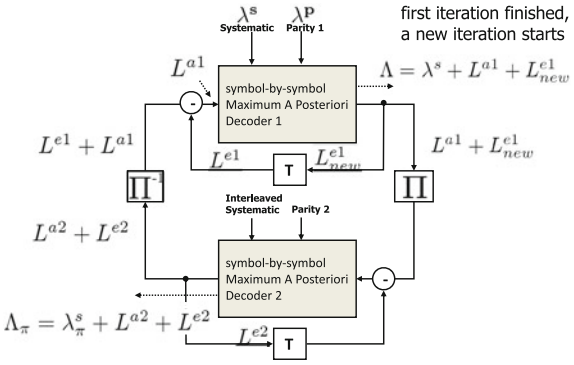
For each component encoder a corresponding component decoder exists. Decoder 1 will consume the received systematic information $\boldsymbol{\lambda}^s$ and the corresponding parity LLRs $\boldsymbol{\lambda}^{p1}$. Decoder one starts to calculate new a posteriori probability information for each systematic bit information. Each APP information can be decomposed in the already existing systematic part and an additional 'extrinsic' gain. Thus, we obtain $\boldsymbol{\Lambda} = \boldsymbol{\lambda}^s + \boldsymbol{L}^{e1}$.



We can isolate the new additional extrinsic information from the APP information by subtracting the systematic information (position by position). The resulting extrinsic vector $\boldsymbol{L}^{e1}$ is passed to the interleaver stage Π and stored in the indicated delay unit (T). The interleaver reorders the information according to the utilized encoding procedure. The new obtained sequence is now treated as new 'a priori' information for the second MAP decoder $\boldsymbol{L}^{a2}$.



Decoder 2 consumes the input a priori information, the interleaved systematic information $\boldsymbol{\lambda}_\pi^s$ and the corresponding parity LLRs $\boldsymbol{\lambda}^{p2}$. Decoder 2 calculates new APP information $\boldsymbol{\Lambda}_\pi$ for each (interleaved) systematic bit. This APP information can be composed as a sum of a priori information $\boldsymbol{\lambda}^{a2}$ obtained from decoder 1, the interleaved systematic information $\boldsymbol{\lambda}_\pi^s$ and an additional gain provided by decoder 2 $\boldsymbol{L}^{e2}$. Only the additional gain $\boldsymbol{L}^{e2}$ is stored in the delay unit. We pass the information $\boldsymbol{L}^{e2} + \boldsymbol{L}^{a2}$ to the deinterleaver stage.



The vector is deinterleaved again position by position. We obtain a sum, composed of the old extrinsic information $\boldsymbol{L}^{e1}$ originally calculated by decoder 1 and a new a priori information $\boldsymbol{L}^{a1}$ which was provided from decoder 2. However, before passing the sum information to decoder 1, we subtract the old extrinsic information ($\boldsymbol{L}^{e1}$). Only the additional information $\boldsymbol{L}^{a1}$ is passed to decoder 1. Otherwise we would rely on our old confidence. One iterations is finished and a new iteration starts by calculating a new APP information $\boldsymbol{\Lambda} = \boldsymbol{\lambda}^s + \boldsymbol{L}^{a1} + \boldsymbol{L}^{e1}_{new}$.

The described iterative procedure is continued several times. In practical applications we rarely iterate more than a maximum amount of 8 iterations. Note that different structure exist for this iterative processing, for example we can pass the information $\boldsymbol{\lambda}^s + \boldsymbol{L}^{e1}$ to the second component decoder. Then, we don't need to provide $\boldsymbol{\lambda}^s_\pi$ separately to the decoder since it is already considered in the passed sum. Still, we have to store this sum in the delay unit (T) to ensure the extrinsic information principle, as already indicated in Sect. 3.5.

6.2.1 Convergence Progress (EXIT Charts)

The convergence of the iterative decoding process can be analyzed by extrinsic information transfer (EXIT) chart analysis [2]. The information characteristics of the component codes are analyzed by tracking the information content of the output information and input information of corresponding component decoders. The soft-in soft-out decoders for iterative turbo decoding have two inputs. The first input comprises the channel values, which depend on the received LLRs, the second input retrieves a priori information which is exchanged during iterative decoding. We can track the information content for input variables and output variables by using the mutual information which was already introduced in Sect. 2.3. The mutual information $I(L^a, S)$ describes the information content of the a priori information L^a while $I(L^e, S)$ represents the mutual information of the extrinsic output values L^e. For binary variables and an antipodal modeling of the symbols ($s \in \{+1, -1\}$), we can calculate the information content of the a priori information $I(L^a, S)$ by:

$$I^a = I(L^a, S) \tag{6.1}$$
$$= \frac{1}{2} \sum_{s_i=\{+1,-1\}} \sum_{\forall a} p(a|s_i) \cdot log_2 \left(\frac{2p(a|s_i)}{p(a|s_i = -1) + p(a|s_i = +1)} \right)$$

The equation of the mutual information here was already used in Sect. 2.3 to derive the channel capacity. The mutual information can be obtained by summing over all possible a priori values expressed by the variable a. Note that here a probability mass function for the a priori values $p(a|s)$ is assumed which allows the modeling of a fixed-point realization. $p(a|s)$ can be tracked by Monte Carlo simulations. The equation for the information content of the extrinsic information is identical, however, with the density mass function for the extrinsic information to be $p(e|x_i)$.

Two assumptions are often assumed for deriving the EXIT charts. First, the interleaver is assumed to be sufficient large, such that the exchanged information can be modeled as independent information. Second, the extrinsic information is assumed to approach a Gaussian distribution with increasing iterations.

Figure 6.4 shows the setup to measure an EXIT chart. The input LLRs for the channel decoder can be described by

$$\lambda_y = \frac{2}{\sigma_n^2} y = \frac{2}{\sigma_n^2}(s + n), \tag{6.2}$$

with σ_n^2 the channel noise variance. Thus, the channel LLR input values are Gaussian distributed with the mean value $\mu_y = \frac{2}{\sigma_n^2}$, while its variance will be $\sigma_y^2 = \frac{4}{\sigma_n^2}$. A Gaussian distribution having the variance twice the mean is said to be consistent.

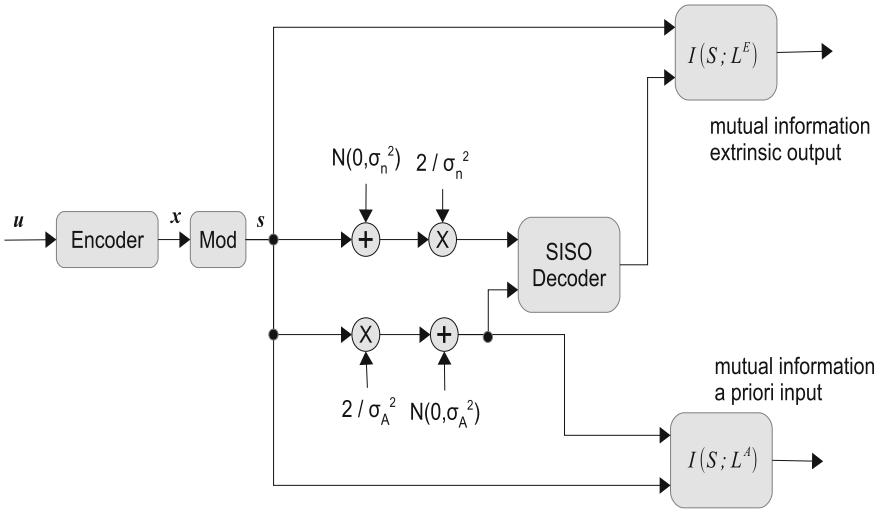


Fig. 6.4 Measurement of the EXIT chart

When the input is consistent it can be observed that the distribution of the extrinsic output is consistent too [2]. The extrinsic output of one component decoder serves as a priori information of the other component decoder. Thus, the a priori information can be modeled as Gaussian variable with a variance of σ_a^2 with a mean value of $\mu_a = \frac{\sigma_a^2}{2}$.

The mutual information I^a depends only on the variance σ_a, while the mutual information of the extrinsic information depends on the a priori input and the current signal to noise ratio. Thus, the extrinsic information characteristics are defined as a transfer function of:

$$I^e = T(I^a, E_b/N_0) \tag{6.3}$$

For each signal-to-noise ratio of the channel we can evaluate another characteristic. Fig. 6.5 shows the extrinsic information transfer characteristics for a convolutional code used for LTE turbo codes (8-states, R = 1/3, Log-MAP). Shown are the decoder behaviors for three different E_b/N_0 values with the x-axis representing the mutual information I^a at the input of the decoder while the y-axis shows the extrinsic information I^e respectively.

The convergence of a turbo decoder can now be visualized by plotting the characteristics of both component decoders into the same chart. However, the axis have to be swapped for the second soft-in soft-out decoder. The resulting EXIT chart is shown in Fig. 6.6 for three different E_b/N_0 values. In addition the so called trajectory

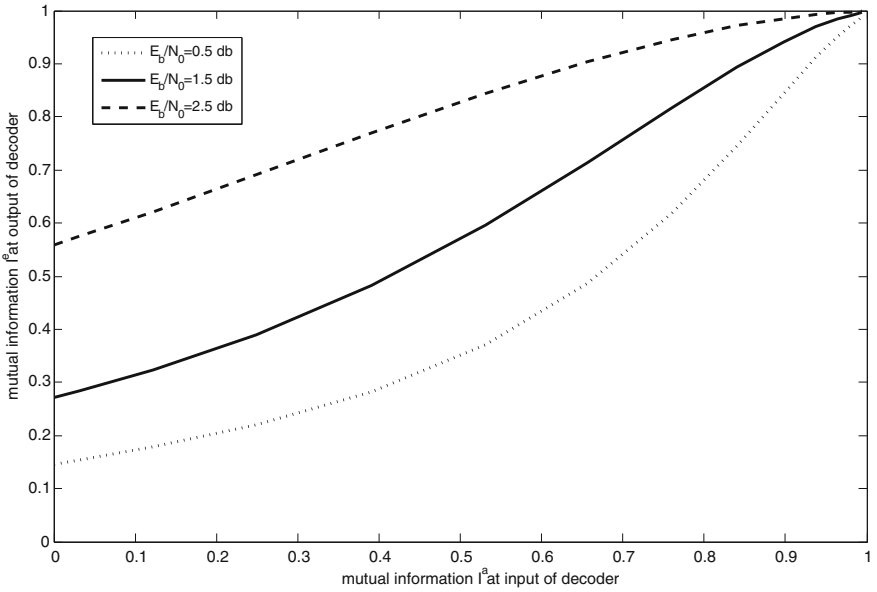


Fig. 6.5 Extrinsic information transfer characteristics of soft-in soft-out component convolutional decoder (8-state, R = 1/3, LTE)

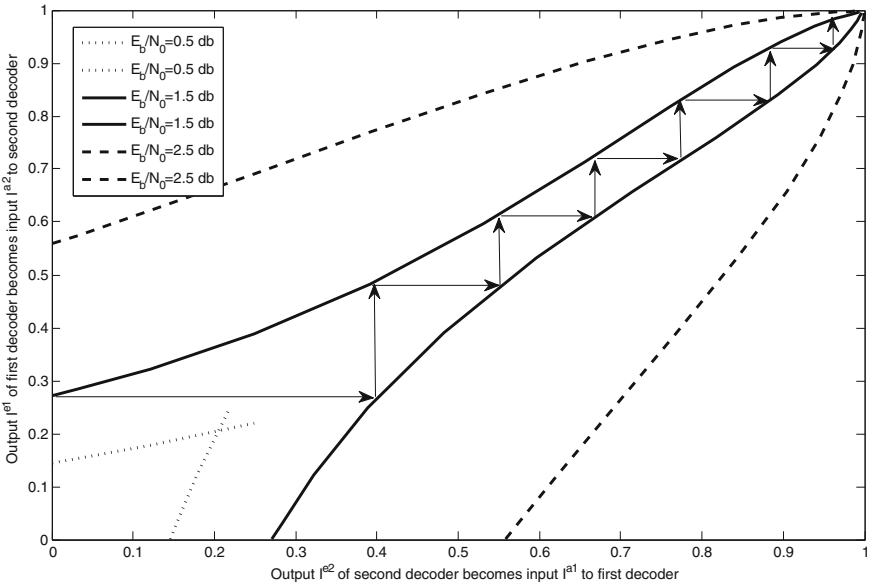


Fig. 6.6 EXIT chart characteristics for three different E_b/N_0 values. The sketched trajectory is only valid for an infinite interleaver size

of the decoding process is shown. There, the output information of one decoder is used as input information of the next decoder. The iteration proceeds as long as there is a gain of information. For $E_b/N_0 = 1.5\,\text{dB}$ the iterative decoding will continue until the maximum information of $I^e = I^a = 1$ is reached, i.e. error free information. For a larger E_b/N_0 the number of iterations is reduced to obtain an error free decoding, while at a low E_b/N_0 the two characteristics will intersect and thus will not gain any information for successive iterations. The decoding procedure is going to get stuck. For the hardware realization the EXIT charts can be used to track the effects of quantization or to visualize the information loss of sub-optimal algorithms. In summary, EXIT chart analysis is a mighty tool to explain, analyze, and as well to design iterative decoders, see [2, 3].

6.2.2 Communications Performance

As mentioned before the communications performance is typically measured in frame error rate (FER) over the signal-to-noise ratio (SNR). The decoding of turbo codes is an iterative process, while the communications performance improves for successive iterations.

Figure 6.7 shows various iterations of a LTE turbo decoder with $K = 6144$ information bits and a code rate of $R = 1/2$. An AWGN channel was utilized for these performance results. The input data has a FER of nearly one for the entire SNR range.

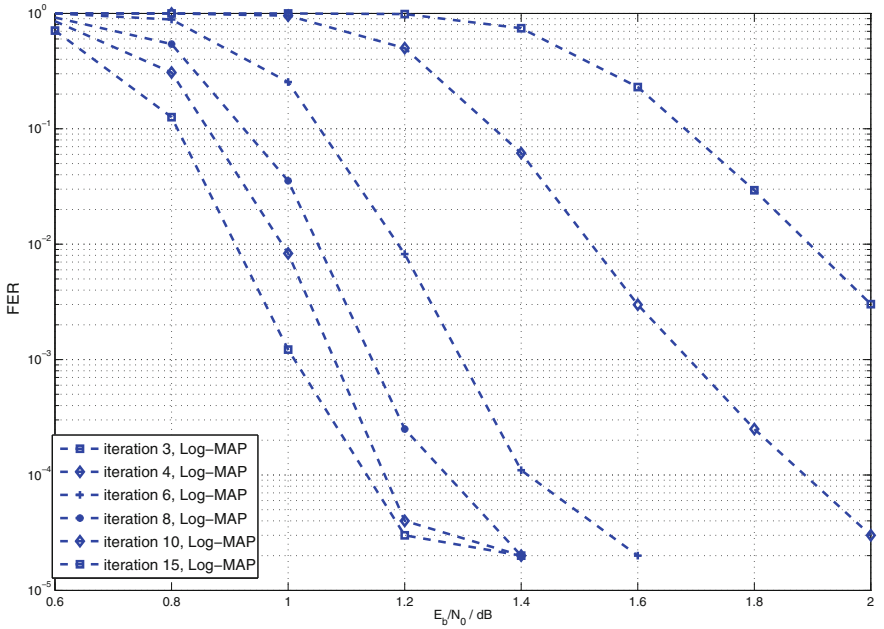


Fig. 6.7 LTE turbo code performance for code rate $R = 0.5$ and a block length of $K = 6144$ [4]

Thus, at least one bit error is occurring in each frame. After 3 iterations a huge coding gain can be observed. For the 4th and 6th iteration the gain is still significant. However, after 8 iteration the additional gain for further iteration gets smaller and smaller. Typically, not more than 8 turbo iterations are performed for decoders realized in hardware. This number is restricted due to latency and throughput constraints of the system. For the LTE standard an achieved $FER = 10^{-3}$ is sufficient, since additional techniques like automatic repeat requests (ARQ) are applied to preserve the desired system quality of service. Anyhow for turbo codes we have to distinguish between convergence gain and low asymptotic gain. Good convergence means that the FER already decreases at an *SNR* close to the theoretical limit. This SNR region with still improving communications performance is also denoted as waterfall region. The asymptotic gain describes the (SNR) coding gain at a very low bit error rate. As mentioned, the gain in communications performance is getting smaller with more iterations. Note that the convergence speed of the algorithms depends also on the block size. The larger the block size the more iterations are mandatory to obtain the best achievable communications performance.

Figure 6.8 shows the 4th and 8th iterations of a LTE turbo decoder ($K = 6144$ information bits, $R = 1/2$) simulated with a simple additive white Gaussian noise channel. Shown are three different algorithm, the optimal Log-MAP simulation, the Max-Log-MAP simulation, and an improved Max-Log-MAP simulation with extrinsic scaling factor (ESF). For eight iterations the performance difference between the

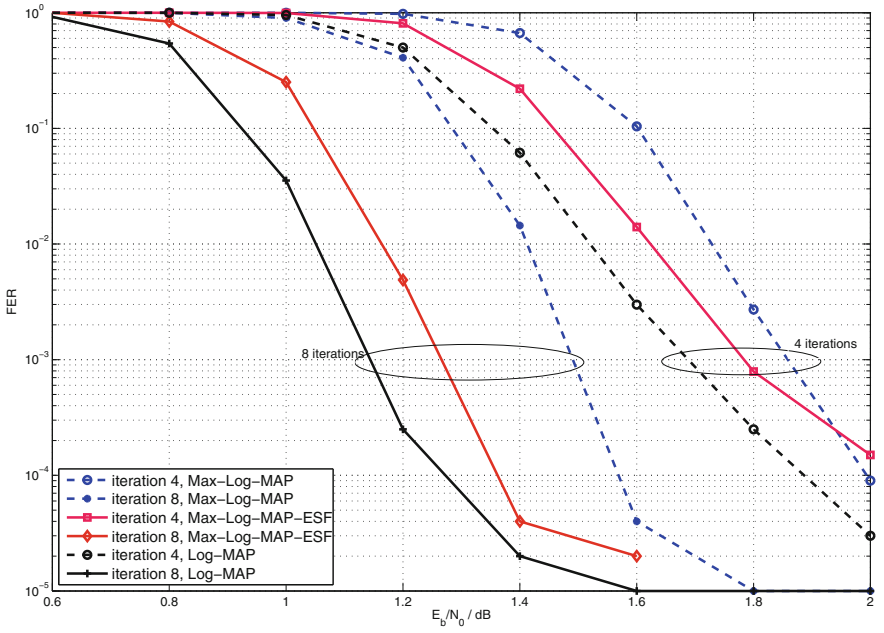


Fig. 6.8 Comparing Max-Log MAP, Log-MAP, Max-Log-MAP-ESF communications performance

three algorithms is already significant. The difference between the optimal Log-MAP implementation and the sub-optimal Max-Log MAP implementation is 0.3 db. It is well know that the Max-Log-MAP algorithm overestimates the additional gain which is passed between the component codes. In [5] a simple extrinsic scaling was introduced which counterbalances this overestimation. The extrinsic information is just multiplied by e.g. ESF $= 0.75$ and the performance improves as shown in Fig. 6.8.

The LTE standard uses new interleavers and a new puncturing scheme compared to its original UMTS definition. Figure 6.9 shows the difference between the LTE system and the HSPA system. Both with identical block length of $N = 5178$ and a high code rate of $R = 0.94$. Both codes utilize the same number of iterations and the same implementation of the component code processing. Noteworthy, a huge difference in communications performance can be seen. The reason for this is the different minimum distance of the resulting code (after puncturing). An appropriate interleaver and a clever puncturing scheme influence the distance spectrum and thus the asymptotic gain. Turbo codes are faced with the convergence versus d_{min} dilemma. The better the early convergence the smaller the minimum distance which results in a so called error floor.

The convergence of a code is defined by the used component codes while the interleaver determines the error floor. Much effort was taken to answer the question: which simple codes have to be concatenated to approach the Shannon limit? The EXIT chart analysis is one major techniques to answer this question.

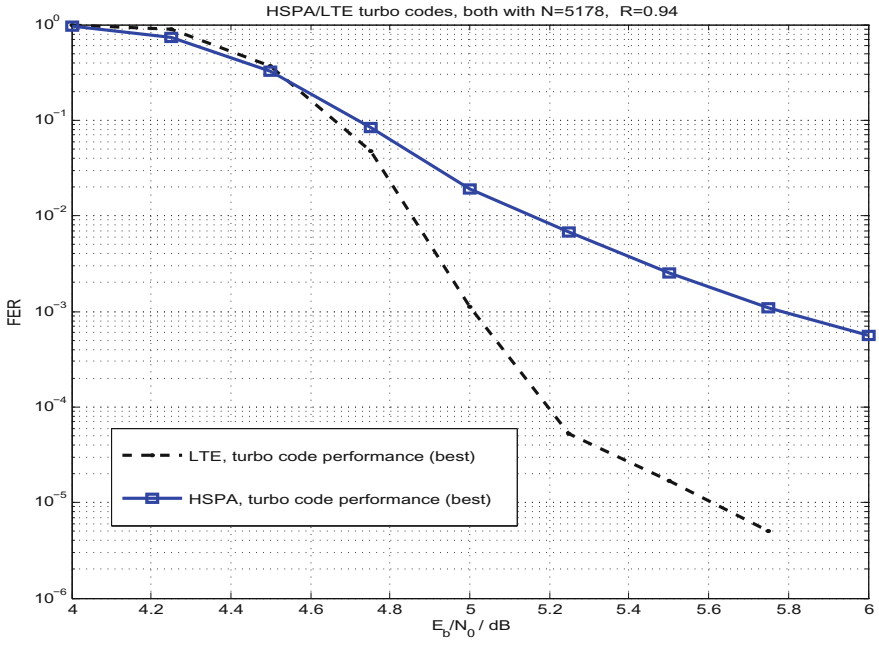


Fig. 6.9 HSPA and LTE turbo code performance for code rate $R = 0.94$ and a block length of $N = 5178$ [4]

6.2.3 Fixed-Point Realization and Robustness

For hardware realization of any channel decoder we have to convert the algorithm from a floating point to a fixed-point realization as shown in Fig. 4.1. Goal is to represent every variable in the algorithm with a limited number of bits, since a smaller bit width results in a smaller area and as well to a reduced power consumption. During the conversion an information loss occurs which may result in a degradation in communications performance. For simple components like the demodulator we can evaluate this information loss in an analytical way, however, for iterative decoding algorithm we have to simulate the resulting communications performance while comparing the result towards an optimal floating point implementation. Turbo decoders are realized in hardware since its standardization in UMTS, since then many explorations have been carried out, e.g. [6, 7].

For the fixed-point realization the quantization of the input data (Q_{in}) is of great importance as the bit width of all other variables can be derived from this number. For example in practical systems the quantization of the exchanged message between the component decoders is chosen to be the input bit width plus one bit, i.e. $Q_{ext} = Q_{in} + 1$. In the following we always simulate the internal bit width of the component decoder sufficiently large such that no degradation occurs w.r.t. communications performance. In the following only the discussion of the input bit width is highlighted.

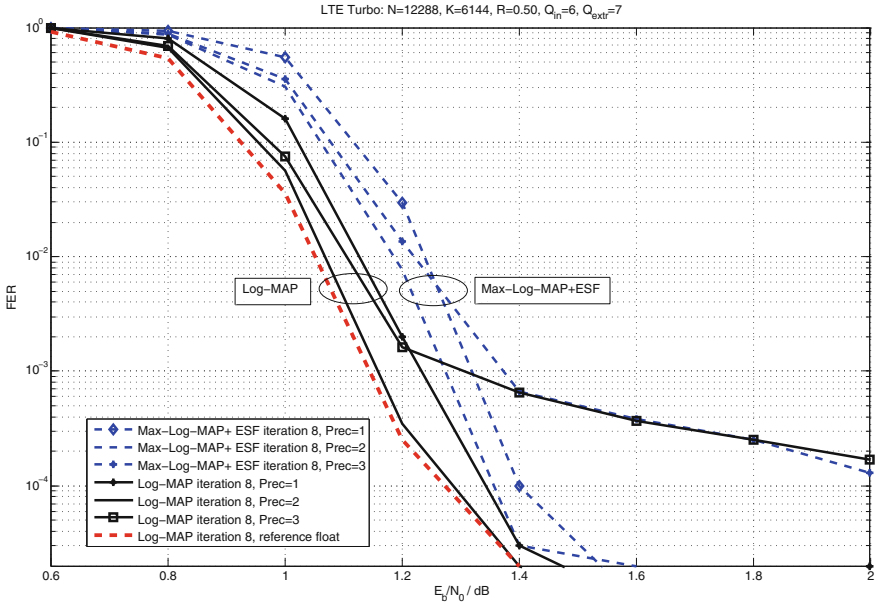


Fig. 6.10 LTE turbo code performance for code rate $R = 0.5$ and a block length of $K = 6144$ bits. All results with 6 bits input width, while the precision is varying

As derived in Sect. 2.2, the inputs to the turbo decoder are the channel LLR values $\lambda = \frac{2}{\sigma^2} y$ which are the received samples corrected by the channel reliability factor $L_{ch} = \frac{2}{\sigma^2}$.

Figure 6.10 shows the communications performance result of a LTE turbo code ($R = 0.5$, $K = 6144$). For all fixed-point simulations an input quantization of $Q_{in} = 6$ bits is assumed, while the number of fractional bits is varying between one bit, two bits and three bits, denoted as Prec $= 1$, Prec $= 2$, and Prec $= 3$, respectively. Always the result of the 8th iteration is shown for a fixed-point Log-MAP and a fixed-point Max-Log-MAP with scaling realization. One can see that the performance degradation of the Log-MAP fixed-point model leads only to a small loss compared to the reference floating point Log-MAP model. Furthermore, the relative behavior between both types of algorithms, Log-MAP and Max-Log-MAP with scaling, is similar. The best communications performance is achieved in both algorithmic cases when simulating with two fractional bits. All simulations in this graph are carried out with an optimal channel reliability factor L_{ch}. Thus, for each SNR point a different scaling of the input values is applied while the input quantization is performed after this optimal demodulator stage.

For an instantiation of a turbo decoder in a larger system this is a too optimistic assumption. In a final system realization we have to estimate σ^2 by a channel estimator. In practical systems we can not assume a perfect estimation and by that optimally scaled input values. Rather, we have to explore the fixed-point analysis with so-called

mismatched SNR estimations. A mismatched SNR estimation refers to the case of difficult channel conditions at which the channel reliability factor is fixed for an entire SNR range. Furthermore, we have to emulate the case that the noise level is estimated imprecisely.

Figure 6.11 shows the performance of the same LTE turbo code, while all simulations are performed with $Q_{in} = 6$ bits input values (two bits fractional part). This time we perform the simulations with different channel reliability values ranging from $L_{CH} = 1$ to $L_{CH} = 3$. For each simulation the corresponding channel reliability factor is fixed for the entire SNR range and denoted as CHR in the legend of the figure.

The Max-Log-MAP with scaling shows for all CHR cases a nearly identical communications performance. The reason for that is that the Max-Log-MAP algorithm is SNR insensitive since only the metric differences are of importance, see [8]. However, the Log-MAP performance results show a huge variation. For CHR $= 1$ the decoding algorithm does not show any convergence at all. Indeed, this is a large mismatch of the channel reliability factor, however, for a mobile device we have to ensure a robustness of the algorithm even under difficult conditions. The reason for the SNR sensitivity of the Log-MAP algorithm is the correction term introduced in Sect. 3.3.4.

In summary, a fixed-point exploration of an algorithm is more than fitting the bit width to one specific simulation set up. We have to explore the communications

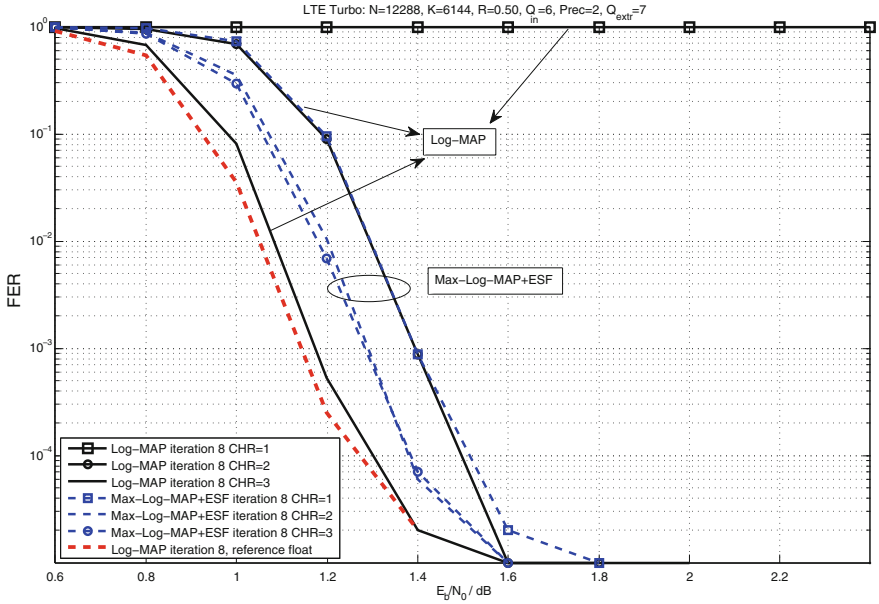


Fig. 6.11 LTE turbo code performance ($R = 0.5$, $K = 6144$). All results with 6 bits input width and 2 precision bits. The scaling of the input LLRs provided by the demodulator is varying

performance with respect to realistic conditions. Thus, we should chose an algorithm with respect to its robust behavior under all conditions, i.e., SNR mismatch effects as shown here, but as well for different modulation schemes, code rates, block sizes, or different channel models as well.

6.3 Turbo Codes Architecture

The previous sections presented turbo codes from an algorithmic point of view. Moving towards implementation, the architectural side of the decoder must be regarded. For hardware realization of a turbo decoder three important design steps have to be done.

- Realization of the processing of the component codes.
- Iterative exchange of the message, see Sect. 6.2.
- Interleaver realization.

As mentioned for future architecture we often have to realize a high throughput which results as well in parallel decoder architectures. Especially the interleaver realization can be a problem for a parallel implementation as already presented in Sect. 4.3. For the realization of the component decoder we have different possibilities. The most common architecture is the so called serial MAP architecture which is presented in the next section. When assembling all components together various design possibilities have to be considered. The design space is presented in Sect. 6.3.2.

6.3.1 Serial MAP Architecture

The component decoder of a turbo decoder has to realize a soft-input soft-output algorithm. Typically, the Max-Log MAP realization is implemented. It uses a forward-backward algorithm—corresponding data paths are derived in Chap. 5. Adapting a serial data path for the MAP architecture is often denoted as serial MAP architecture. The serial MAP (SMAP) architecture is the most common architecture in literature which is instantiated within a turbo decoder. The now presented architecture is based on the derived serial data path example of Sect. 5.1.2.

The decoding consists of two steps, the forward recursion and the backward recursion. Figure 6.12 shows the architecture of the forward processing, Fig. 6.13 shows the final architecture of the backward processing and output calculation.

We deal with a (information) block of length BL, thus we need $BL-1$ clock cycles for the forward processing and BL cycles for the backward and output processing. The block length processed in hardware is often $BL \leq K$, with K the number of information bits. BL in hardware can be smaller since we can either process two trellis steps within one clock cycle or we can even partition the entire block into so called windows. Both techniques are not treated in this section and are advanced

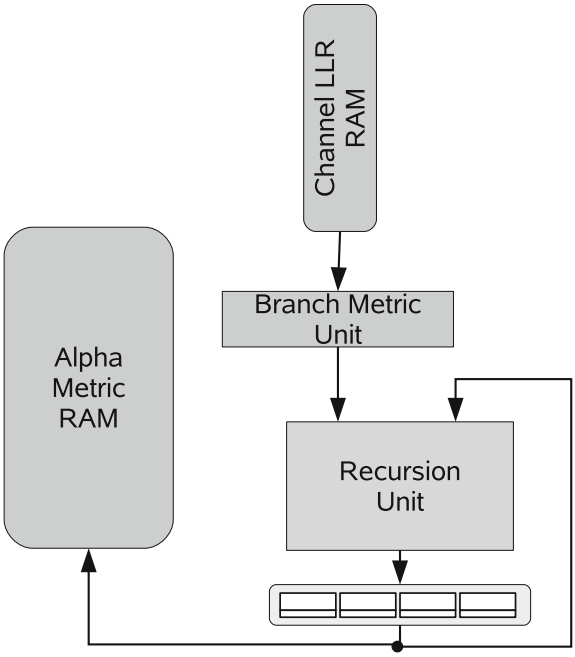


Fig. 6.12 SMAP architecture during forward recursion

optimization techniques to increase throughput or to decrease storage demands, for details we refer to [9]. For the discussion here a processing of one information bit per trellis step is assumed ($BL = K$).

We now introduce some generic parameters to describe corresponding number of values passed between the processing units, i.e., how many values are read, written or processed per clock cycle. We assume a convolutional code of rate R with 2^M states, with M the number of registers within the encoder. Depending on the utilized code rate R always $1/R$ values are read from the channel LLR memory. The branch metric unit calculates $2^{1/R}$ values and passes these to the recursion unit. The recursion unit, assumed to process all 2^M states in parallel, passes 2^M state metrics to the register bank. These 2^M values are stored in the alpha memory. When processing a trellis featuring 64-states as is used in the WiFi standard the number of bits to store or to process is very large. Under the assumption that each value is represented by 10 bits we have to store 640 bits per clock cycle. If BL is large, the alpha memory may becomes the largest part of the entire architecture. Yet considering turbo decoders, the number of states is restricted at most to 16 states, see Table 6.1.

Of course there are various possibilities to perform the forward-backward processing. Each data flow possibility results in different architectural characteristics for, e.g., input data retrieval, number of instantiated recursion units, latency, and resulting throughput. The starting point is often a graphical representation to analyze the so called life time analysis of processed data. The life time analysis of the presented architecture is shown in Fig. 6.14.

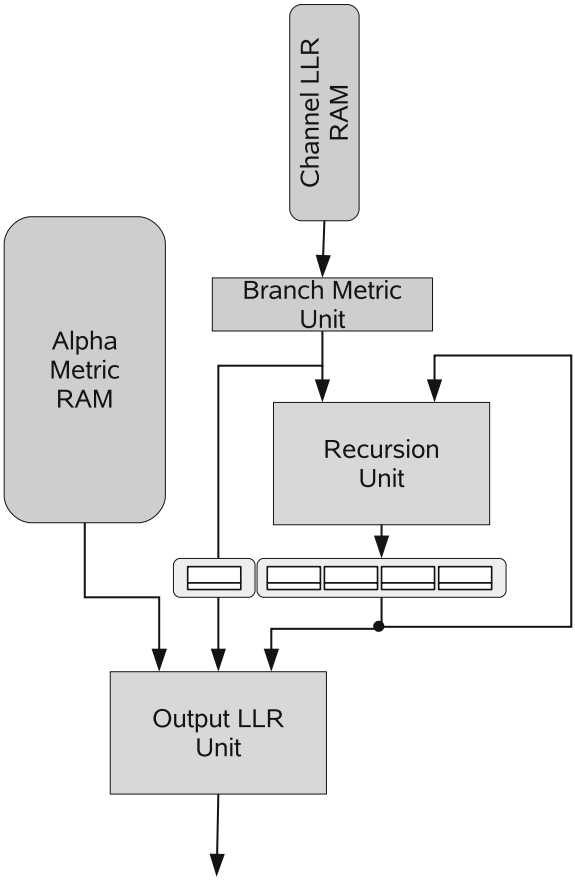


Fig. 6.13 SMAP architecture during backward recursion and output processing

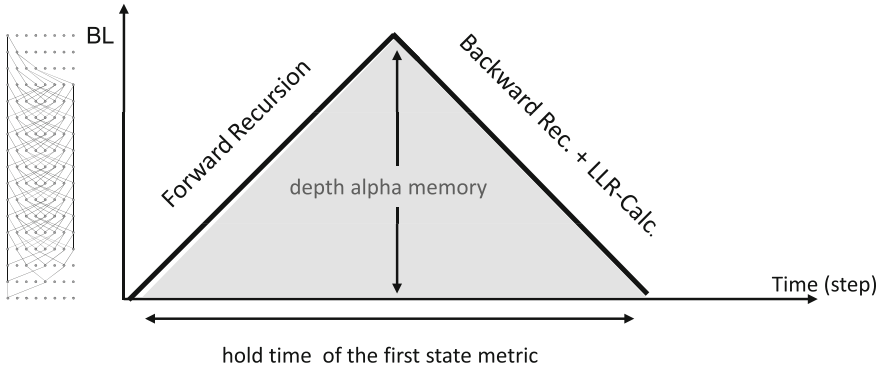


Fig. 6.14 Graphical data life time analysis. y axis reflects the position in a data block of length BL, the x axis reflects the time or trellis steps

The y-axis reflects the position in a data block of length BL, the x-axis reflects the processing time or trellis step. We analyze the graph from left to right. Each time step is associated with one recursion step. The processing of the forward recursion is indicated by the diagonal bottom left to top right. The processing of the backward recursion is depicted by a diagonal from top left to bottom right. At each time step exactly one recursion step is performed. Thus, it is obvious that we have to instantiate one RU which performs first the entire forward recursion and then the same RU is utilized for the backward recursion. The output processing is done simultaneously with the backward processing which requires additional logic. We can see that the input data has to be read from the first block position in an incrementing order. However, the output information is calculated in a reversed direction. Thus the result of position $BL-1$ is obtained first. One important information that we can extract from this representation is the storage time of the state metrics of each time step. The first state metrics obtained in the very first clock cycle is used again at the end of the backward recursion. This is the maximum storage time occurring during the backward forward processing. The depth of the state metric memory is determined by the hight of the triangle. Typically this memory is called state metric memory or alpha memory as already mentioned.

One important aspect of the backward forward algorithm is the possibility to perform the backward recursion first as shown in Fig. 6.15. The major difference for processing is the inverse reading sequence of the input values and also a new order of the output values. This can be of advantage when output values have to be provided in the original order for the next stage.

Figure 6.16 shows a different approach, leading to a different architecture with changed architectural characteristics. In each time step a forward and a backward recursion is active. Thus, two RU units have to be instantiated. After processing half of the block again two RUs and two output LLR units are processing data. As a result, two output data are provided per clock cycle. In this data flow always two distinct data sets, input data and state metrics, have to be provided to the processing units. The maximum storage time is reduced by a factor of two and the throughput

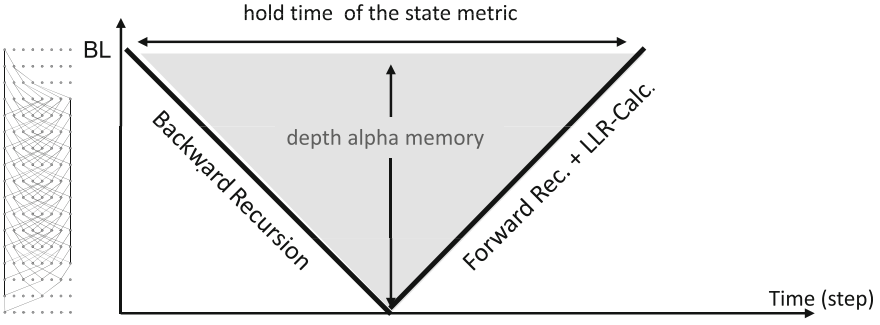


Fig. 6.15 Processing schedule: first the backward recursion then the forward recursion with output LLR processing

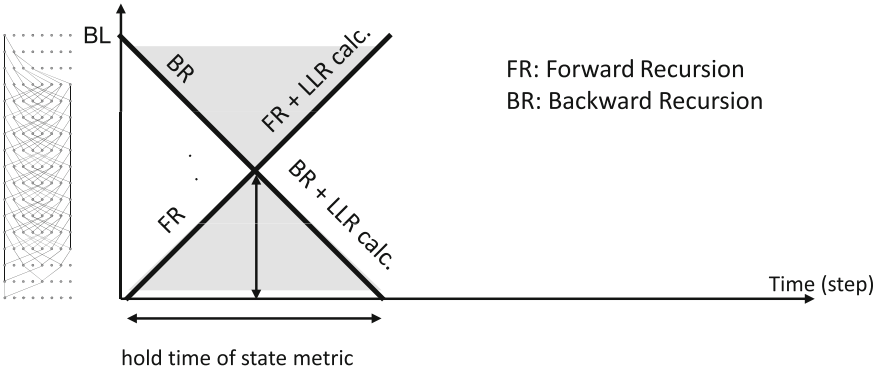


Fig. 6.16 Two recursions are running in parallel, forward and backward

increased by a factor of two compared to Fig. 6.14. The number of processing units is exactly doubled compared to Fig. 6.14, while the number of state metrics to be stored is identical. However, another access characteristic of the state metric memory results. In every clock cycle two accesses are required, either two write access during the first $BL/2$ clock cycles, or two reading accesses throughout the second half of the process. There are many other possibilities either to increase the throughput or to reduce the storage requirements.

The high throughput demands of a Max-Log-MAP decoder is mainly driven due to the increasing throughput demands of turbo code decoders.

6.3.2 Design Space and Design Choices

In this section the design space for turbo decoder architectures is presented along with the design choices which are most often utilized for industry driven designs. The possible design steps in the following are not described in detail, rather the most important references are given. The detailed description and compact analysis of full turbo decoder designs can be found in various thesis like [9–11].

High Level Architecture Decisions

The first and very important high level architectural decision is the question of coupling of the component decoders. Different methods how to exchange the information exist. Furthermore additional interface options have to be considered for storing the input and output data. We list the 'Pro' and 'Con' for each decision.

- It is possible to exchange combined extrinsic data plus information bit LLR within one half iteration, see Fig. 6.17a.

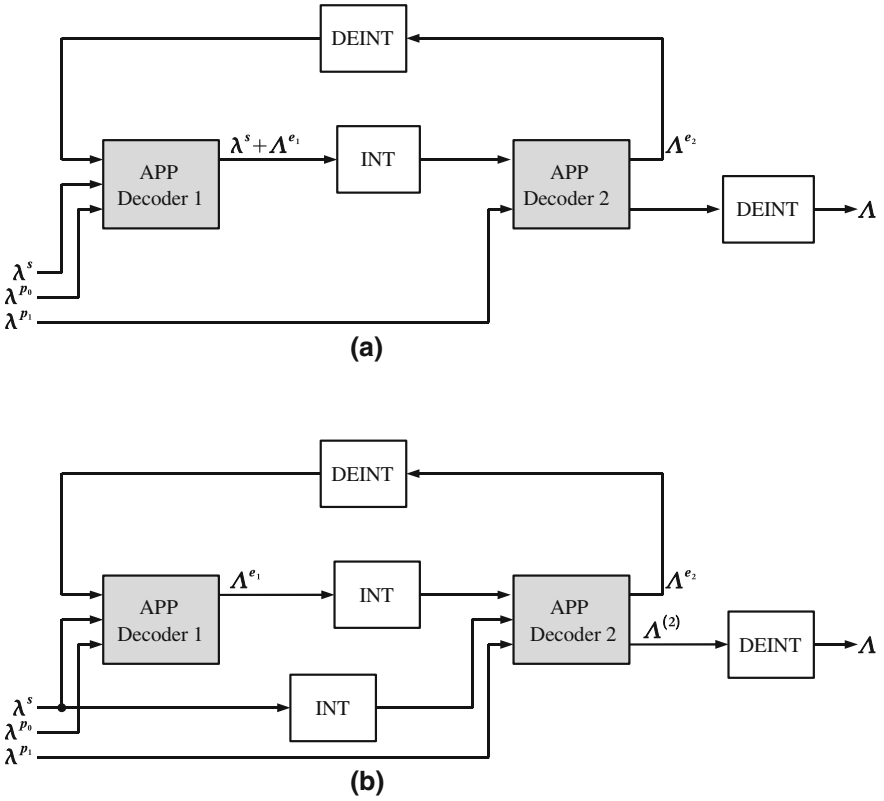


Fig. 6.17 **a** Turbo decoder which passes the full a posteriori information from MAP1 to MAP2, **b** with separate interleaved systematic information

  - Pro: The reading process and storing of interleaved systematic information can be omitted.
  - Con: The bit width of the exchanged information and thus stored information is increased.

- Reading the interleaved information bit LLR within one half iteration, see Fig. 6.17b.
  - Pro: A symmetric/identical processing of the component decoder is possible which enables, e.g., a simpler programming and a simpler control flow.
  - Con: The technique is less power efficient since an addition reading of the interleaved information becomes mandatory.

- Depending on the system considerations the input I/O features a double buffering scheme. The interface writes to one buffer (memory) while the other buffer is utilized for processing. The role of these two memory groups alternate for each block.

  - Pro: The interface causes no additional latency. A simple control flow results while the double I/O solution is often mandatory due to throughput and latency constraints.
  - Con: It doubles the area of input memories.

- A codeword featuring a code rate $R > 1/3$ is obtained by puncturing. The input interface has the possibility to store the punctured information as LLR $= 0$ to the corresponding memory address.
  - Pro: The decoder needs no knowledge about the puncturing scheme at all. The reading process for the channel values during MAP processing becomes straight forward.
  - Con: For high data rates all the unneeded 'zero' information is read and processed.

- Typically for the output part of the I/O interface only hard values are provided. However, providing soft-outputs for the information bits or parity bits is possible.
  - Pro: Soft-outputs can be used as feedback information to demodulator or synchronization unit.
  - Con: The output memories are increased. For the soft-output parity bits an additional output calculation stage has to be instantiated. Attention: the outer inter-block interleaving has to be implemented for a possible feedback loop as well.

- The turbo decoder can provide additional information for the MAC layer or previous processing stages. This additional information can enable e.g. dynamic time allocation to process a block within a TTI frame, or it can support the channel estimation etc.
  - The number of required iterations is one information which can be used for dynamic time allocation.
  - Monitoring the convergence speed by utilizing a reliability measure gives an information about the reliability of decoding.
  - Tracking the saturation level of the input data will help to indicate a wrong LLR scaling.

Quantization Issues

Quantization aspects directly influence the power consumption and the communications performance. Most of the current state-of-the-art implementations operate on 6 bits input quantization and 7 bits extrinsic quantization. For conservative designs and robustness towards SNR mismatch up to 8 bits are sometimes utilized for input LLRs. The input quantization directly influences the bit width of the state metrics. The bit width of sate metrics have to be normalized due to the accumulative functionality of the recursion units.

- An efficient state metric renormalization technique is the modulo normalization [12, 13]. The modulo normalization utilizes an overflow technique to limit the bit width of the state metrics. It is performed on the fly within the recursion unit.
  - Pro: The modulo normalization is a simple realization which requires no additional hardware units. The critical path is not prolonged.
  - Con: The bit width of each state metric is slightly larger compared to subtractive normalization.
- Limiting the bit width of the state metrics can be obtained by subtractive normalization. The normalization can be done by subtracting always the zero state or by subtracting the maximum state [12].
  - Pro: The bit width of the state metric can be kept as small as possible.
  - Con: The normalization during the recursion prolongs the critical path.
- Limiting the bit width of the stored state metrics can be achieved by subtractive normalization prior the state metric storage. In this case the normalization is best done by subtracting always the zero state [14].
  - Pro: For LTE or HSDPA only 7 state metrics have to be stored, i.e. one state can always be normalized to zero.
  - Con: It can be cumbersome if already a modulo normalization is used within the recursion units.

Data Path MAP Component

Data path aspects is one of the most published topics for component MAP implementation. An overview can be found in [15]. Major issue is: how to partition a block into sub-blocks which can be processed independently. This kind of partitioning is mandatory for high throughput turbo decoders. Two fundamental techniques can be distinguished. The block of length N is divided in P sub-blocks, always $\frac{N}{P}$ consecutive bit positions belong to one partition [16, 17]. The resulting MAP engine processing one sub-block is called serial MAP, see Chap. 5. The second partitioning possibility is to utilize a pipelined XMAP architecture which accepts P consecutive bit positions each clock cycle [18, 19], see Chap. 5.

- XMAP data flow
  - Pro: In [15] it is proven that for final LLR calculation of this data flow is most efficient in terms of state metric storage, if no acquisition phase is mandatory. Note that the basic data flow principle can be also used for SMAP decoders.
  - Con: The XMAP data flow requires an acquisition (training phase) from both sides (α, β). The longer the mandatory training phase the larger the additional overhead.

- SMAP data flow
  - Pro: The SMAP data flow is state-of-the-art and most often utilized in industry designs. The throughput scaling of the resulting architecture is straight forward.
  - Con: The state metric storage becomes large for a large window length, thus additional techniques should be applied.

Windowing Scheme

A block which is processed by one SMAP decoder can be partitioned further by a so called windowing scheme [19] which was already utilized for Viterbi decoding [20]. The boundaries of the windows can be initialized by state metrics of the previous iteration or by an acquisition (training) phase, see Fig. 6.18 and Fig. 6.19.

- Depending on the window size an additional training phase may become mandatory. The training phase is called acquisition (ACQ).
  - Pro: For long ACQ values the communications performance will be comparable to that without windowing scheme.

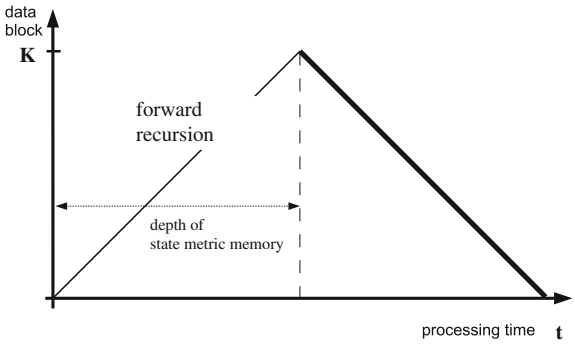


Fig. 6.18 MAP decoder which processes a full block of length K

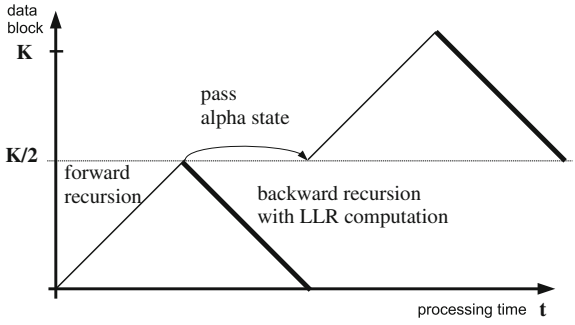


Fig. 6.19 MAP decoder which processes two windows, each with K/2 data

  - Con: Without additional hardware the latency of the decoder increases. With additional hardware the area increases and the control flow will get difficult.

- Next Iteration Initialization: State metrics at the window boundaries are stored at iteration i and reused at iteration $i + 1$ [14].
  - Pro: The technique is very effective to enhance the convergence of the decoder. It can be used as well in combination with ACQ.
  - Con: Non, it should always be utilized.

Recursion Unit

The recursion unit (RU) is the basic building block for performing the mandatory forward or backward processing. One basic issue is the number of instantiated RUs. An other more detailed question is the number of processed trellis steps within a RU unit. Radix-2 processes one trellis step, Radix-4 processes two trellis steps, respectively.

- Number of instantiated recursion units within a SMAP unit.
  - 1 RU: One recursion unit can process either α or β or an acquisition phase. This is a feasible solution for lower throughputs.
  - 2 RUs: Two recursion units process α and β concurrently. This is state-of-the-art and allows a huge variety of windowing schemes.
  - 3 RUs: Two RUs are used for α and β recursion, the third for acquisition phase. Note, that with 3 recursion units special care has to be put on the branch metric storage.

- The recursion unit can be implemented to process two trellis steps within one clock cycle. This implementation is called radix-4 unit [21, 22].
  - Pro: The number of stored state metrics reduces by a factor of two while the throughput is doubled.
  - Con: The critical path is longer compared to a radix-2 implementation, which may cause problems for stringent frequency constraints.

- Re-computation approach: state metric values (α or β) are only partially stored and recomputed during LLR calculation [16].
  - Pro: It reduces the state metric memory and also the power consumption.
  - Con: Additional RU units are required and the control flow is more complicated.

Parallel Interleaving

One difficult problem to support high throughput turbo decoding is the parallel interleaving. The possible memory access problems can cause big problems. Especially

for HSPA advance which requires throughput rates up to 150Mbit/s. Note that LTE turbo codes feature interleavers which provides a conflict free access scheme with respect to a maximum parallelism of 64.

Three different possibilities exist to store the corresponding soft-output values of the component decoders. The soft-values which have to be interleaved can be stored before or after (de)interleaving, as shown in Fig. 6.20.

- Access scheme a: The interleaving is performed during writing.
  - Pro: There exists an identical flow for both component decoders. Occurring conflicts may be resolved by buffering or flow control methods.
  - Con: A worst case analysis has to be done for all block sizes to ensure the required throughput demands.
- Access scheme b: The interleaving and deinterleaving is performed during reading and writing data of the second component decoder.
  - Pro: One component MAP can be realized highly parallel since no conflicts occur. This procedure seems to be logical for an unified decoder (LTE and HSPA) since the maximum throughput demand of a LTE mode is typically 2 times higher than the corresponding HSPA mode.
  - Con: The second component decoder should operate on a different parallelization level since conflicts have to be resolved during reading and writing.

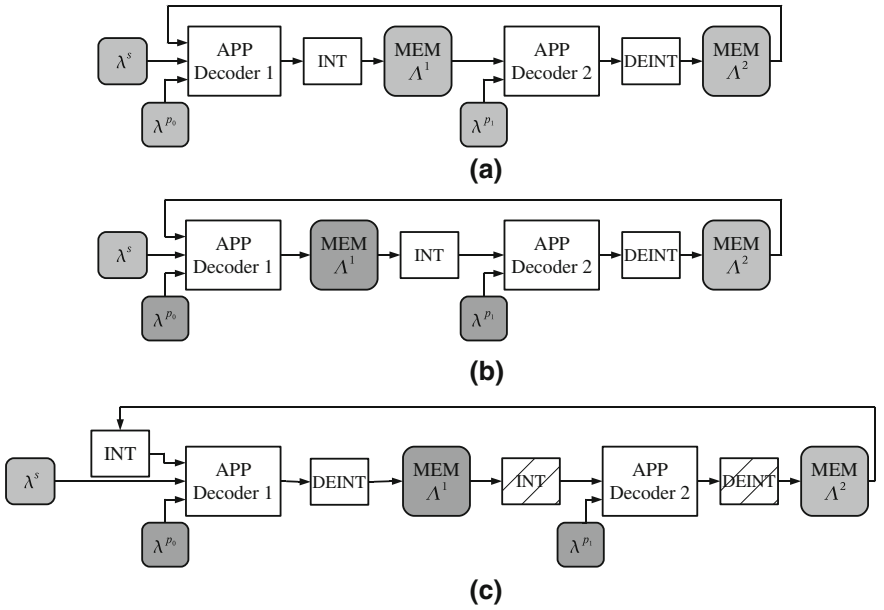


Fig. 6.20 Storage and interleaving of exchanged soft-values: **a** interleaving and deinterleaving upon write; **b** interleaving upon read, deinterleaving upon write, **c** two stage interleaving process

- Access scheme c: Here a two stage interleaving process is assumed with two interleaver tables.
  - Pro: This access scheme allows a parallel processing of all HSPA interleavers [23].
  - Con: For each block length and thus interleaver pre-computed access patterns have to be stored. The storage demand of this technique is high. Note that a different pattern for MAP1 and MAP2 exists.

Low Power Techniques

Iteration control is one of the most important techniques to reduce the power consumption. Goal is to reduce the average number of iterations. It has to be distinguished between techniques to detect undecodable blocks and decodable blocks. Control criteria can be based on soft (reliable) or hard information. An overview of different techniques is presented in [24].

- Iteration control criteria based on soft information. Soft information can be either the exchanged extrinsic information or the computed APP information.
  - Pro: It does work for undecodable blocks by tracking the convergence of the decoder.
  - Con: In normal mode (decodable blocks) the additional energy consumption and overhead can be large. There may exists a high false alarm rate for varying channels.
- Iteration control criteria based on hard information.
  - Pro: The implementation comes with only a small hardware overhead and does work for decodable blocks. It is especially good for, e.g., HSPA decoding.
  - Con: The detection of undecodable blocks is not reliable. It has to be switched of for LTE decoding since the existing CRC check is more reliable.
- LTE: CRC check hardware instances after MAP1 and MAP2, since the decoding process may oscillate for very high code rates [25].
  - Pro: It is very important to perform the CRC after each MAP.
  - con: Two separate CRC units may be mandatory due to latency reasons.

6.3.3 Dependencies of Architectural Parameters

In this section we show the dependencies of various architectural parameters on throughput and area of state-of-the art turbo decoders. We use the following nomenclature and parameters:

t_{cond}	technology node and operating conditions w.r.t. feature size variation, V_{dd}, temperature
f_{cyc}	frequency of the design
$Q = Q_{in}$	quantization of the input data, the quantization of all other variables are derived from this
$iter$	iteration number
K	number of information bits
WL	window length, in hardware the codeword is processed window by window
AL	acquisition length, the number of training steps for the forward or backward recursion
rdx	radix-2 or radix-4 realization of the recursion units
P	architectural parallelism
C_N	latency due to network to realize the interleaving, strongly depends on P and interleaver structure

The throughput (T) for state-of-the-art turbo decoder architectures can be calculated by the frequency times the number of cycle needed to process an information word of length K. An increased throughput requires a higher parallelism of the architecture, which increases the number of overhead cycles for interleaving $C_N(P)$. The throughput is given as follows:

$$T = f_{cyc} \cdot \frac{K}{2 \cdot iter \cdot \left(\frac{WL+AL}{log_2(rdx)} + \frac{K}{P \cdot log_2(rdx)} + C_N(P) \right)} \tag{6.4}$$

The frequency itself mainly depends on technology t_{cond}, the quantization of the input data, and the critical path in the combinatorial logic.

The area A_{all} of a turbo decoder is composed of three parts:

$$\begin{aligned} A_{all} = {} & P \cdot A_{MAP}(t_{cond}, Q, WL, AL, rdx) \\ & + A_{ctrl}(t_{cond}) \\ & + A_M(t_{cond}, Q, WL, AL, rdx, P). \end{aligned} \tag{6.5}$$

A_{MAP} is the logic area for a single MAP processing kernel which has to be instantiated P times depending on the parallelism. A_{ctrl} is the area of the controller which is typically small compared to the other two parts. A_M is the required area for instantiated memories. The area to store the input data ($\mathrm{A}_M^{I/O}$), the area of extrinsic data (A_M^{extr}) which are exchanged between component decoders, and the area to store state metric values used within the processing units (A_M^{MAP}). Thus the area of the memory which is a large portion of the overall area is given by

$$\begin{aligned} A_M = {} & P \cdot \mathrm{A}_M^{MAP}(t_{cond}, WL, Q, rdx) \\ & + \mathrm{A}_M^{I/O}(t_{cond}, P, K, Q) \\ & + \mathrm{A}_M^{extr}(t_{cond}, P, K, Q). \end{aligned} \tag{6.6}$$

In a similar way it is possible to derive equations for the energy consumption. Energy consumption of a turbo decoder can be expressed as

$$E_{block} = iter \cdot [2 \cdot P \cdot \left(WL + AL + \frac{K}{P}\right) \cdot E_{kernel}(t_{cond}, P, Q) + E_{network}(t_{cond}, P, Q)] + E_{I/O}(t_{cond}, Q) \quad (6.7)$$

E_{block} is the energy consumption of the entire block and depends of course directly on the number of utilized iterations. The final energy per bit is thus

$$E_{bit} = \frac{E_{block}}{K} \quad (6.8)$$

The power consumption results in

$$P = \frac{E_{block}}{2 \cdot it \cdot \left(\frac{WL+AL}{log_2(rdx)} + \frac{K}{P \cdot log_2(rdx)} + C(P)\right) + I_{I/O}} \cdot f(t_{cond}, Q) \quad (6.9)$$

Area and power results of various turbo decoder implementations will be presented in Chap. 9. Understanding the trade-offs between implementation efficiency, communications performance and flexibility will be key for designing efficient turbo decoders. Meaningful efficiency metrics are mandatory to explore and evaluate the resulting huge design space which will be presented as well in Chap. 9.

References

1. Berrou, C., Glavieux, A., Thitimajshima, P.: Near Shannon limit error-correcting coding and decoding: turbo-codes. In: Proceedings of 1993 International Conference on Communications (ICC '93), pp. 1064–1070, Geneva, Switzerland (1993)
2. ten Brink, S.: Convergence behavior of iteratively decoded parallel concatenated codes. IEEE Trans. Commun. **49**(10), 1727–1737 (2001). doi:10.1109/26.957394
3. ten Brink, S., Kramer, G., Ashikhmin, A.: Design of low-density parity-check codes for modulation and detection. IEEE Trans. Commun. **52**(4), 670–678 (2004). doi:10.1109/TCOMM.2004.826370
4. May, M.: Dissertation in preparation: Architectures for High-throughput and Reliable Iterative Channel Decoders. Ph.D. Thesis, Department of Electrical Engineering and Information Technology, University of Kaiserslautern (2012)
5. Vogt, J., Finger, A.: Improving the Max-Log-MAP turbo decoder. IEEE Electron. Lett. **36**, 1937–1939 (2000)
6. Michel, H., Wehn, N.: Turbo-decoder quantization for UMTS. IEEE Commun. Lett. **5**(2), 55–57 (2001)
7. Wu, Y., Woerner, B.D.: The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes. In: Proceedings of 1999 International Conference on Vehicular Technology (VTC '99), pp. 1683–1687 (1999)
8. Worm, A., Hoeher, P., Wehn, N.: Turbo-decoding without SNR estimation. IEEE Commun. Lett. **4**(6), 193–195 (2000)

9. Vogt, T.: A Reconfigurable Application-specific Instruction-set Processor for Trellis-based Channel Decoding. Ph.D. Thesis, University of Kaiserslautern (2008)
10. Worm, A.: Implementation Issues of Turbo-Decoders. Ph.D. Thesis, University of Kaiserslautern (2001). ISBN 3-925178-72-4
11. Alles, M.: Implementation Aspects of Advanced Channel Decoding. Ph.D. Thesis, University of Kaiserslautern (2010)
12. Hekstra, A.P.: An alternative to metric rescaling in Viterbi decoders. IEEE Trans. Commun. **37**(11), 1220–1222 (1989). doi:10.1109/26.46516
13. Worm, A., Michel, H., Gilbert, F., Kreiselmaier, G., Thul, M.J., Wehn, N.: Advanced implementation issues of turbo-decoders. In: Proceedings of 2nd International Symposium on Turbo Codes & Related Topics, pp. 351–354, Brest, France (2000)
14. Dielissen, J., Huiskens, J.: State vector reduction for initialization of sliding windows MAP. In: Proceedings of 2nd International Symposium on Turbo Codes & Related Topics, pp. 387–390, Brest, France (2000)
15. Mansour, M.M., Shanbhag, N.R.: VLSI architectures for SISO-APP decoders. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **11**(4), 627–650 (2003)
16. Schurgers, C., Engels, M., Catthoor, F.: Energy efficient data transfer and storage organization for a MAP turbo decoder module. In: Proceedings of 1999 International Symposium on Low Power Electronics and Design (ISLPED '99), pp. 76–81, San Diego, California, USA (1999)
17. Dawid, H., Meyr, H.: Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding. In: Proceedings of 1995 International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '95), pp. 193–197, Toronto, Canada (1995)
18. Worm, A., Lamm, H., Wehn, N.: VLSI architectures for high-speed MAP decoders. In: Proceedings of Fourteenth International Conference on VLSI Design, pp. 446–453, Bangalore, India (2001)
19. Dawid, H., Gehnen, G., Meyr, H.: MAP channel decoding: algorithm and VLSI architecture. In: VLSI Signal Processing VI, pp. 141–149. IEEE (1993)
20. Fettweis, G., Meyr, H.: High-speed parallel Viterbi decoding: algorithm and VLSI-architecture. IEEE Commun. Mag. **29**, 46–55 (1991)
21. Bickerstaff, M., Davis, L., Thomas, C., Garrett, D., Nicol, C.: A 24 Mb/s radix-4 LogMAP turbo decoder for 3GPP-HSDPA mobile wireless. In: Proceedings of 2003 IEEE International Solid-State Circuits Conference (ISSCC '03), pp. 150–151, 484, San Francisco, CA, USA (2003)
22. Black, P.J., Meng, T.H.: A 140-Mb/s, 32-state, radix-4 Viterbi decoder. IEEE J. Solid-State Circ. **27**(12), 1877–1885 (1992)
23. Tarable, A., Benedetto, S., Montorsi, G.: Mapping interleaving laws to parallel turbo and LDPC decoder architectures. IEEE Trans. Inf. Theory **50**(9), 2002–2009 (2004). doi:10.1109/TIT.2004.833353
24. Gilbert, F., Kienle, F., Wehn, N.: Low complexity stopping criteria for UMTS turbo-decoders. In: Proceedings of VTC 2003-Spring. The 57th IEEE Semiannual Vehicular Technology Conference, pp. 2376–2380, Jeju, Korea (2003)
25. May, M., Ilnseher, T., Wehn, N., Raab, W.: A 150 Mbit/s 3GPP LTE turbo code decoder. In: Proceedings of Design, Automation and Test in Europe, 2010 (DATE '10), pp. 1420–1425 (2010)

Chapter 7
Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were introduced in 1960 by R. Gallager [1] in his Phd thesis. He already introduced the iterative method for decoding LDPC codes. However, also due to their computational and implementation complexity the iterative decoding was largely ignored. Before the introduction of turbo codes only a small group of researchers were interested in the field of iterative decoding, e.g. P. Elias [2], M. Tanner [3]. With the rise of turbo codes 1993 iterative decoding turned into focus again and thus LDPC codes were re-discovered in 1996 by MacKay and Neal [4].

Since 1996 LDPC codes have experienced a renaissance and they are among the best codes, especially for very large block lengths. Many communications standards feature LDPC codes as their channel coding scheme. The most prominent standards utilizing LDPC codes are the second generations of digital video broadcasting services (DVB-T2 [5], DVB-S2 [6], DVB-C2 [7]). Further standards featuring LDPC codes are WiMAX IEEE 802.16 [8], WLAN 802.11n [9], 802.3an [10],WiMedia 1.5 [11], and 802.15.3c [12]. The standards with corresponding codeword length and rates are shown in Table 7.1. Shown is as well the maximal throughput for each standard, e.g. for the 10 GBASE-T Ethernet standard the throughput is 10 Gigabit per second.[1]

7.1 Basic Definition

An LDPC code is a linear block code defined by its sparse $M \times N$ parity check matrix H. The code is represented as the set of all binary solutions $\boldsymbol{x} = (x_0, x_1, ..., x_{N-1})$ to the linear algebraic equation

$$H \cdot \boldsymbol{x}^T = \mathbf{0}^T \tag{7.1}$$

[1] Further list of the standards and the details parameter settings are shown in the appendix (A).

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_7, © Springer Science+Business Media New York 2014

Table 7.1 Selection of standards utilizing LDPC codes, from [13]

Standard	Codes	Throughput	Code rates	Codeword size
IEEE802.11n (WiFi) [9]	LDPC	600Mbit/s	1/2, 2/3, 3/4, 5/6	Up to 1620
IEEE802.16e (WiMAX) [8]	LDPC	96 Mbit/s	1/2, 2/3, 3/4, 5/6	Up to 1920
DVB-S2/T2/C2 [5–7]	LDPC	$\sim$ 90 Mbit/s	1/5–9/10	1,6200,64,800
WiMedia 1.5 (UWB) [11]	LDPC	$\sim$ 1000Mbit/s	1/4–4/5	1,200,1320
802.15.3c (60 GHz) [12]	LDPC	$\sim>$ 500Mbit/s	1/2,3/4,7/8,14/15	672, 1,440
802.3an [10] (10 GBASE-T)	LDPC	$\sim$ 10 Gbit/s	0.84	2048

The elements of the parity check matrix are 0s and 1s. The entries could also be elements of a finite field $GF(p)$. Here, only binary codes $GF(2)$ are considered. The multiplication of $\boldsymbol{x}$ by a row of H corresponds to a parity check equation, which is the *XOR* of the bits in $\boldsymbol{x}$ at the '1' positions in the row of H.

Gallager introduced so called (j, k) **regular** LDPC codes, where each column of H contains j 1s and each row contains k 1s. (j, k) are called column and row weight respectively. The fraction of 1s in the parity check matrix is $\frac{M \cdot k}{M \cdot N} = \frac{k}{N}$, which gets very small for large N, thus the name low-density. The resulting code rate is $R = (N - M)/N$ if H is of full rank, i.e. all rows are linearly independent.

A good way to represent an LDPC code is a bipartite graph or Tanner graph [3]. Figure 7.1 shows the H matrix of a (3,6) parity check code and the resulting Tanner graph. The M check nodes (CN) correspond to the parity constraints (rows in H), the N variable nodes (VN) represent the columns in H. An edge in the graph corresponds to a 1 entry in the parity check matrix. As an example the connectivity of the first check node is highlighted.

Figure 7.1 shows the Tanner graph of a regular LDPC code, however, it was shown that irregular or structured codes show superior communications performance. **Irregular** LDPC codes have nodes of varying degrees and were introduced in 1997 by Luby et al. [14]. The communications performance of an irregular LDPC code can be superior to that of a regular one.

An irregular LDPC code is defined by its degree distributions $(\boldsymbol{f}, \boldsymbol{g})$ instead of the tuple (j, k). The degree distribution $\boldsymbol{f}$ gives the fraction of variable nodes for a certain degree. The variable node degree distribution can contain many nodes of different

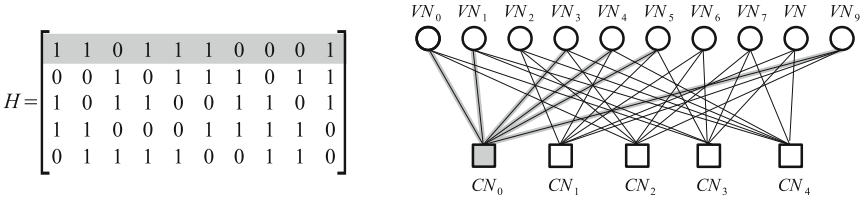


Fig. 7.1 Parity check matrix H and the corresponding Tanner graph representation

degrees $\boldsymbol{f} = (f_1, f_2, f_3, \ldots, f_{VN^{max}})$, while the check node degree distribution $\boldsymbol{g}$ often contains two or three different CN degrees $\boldsymbol{g} = (g_{CN^{max}-2}, g_{CN^{max}-1}, g_{CN^{max}})$.

A Tanner graph for irregular LDPC codes is shown in Fig. 7.2, the node fractions f_1 are here omitted since these type of VNs need a special type of structure which is explained in Sect. 7.5. The connections between VNs and CNs are indicated by the connectivity box Π. The degree distribution $(\boldsymbol{f}, \boldsymbol{g})$ defines an ensemble of codes, in conjunction with a fixed Π a single code is specified.

The degree distributions for the WiMAX LDPC codes are shown in Table 7.2. Three different code rates are specified in the standard. The codeword size ranges from $N = 576$ bits to $N = 2304$ bits with a step size of 24 bits. For the code rate $R = 2/3$ and $R = 3/4$ two different codes are specified, here indicated as code **A** and **B**. Note, that both codes show nearly identical communication performance, while both codes can be implemented with an advanced scheduling method (layered decoding) which is presented in the next section.

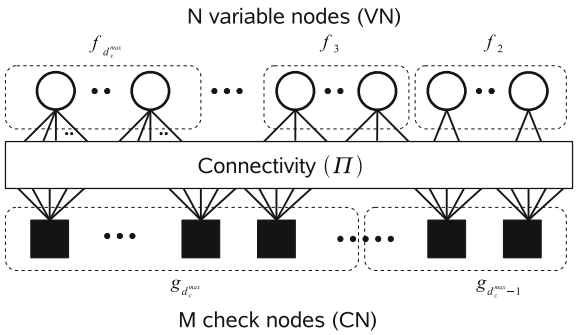


Fig. 7.2 Tanner graph of an irregular LDPC code

Table 7.2 Degree distributions of all WiMAX 802.16 LDPC code classes, the codeword size ranges from N=576 to N=2304 bit in steps of 24 bits

R	WiMAX 802.16	
	Variable node degree f	Check node degree g
1/2	$f_{[2,3,6]} = \{11/24, 1/3, 5/24\}$	$g_{[6,7]} = \{2/3, 1/3\}$
2/3 A	$f_{[2,3,6]} = \{7/24, 1/2, 5/24\}$	$g_{[10]} = \{1\}$
2/3 B	$f_{[2,3,4]} = \{7/24, 1/24, 2/3\}$	$g_{[10,11]} = \{7/8, 1/8\}$
3/4 A	$f_{[2,3,4]} = \{5/24, 1/24, 3/4\}$	$g_{[14,15]} = \{5/6, 1/6\}$
3/4 B	$f_{[2,3,6]} = \{5/24, 1/2, 7/24\}$	$g_{[14,15]} = \{1/3, 2/3\}$
5/6	$f_{[2,3,4]} = \{3/24, 5/12, 11/24\}$	$g_{[20]} = \{1\}$

7.2 Decoding

LDPC codes are decoded by using a message passing (MP) algorithm [1] either in hard or soft decision form. Typically, a soft decision algorithm is used which is then called belief propagation (BP). The decoding is an iterative process which exchanges soft-values (beliefs) between variable and check nodes along the edges of the graph. Each node can be seen as an independent component decoder which calculates an update of the incoming messages. The most common message update scheme is the two-phase MP. There in the first phase, all variable nodes are processed and in the second phase all check nodes. This two-phase MP is described in the following:

1. Initialize each VN with the received channel LLR λ_n^y of the associated bit of the codeword. The codeword position index $n \in \{0, \ldots, N-1\}$ is dropped in the following. In the first step all outgoing messages of each VN are initialized with the corresponding λ^y value.
2. Propagate the outgoing VN messages to the CNs along the edges in the graph.
3. Process the check nodes. Each CN calculates new probabilities for each incident edge. For a CN of degree k the resulting output λ_l of edge l can be calculated by

$$\tanh\left(\frac{\lambda_l}{2}\right) = \prod_{i=0, i\neq l}^{k-1} \tanh\left(\frac{\lambda_i}{2}\right). \tag{7.2}$$

   $i \neq l$ means that we do not take this edge into account for calculation. An outgoing message from a CN to a VN represents the belief that the check node condition is fulfilled assuming the corresponding bit of the codeword is either zero or one.
4. Propagate the outgoing CN messages back to the VNs along the edges of the graph.
5. Process the variable nodes. Each variable node calculates a new MAP probability of the associated bit of the codeword which is the sum of all incoming messages to the corresponding VN.

$$\Lambda = \lambda^y + \sum_{i=0}^{j-1} \lambda_i. \tag{7.3}$$

   The sign of the resulting APP information is passed as output information for the corresponding bit estimate $\hat{x}$. In a second step we have to update the information of each incident edge. For a VN of degree j the resulting output λ_l of edge l can be calculated by

$$\lambda_l = \lambda^y + \sum_{i=0, i\neq l}^{j-1} \lambda_i. \tag{7.4}$$

   An outgoing message from a VN to a CN represents the belief that the associated bit of the codeword is either zero or one.

6. Repeat steps (2–5) until all parity checks are fulfilled (Eq. 7.1), which is $\boldsymbol{H}\hat{\boldsymbol{x}}^{\boldsymbol{T}} = 0$. The decoding is stopped as well after a fixed number of iterations.

Note that the processing of variable and check nodes are exactly the symbol-by-symbol MAP calculations of repetition codes and single-parity check-code as described in Sect. 3.3.3. The message exchange between VNs and CNs again utilizes the basic extrinsic information concept. Only the additional gain has to be passed to the next processing node.

Further Scheduling Methods

The described decoding method is only one scheduling method. As mentioned, an LDPC code is composed of so called repetition codes and single-parity check-codes. Each row in $\boldsymbol{H}$ represents one single-party check-code and each column one repetition code. For the iterative decoding different scheduling methods for the exchange of the messages exist. Figure 7.3 shows the message flow of three different scheduling possibilities, note that hybrid update schemes are always possible. Each message flow in Fig. 7.3 starts from left, while the next update step evolves to the right.

The top graph shows the classical two-phase scheduling which was described in the previous section. First all variable nodes are active, then the messages are sent to all check nodes and finally back to all variable nodes. One iteration is finished after one round trip (VNs $\rightarrow$ CNs $\rightarrow$ VNs). The advantage of this scheduling method is the clear separation between the processing of VNs and CNs. There are many situations during a hardware realization where only this kind of scheduling is possible, e.g. a fully parallel implementation as explained in Sect. 7.4.

The figure in the middle shows the message flow of a so called layered schedule (horizontally). Here, the first step is to pass all messages to the first CN. Only this messages are passed which belong to the first row in $\boldsymbol{H}$, thus all other CNs are inactive. The first CN calculates an update of the messages and passes them back to the corresponding VNs. Only the VNs with new input messages update there values according to Eq. (7.4). In fact the VNs at the current iteration Λ_{VN}^{iter} assuming a new information at edge l can be calculated by

$$\Lambda_{VN}^{iter} = \Lambda_{VN}^{iter-1} - \lambda_l^{iter-1} + \lambda_l^{iter}. \tag{7.5}$$

If a message of the previous iteration was not yet initialized, we assume $\lambda_l^{iter-1} = 0$. In a next step all updated VN APP messages are passed to the second CN and so on. One iteration is finished when all CNs have been processed once. At each iteration we have to ensure that the extrinsic information principle is maintained.

The advantage of this scheduling method is the continuous update of the APP messages at the variable nodes which results in a faster convergence of the communications performance which is shown below. Note, that this scheduling can be applied to entire groups of CNs.

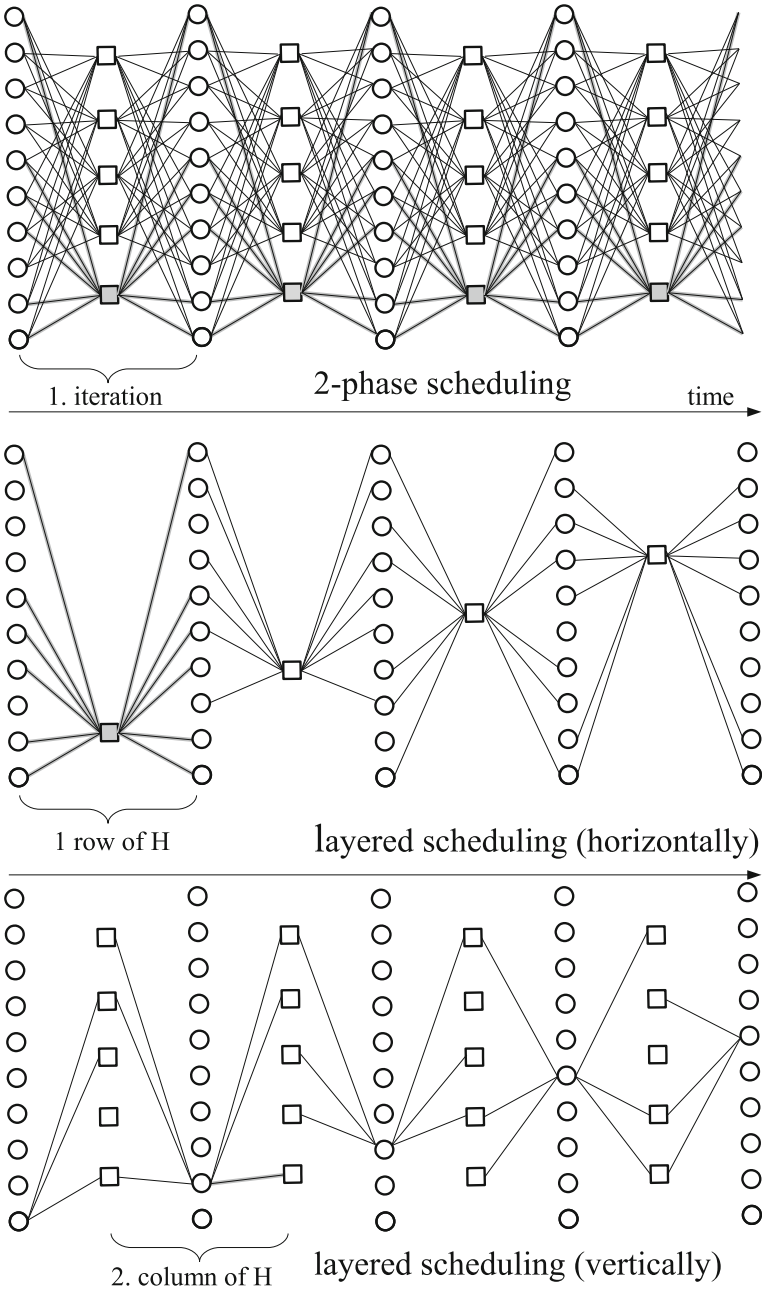


Fig. 7.3 Three different scheduling methods: two-phase, layered (horizontally), layered (horizontally), and layered (vertically)

Shown at the bottom of Fig. 7.3 is a so called vertical scheduling. The update scheme is related to the horizontal one, however, now variable node centric. One iteration is finished when all VNs have been processed. The trick of this scheduling method is to hold the intermediate results of each CN which is then continuously updated. The intermediate result of a CN can be expressed by the full product of Eq. (7.2). This vertical or CN centric scheduling is of interest for realizing high throughput LDPC decoder which have to process high code rates [15].

Examples of Communications Performance

Figure 7.4 shows the communications performance of a WiMAX LDPC code with a codeword length of $N = 2,304$ bits and a code rate of $R = 0.5$. The corresponding degree distribution is presented in Table 7.2, while the full code structure with respect to Π in Fig. 7.2 is specified in the standard. Shown are the performance results for different iterations (5, 10, 20, 40) utilizing two different scheduling methods (two-phase and horizontally layered). The layered scheduling method shows a faster convergence behavior. E.g the 10th iteration of the layered decoding has an equivalent communications performance as the 20th iteration of a two-phase scheduling. Note

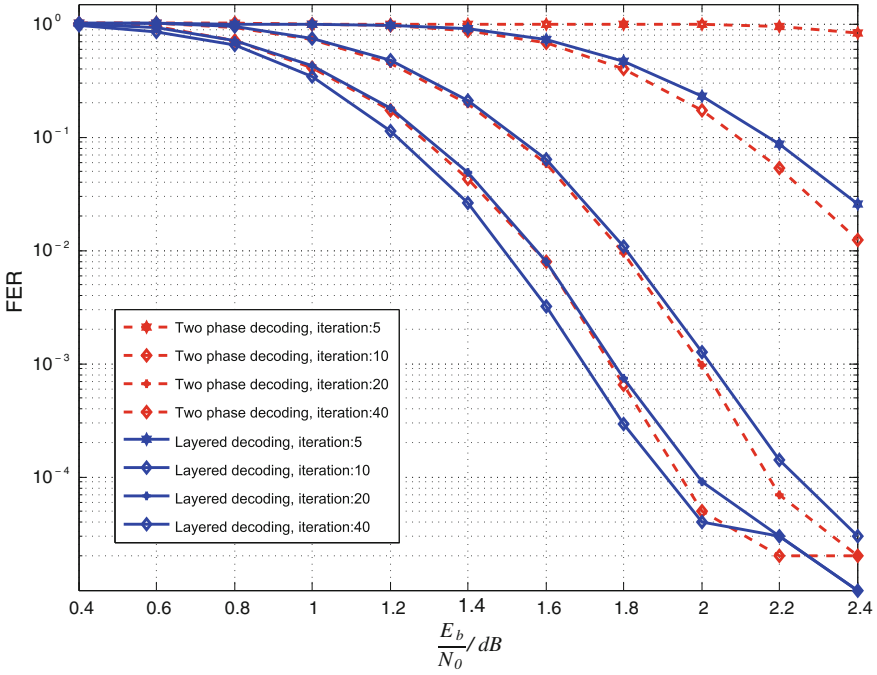


Fig. 7.4 Communications performance for two-phase vs layered schedule for a WiMAX LDPC code of codeword size $N = 2,304$ and $R = 0.5$

that this faster convergence behavior of the layered decoding is especially true for lower code rates. For very high code rates the convergence speed advantage slightly diminishes, since the overall convergence is faster for high code rates. For LDPC decoder hardware realizations the layered scheduling should always be utilized, if possible.

One major enabler for all iterative decoding techniques is the assumption that the incoming messages are independent. As long as this can be ensured each component decoder computes an optimal symbol-by-symbol decision and passes this message to other component decoders. The message passing results in an optimal symbol-by-symbol MAP decoding if the Tanner graph is cycle free [16]. A cycle in the Tanner graph is defined as the path with the same starting and ending VN, while an edge can be traveled only once. For finite block length the Tanner graph will contain cycles. Once a message has completed a cycle, the nodes update become suboptimal. Therefore, the longer the cycles the longer the optimality of the iterative decoding process. During code design we often try to maximize the girth of the graph. The girth is defined as the shortest cycle in the graph. The iterative message passing algorithm is thus a clever heuristic for the decoding problem. Now, the question arises, how much is the communication performance loss in dB when comparing to the optimal ML decoding. Solving the optimal ML criterion is NP-hard and can only be evaluated for small block sizes. The ML solution for this code was found by utilizing so called integer linear (IP) programming methods [17]. Figure 7.5 shows the ML decoding

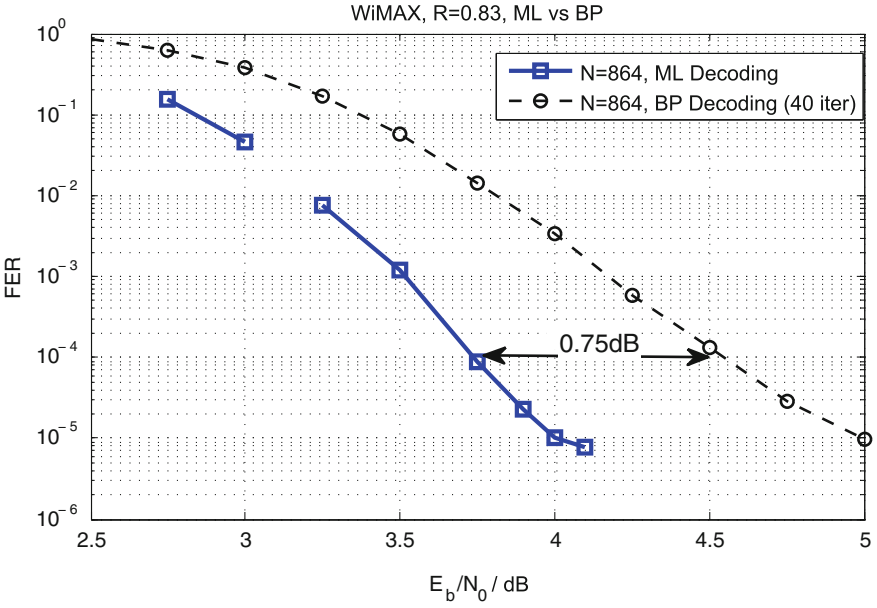


Fig. 7.5 ML performance vs. BP performance (40 iterations) of a WiMAX LDPC code, with codeword length of $N = 864$ and $R = 0.83$ [18]

performance compared to 40 iterations layered decoding. The utilized code is a WiMAX code of codeword length of $N = 864$ and a code rate of $R = 0.83$. The performance difference is 0.75 dB at $FER = 10^{-4}$. Note that this gap is supposed to get smaller for larger block sizes. However, the gap shows that there is still room for improvement for a decoding heuristic, at least for smaller block sizes.

7.3 Structured LDPC Codes

As mentioned in the previous chapter an LDPC code can be defined by its degree distribution of the variable nodes and the check nodes. However, starting with a degree distribution we still have to realize the message exchange between the VNs and CNs, which is defined by the connectivity pattern Π. An unstructured placement of the edges may have disadvantage for the communications performance and will be an obstacle for the corresponding hardware realization. Two important techniques to design structured codes are introduced here, the quasi-cyclic LDPC codes and the multi-edge type LDPC codes.

7.3.1 Quasi-Cyclic LDPC Codes

Low latency and high throughput LDPC decoder architectures have to realize the exchanging of messages between variable and check nodes. It was soon realized that a random connectivity structure of a Tanner graph poses big problems for the decoder realization, since a network has to handle this parallel message exchange between both kinds of nodes. For a random connectivity, memory access conflicts will occur, which are identical to the problem of parallel interleaving as shown in Sect. 4.3.2.

Thus, quasi-cyclic LDPC codes where introduced which have big advantages for the hardware realization. Quasi-cyclic LDPC codes are composed of sub-matrices H_{m_z,n_z} of size $z \times z$:

$$H^{Macro} = \begin{pmatrix} H_{1,1} & H_{1,2} & \dots & H_{1,N/z} \\ H_{2,1} & H_{2,2} & \dots & H_{2,N/z} \\ \vdots & \vdots & \ddots & \vdots \\ H_{M/z,1} & H_{M/z,2} & \dots & H_{M/z,N/z} \end{pmatrix}, \text{ with } H_{m_z,n_z} = \begin{cases} 0 \\ I^x \\ I^x + I^y + \dots \end{cases} \tag{7.6}$$

The sub-matrices H_{m_z,n_z} are either the zero matrix or given by a cyclically shifted identity matrix I^x of size $z \times z$ with $x - 1$ the amount of circular right of left shifts. Some codes also use superposed shifted identity matrices, where more than one shift value per sub-matrix is defined ($I^x + I^y + \dots$). However, these superposed identity

matrices may pose a problem for a decoder hardware realization and should be avoided, if possible, during code construction, see [13]. In communication standards like WiMAX, WiFi or WiMedia so called macro matrices are defined, with the corresponding shift values. We have to distinguish between a so called macro matrix, at which only the positions of a shifted identity matrix is shown, and a detailed realization for a given block length. For example, the description of one specific WiMAX LDPC code ($N = 576$, $M = 288$) results in:

$$H^{Macro} = \begin{pmatrix}
0 & 24 & 19 & 0 & 0 & 0 & 0 & 0 & 14 & 21 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 7 & 0 & 0 & 0 & 6 & 20 & 3 & 0 & 0 & 0 & 4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 6 & 21 & 0 & 9 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
16 & 0 & 12 & 0 & 0 & 0 & 0 & 0 & 17 & 7 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 10 & 0 & 0 & 0 & 22 & 0 & 0 & 11 & 19 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 12 & 11 & 0 & 21 & 0 & 0 & 0 & 20 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 24 & 14 & 0 & 0 & 0 & 0 & 0 & 4 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 3 & 19 & 0 & 0 & 0 & 1 & 0 & 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 21 & 7 & 0 & 11 & 0 & 0 & 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 24 & 0 & 15 & 0 & 0 & 18 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 2 & 17 & 0 & 0 & 0 & 0 & 10 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
11 & 0 & 0 & 0 & 0 & 17 & 0 & 11 & 0 & 0 & 0 & 7 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{7.7}$$

In this notation, an entry still represents a $z \times z$ sub-matrix, with $z = 24$. In case an entry is greater than zero it gives the amount of cyclic right shifts of a permuted identity matrix, with the value 1 defining an identity matrix without shift. Zero sub-matrices are indicated by the 0 entries. Note, that sometimes the -1 indicates a zero sub-matrix, then the 0 would indicate an identity matrix.

Figure 7.6 shows the plot of the full binary parity check matrix $\boldsymbol{H}$ of the same WiMAX code. The diagonals indicate where the parity check matrix is 1, in all other

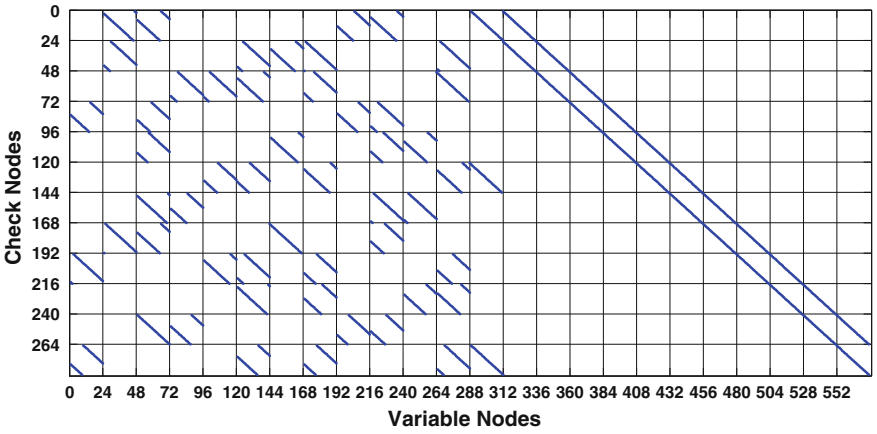


Fig. 7.6 A parity check matrix of the WiMAX code with $N = 576$, $M = 288$, $R = 1/2$, $P = 24$

cases the parity check matrix contains a zero entry. In theses WiMAX codes the variable nodes associated to information bits are located at the left of this matrix, the variable nodes of the parity bits are allocated to right. The right columns of the matrix show a typical double diagonal structure which is obtained by special structure of the encoding scheme. The relation between encoder, parity check matrix and Tanner graph is discussed in Sect. 7.3.3.

The separation of macro matrix $\boldsymbol{H}^{Macro}$ and a final $\boldsymbol{H}$ matrix realization have big advantages for the code design and analysis of the corresponding LDPC code class. A macro matrix defines the positions of permuted identity matrices and thus serves as template to derive different block sizes. This can be done by substituting different $z \times z$ sub-matrix realizations as show in Fig. 7.6. The resulting communication performance is defined by the degree distribution, in particular the convergence behavior. All analysis with respect to the macro matrix can be done independently to the detailed realization for a certain block length. Thus during LDPC code design typically a two step approach is done:

1. Designing the macro matrix:
   Designing the macro matrix concerns about the convergence behavior and the basic structure which group of z VNs is connected to which group of z CNs. The macro matrix defines thus the degree distribution and not the final Tanner graph realization. The entire analysis to define a good degree distribution is called density evolution [19]. However, it is as important which VN group is connected to which CN group which is explained in Sect. 7.3.2.
2. Determining the cyclic shift numbers:
   Assuming one macro matrix we can realize various block sizes by a different size of $z \times z$. For a fixed z we still have to derive each specific cyclic shift entry as shown for the WiMAX example. The corresponding shift entries which define the final Tanner graph can be determined by a so called progressive edge growth technique [20]. This technique starts with a Tanner graph without connection and successively places z edges by z edges always checking the resulting girth of the graph. Thus, the macro matrix will be lifted to the final size, step-by-step. Typically we start the progressive edge growth from the VN groups of smallest degree, since these nodes determine the low weight code words. For each newly placed shifted entry we test all shift possibilities and decide in favor of the shift value with the best girth or other appropriated measures [21].

Nearly all communication standards featuring LDPC codes utilize a quasi-cycle code structure, at least when a certain code rate flexibility is required. In summary, the quasi-cyclic LDPC codes have the following advantages:

- They allow for a compact description of the entries in the matrix.
- They enable a simple encoding scheme due to the quasi-cyclic property.
- The permuted identity matrix allow the use of simple barrel shifter for the hardware implementation.
- The code construction itself can be simplified, since we can clearly separate code properties defined by a macro matrix and a detailed realization of one code.

7.3.2 Hidden Nodes/Multi-Edge Type LDPC Codes

In the previous section we have highlighted the advantage of the quasi-cyclic structure. All codes presented in this section can be defined by a macro-matrix and will have a resulting quasi-cyclic structure, too. The additional feature of so called multi-edge type LDPC code is the classification of variable nodes and check nodes in groups. Typically, the connectivity structure of each group of nodes will follow rules, i.e. which group of VNs are connected to which group of CNs. The advantage of the multi-edge type classification is the larger flexibility with respect to the resulting communications performance which can be either a better convergence behavior or better minimum distance properties. These advantages can be obtained by embedding so called hidden nodes and by using variable nodes of degree one. However, then we have to take care how these group of nodes are embedded in the overall code structure.

Figure 7.7 shows a general Tanner graph of a multi-edge type LDPC code. The gray VNs are here the hidden nodes. For these group of nodes there is no initial information available, i.e. the information of these nodes are not transmitted. Hidden nodes can be seen as a kind of puncturing during the encoding process, e.g. nodes and thus states which are used during the encoding process but not transmitted. The expression hidden nodes and state nodes are used as synonyms in the following. The Tanner graph shows N_H additional variable nodes and N_H additional CNs. The problem with hidden nodes arises during the decoding process which is illustrated in Fig. 7.8 by a small tree example. The four gray variable nodes in this graph are so called hidden nodes or punctured nodes. When two messages-passed to one check node-are zero, all output messages of this CN will be zero too. Thus, the lower variable nodes will never receive an updated information. This is a typical problem of the message passing algorithm. When the algorithm get stuck and is not able to update messages any more we have identified a so called stopping set or stopping condition. Such a trapping condition could arise by a disadvantageous puncturing scheme at the transmitter side or even by chance caused by the channel.

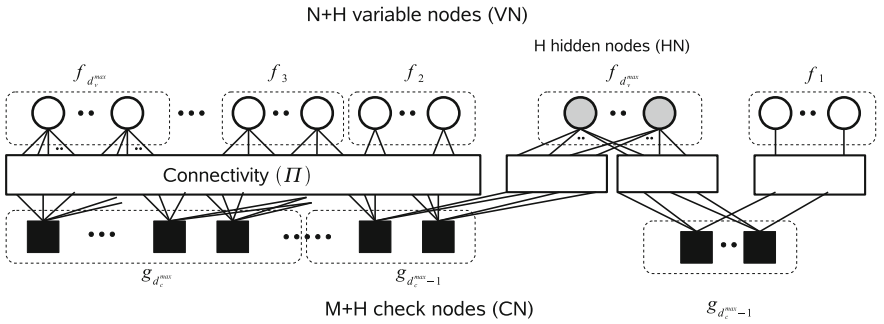


Fig. 7.7 Multi-edge type LDPC code

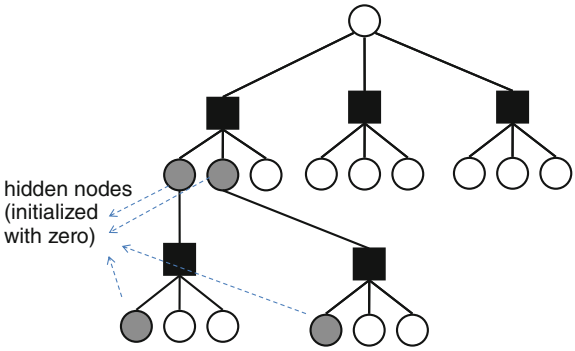


Fig. 7.8 Problem of hidden or erased nodes, i.e. problem of message passing when variable nodes are not initialized for decoding

Thus, designing LDPC codes utilizing hidden nodes require a special connectivity structure of the Tanner graph. It is not enough to specify a simple degree distribution. The Tanner graph of Fig. 7.7 describes the presence of hidden nodes but not a precise structure.

Multi-edge type LDPC codes are best described by a macro matrix. One exemplary macro matrix for a rate $R = 1/2$ code is shown in Fig. 7.9. Remember, ones in the macro matrix indicate positions of identity matrices of size $z \times z$, zeros indicate all zero matrix of size $z \times z$. The allocation of columns to the information bits, parity bits, and hidden bits is shown. Note that the hidden nodes have the highest column weight and thus node degree. Here the degree of these nodes is six. The variable nodes of degree one are only connected to the hidden nodes, this is an important aspect for using VNs of degree one. The hidden nodes themselves require an elaborated connectivity structure. Hidden nodes are initialized with a zero LLR information.

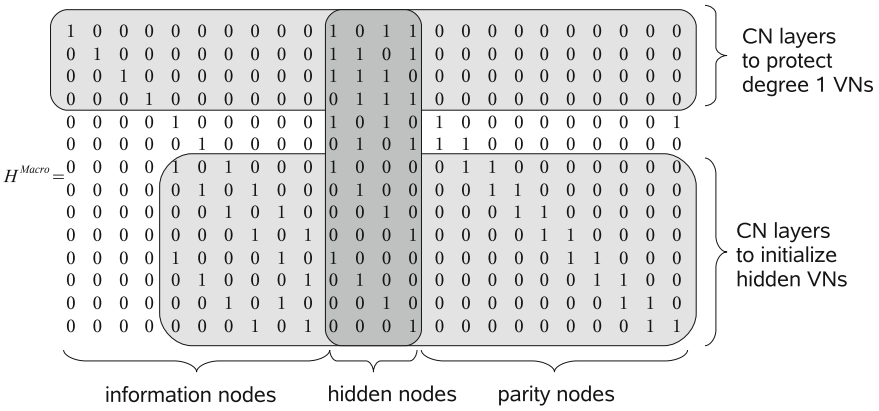


Fig. 7.9 Macro matrix of a multi-edge type LDPC code of R = 1/2

Thus, situations as shown in Fig. 7.8 should be avoided during code design. Special CN layers are utilized, which have to ensure that the hidden nodes are initialized with values. These CN layers are only connected with one entry (edge) to a corresponding VN group. The original idea of the multi-edge type LDPC codes is presented in the book of Richardson and Urbanke [22].

Figure 7.10 shows the communications performance comparison of a WiMAX LDPC code and a multi-edge type LDPC code. Both codes have $K = 1154$ information bits with a code rate of $R = 0.5$. 40 layered iterations are performed in maximum. The final performance of 20 and 40 iterations of the multi-edge type LDPC is nearly 0.3 dB superiority to that of the WiMAX code. The 5th iteration the WiMAX performance is slightly better, this is due to the hidden node initialization which has to be done in the first iteration. We need one full iteration only for initialization of the hidden nodes. The macro matrix of Fig. 7.9 was used for the lifting process, the detailed permutations of this multi-edge type code is shown in the Appendix, with further communication performance results.

7.3.3 Encoder, H Matrix, Tanner Graph

Since LDPC codes are linear block codes the encoding can be defined by multiplying the systematic information with the generator matrix G:

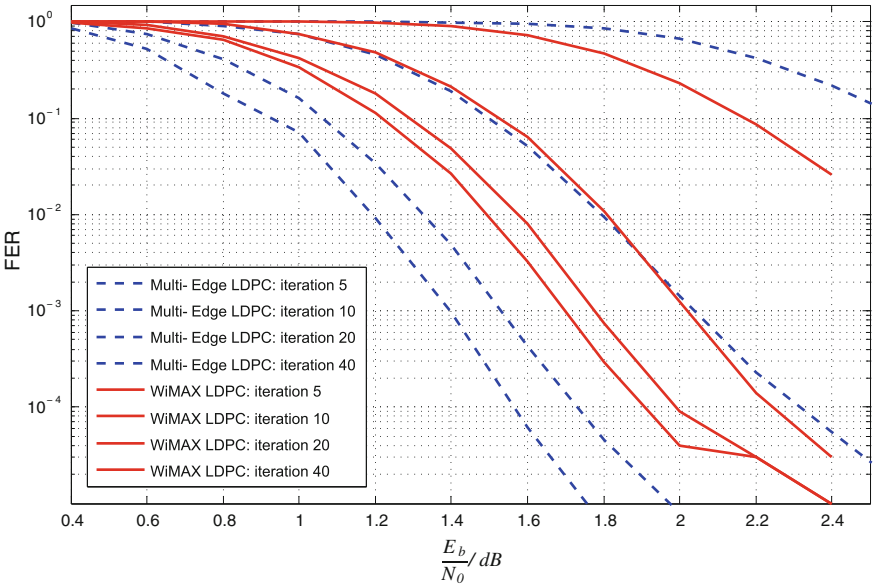


Fig. 7.10 Communications performance comparison of a multi-edge type LDPC code and a WiMAX LDPC code, both with $K = 1154$ infobits and $R = 0.5$

$$\boldsymbol{x} = \boldsymbol{u}G \tag{7.8}$$

However, encoding of LDPC codes—which are utilized in standards—is usually not performed with the generator matrix. These LDPC codes can be encoded by a low complexity encoding procedure exploiting properties of the code structure. For example, the dual diagonal on the right hand side of Fig. 7.6 allows for an efficient encoding procedure which is described in the corresponding documentation of the standards [8]. The second generation DVB codes [5–7] feature so called repeat accumulate codes. The parity check matrices of these codes show a diagonal staircase structure which enables a recursive encoding procedure.

Clear dependencies exist between parts of an encoder, parts of the resulting tanner graph and the corresponding part of the parity check matrix which is shown in Fig. 7.11. All presented encoder parts are composed of simple 2-state NSC or RSC codes and a so called repetition part. The repetition part (rep) repeats the corresponding input bits, e.g. 3 or 4 times. Depending on location of the corresponding encoder part a different connectivity structure of the check nodes exists. For example an accumulator at the output of a possible encoder structure will result in the already mentioned staircase structure of the parity check matrix. This is shown at the bottom of Fig. 7.11. The input in this example are bits which are not transmitted. These bits will correspond to hidden nodes with respect to the Tanner graph. The recursive XOR encoder structure will result in the zigzag connectivity of the check nodes.

When designing an encoder structure with an accumulator plus repetition unit located at the input we will obtain variable nodes of degree one as shown at the top of Fig. 7.11. These VNs of degree one are connected to CNs which are connected in a zigzag pattern to hidden nodes. The corresponding part of a possible parity check matrix already shows similarities to the structure of Fig. 7.9. An NSC structure at the input results in a different connectivity as shown in the middle of the figure.

In the following we describe an example of a flexible encoding scheme which is composed of the elements described above. The resulting LDPC codes are related to accumulate repeat accumulate codes (ARA) which were introduced by Divsalar [23] and later extended to proprietary products, e.g. for TrellisWare [24]. The flexibility of the encoding and the communications performance is comparable to that of turbo codes utilized in 3GPP. However, a possible LDPC decoder can be designed with a throughput that is much higher.

Figure 7.12 shows the macro matrix view together with the corresponding encoder structure. The encoder shows two different systematic input parts (sys 1 and sys 2) which have a different encoding structure and thus a different connectivity with respect to the hidden nodes. Rep 4 and rep 3/4 indicates a simple repetition of the input bit. The information part 1 will result in a degree 1 variable node which is then passed to a simple accumulator. The following 4 times repetition of these hidden bits results in a degree 6 (hidden) VN node. As mentioned before, hidden nodes should have a high variable node degree to ensure a good recovery of the information. A further important rule to design hidden nodes is the fact that at least one CN subgroup exists where only single edges are connected. This property cannot be seen at the encoder structure but when looking to the macro matrix. The initialization of the

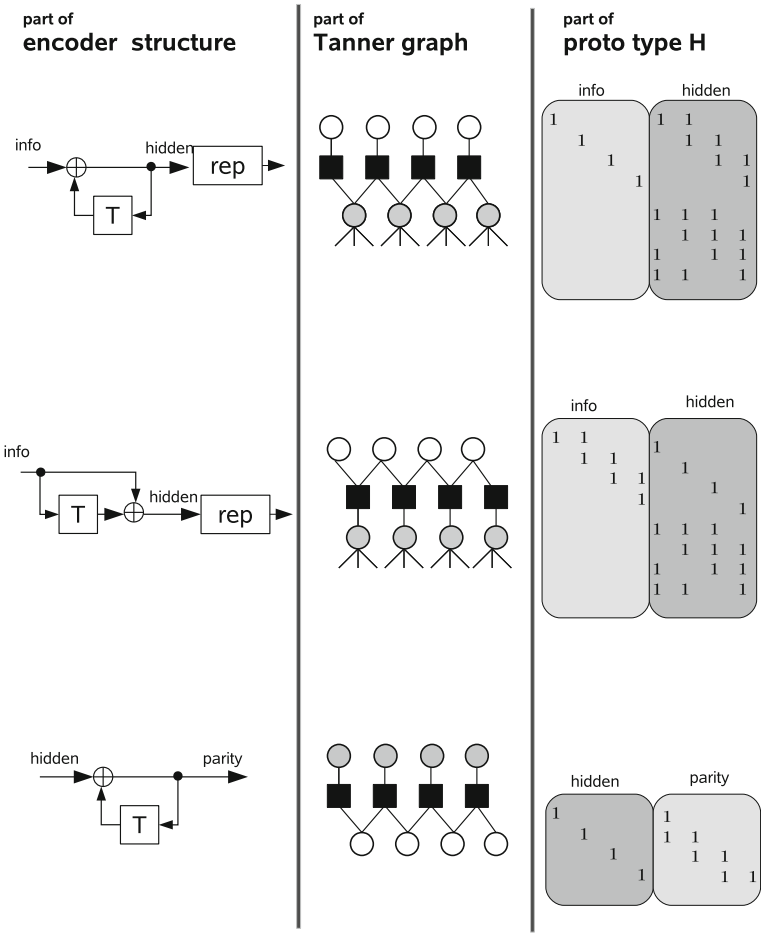


Fig. 7.11 The interrelation of encoder part, resulting Tanner graph, and PCHK structure

hidden nodes is ensured at the bottom layer of the macro matrix. The corresponding interleaver Π for the encoder is derived by the already mentioned lifting process. The encoder itself with a random interleaver would result in a very bad communications performance. The puncturing unit (punct) determines the final code rate and the resulting CN degree. The code rate can thus be adjusted by the puncturing unit or by using additional systematic (sys 2) information. The encoder structure presented here is one exemplary use case of a simple encoder which corresponds to one sub-class of multi-edge type LDPC codes. Many more interesting and very powerful structures exist which fit into the general framework of multi-edge type LDPC codes, see [22].

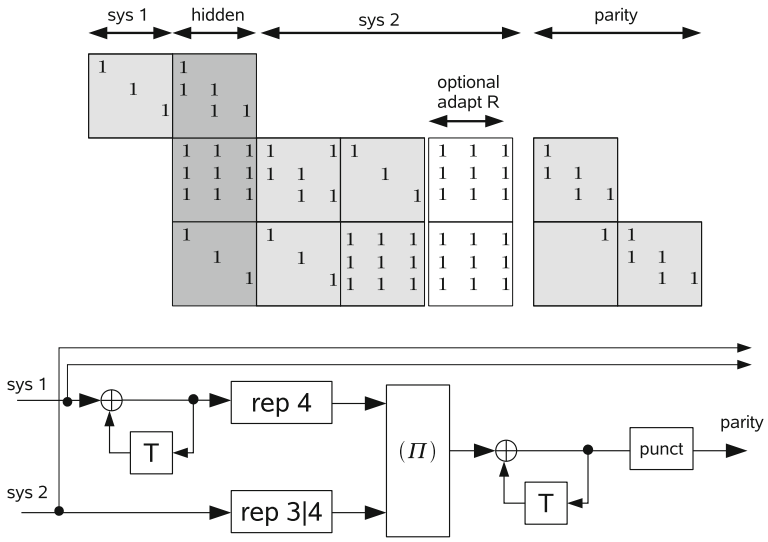


Fig. 7.12 Flexible encoder structure together with the corresponding macro matrix view, zeros within the macro matrix are not shown

7.4 LDPC Decoder Architecture

This section shows implementation aspects of LDPC decoder architectures. As already explained the decoding of LDPC codes provides an inherent parallelism. Each check node and variable node can be processed independently. This feature can be explored for different scheduling methods, see Sect. 7.2. The independence of the node processing is of great advantage for an LDPC decoder implementation. There are some general components required for all LDPC decoder architectures:

- Functional units have to provide the VN and CN processing.
- Networks support the transportation of messages with respect to the Tanner graph of the code.
- Memories are mandatory to store the received input channel LLRs and the exchanged messages within the iterative decoding process.

However, each part depends on the input constraints of the standard to be supported. Many standards require flexibility regarding block lengths and code rates while supporting a high decoding throughput. For example, an ultra wide band (UWB) compliant LDPC decoder requires a maximum throughput of 1,024 Mbit/s with a maximum codeword length of 1,200 bits. An LDPC decoder for GMR-1 requires only a throughput of less than 1Mbit/s, however, with a maximum codeword length of 64,800 bits. Communication standards which utilize LDPC codes and the corresponding data for different services can be found at www.ldpc-decoder.com. Thus, for each of these standards different solutions have to be derived. The cost or

feasibility of each solution mainly depends on the storage requirements, throughput, and flexibility of the decoder.

In the following the design space is derived together with some typical design choices (Sect. 7.4.1). Possible solutions are identified for the message distribution, the message storage, and the processing units. Different implementation possibilities for CN processing are presented in Sect. 7.4.2. The already presented quasi-cyclic LDPC codes provide a regularity which can be efficiently exploited during decoder realization. The mapping process to hardware for quasi-cyclic LDPC code is presented in Sect. 7.4.3.

7.4.1 Design Space and Design Choices

The design space which reflects the possibilities of LDPC decoder implementations can be divided into several levels. An enhanced list of design possibilities can be found in [13]. From a high level point of view mainly the number of processed edges per clock cycle defines the basic architecture. Then, only three basic possibilities exist on a very high level:

- Fully parallel decoder: all edges are processed in one clock cycles.
- Partly parallel decoder: P edges are processed in one clock cycle.
- Serial decoder architecture: one functional node is processed per clock cycle. This may result in a varying, however, small number of processed edges.

Depending on these basic techniques different solutions exist for the message distribution, the message storage, and the realization of the processing units. The three basic architectural decisions are presented in the following.

Fully Parallel LDPC Decoder

In a fully parallel LDPC decoder realization the Tanner graph is directly mapped to hardware. Each variable and check node is instantiated and the connections are realized hard-wired. This was shown in [25] for a $R = 1/2$ irregular LDPC code with a codeword length of $N = 1,024$ bits. They used a message quantization of 4 bits while the messages from variable to check nodes and the returning are carried on distinct sets of wires. The data path of a fully parallel architecture is shown in Fig. 7.13. The figure shows one functional node which processes a variable node of degree 3, the functional node processes a check node of degree 4, respectively. It can be seen that both types of functional nodes are implemented in parallel manner and accepts all mandatory input messages. With this data path we need 1 clock cycle for one iteration. The critical path of such an architecture is pretty long, since we have to go through the CN and the VN functional node.

The resulting throughput of the decoder in [25] is 1 Gbit/s assuming 64 iterations. The fully parallel implementation is the fasted possible implementation since all

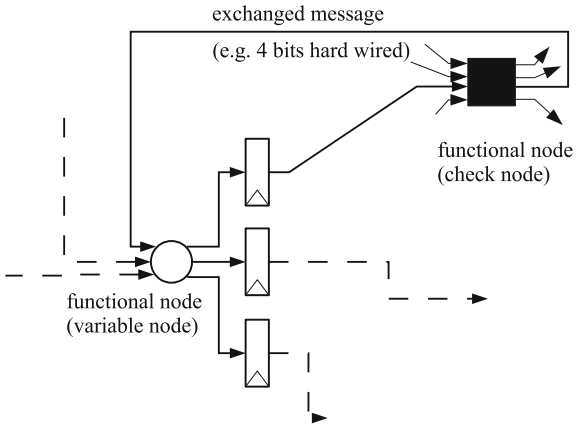


Fig. 7.13 Data path architecture for a fully parallel realization, the figure is derived from [25]

nodes and thus all edges are processed concurrently. Thus, the advantage of a fully parallel implementation is the high throughput. Especially in optical transmission systems this is of great interest where one fixed LDPC code with a high code rate ($R > 0.8$) could be established. The major problem for the hardware realization is the routing complexity which becomes prohibitive for a large codeword length due to routing congestions. Furthermore, this approach is only suitable for applications where only one fixed code is used.

Partly Parallel LDPC Decoder

A partly parallel LDPC decoder processes P edges in one clock cycle, while P is much smaller than the block length N. Often a partly parallel decoder becomes mandatory to provide flexibility, at which only a subset of functional nodes are instantiated. The variable nodes and check nodes are then processed sequentially on these processing units. A processing node can be instantiated to process one edge per clock cycle or multiple edges (degree dependent) per clock cycle. Hence, the number of processed edges P and the number of instantiated functional nodes may differ. The throughput is only determined by the number of edges (P) processed per clock cycle. The major question for the mapping process is the allocation of a node in the Tanner graph to a physically realized hardware unit. Thus, the question has to be answered: which node has to be processed on which unit at which time? One mapping procedure for a partly parallel architecture is shown in Fig. 7.14. In this figure it is assumed that one functional unit processes one edge per clock cycle.

The messages which are exchanged between the functional units processing variable nodes (VNFU) and check nodes (VNCN) have to be stored in memories. A permutation network has to implement the desired connectivity of the given Tanner graph, while the connectivity pattern has to be stored in additional memories. The big advantage of a partly parallel architecture is the possibility to provide

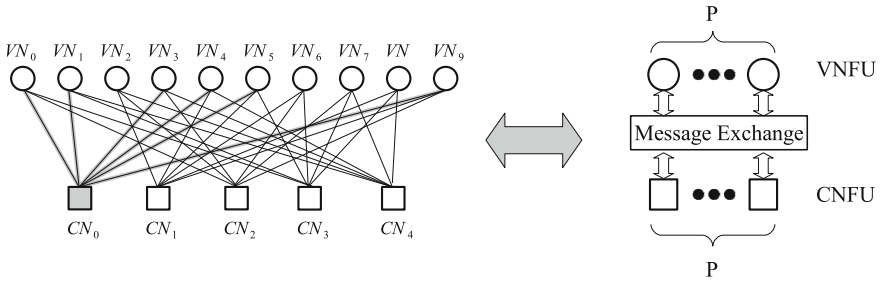


Fig. 7.14 Mapping from Tanner graph to parallel LDPC decoder architecture

flexibility, allowing different block lengths and code rates. The disadvantage is the limited throughput which is smaller compared to a fully parallel decoder implementation.

Serial LDPC Decoder

The serial LDPC decoder is the simplest implementation. Only one functional variable node and one functional check node are instantiated with a sequential processing of all variable nodes and parity check nodes. The processed messages and the code structure have to be stored in memories. The advantage of such an architecture is the straight forward message distribution. A further advantage is its flexibility, the disadvantage the very low throughput.

In [26] a serial decoder architecture is presented. They suggest using several instantiated decoders in parallel to increase the throughput, but this results in a huge decoding latency and large memory requirements. Instantiating several LDPC decoders is always possible. The overall area increases linearly with the number of decoder instantiation, however, this solution is not preferable if latency is a critical issue.

In the following only partly parallel architectures are discussed since they can provide flexibility in contrast to a fully parallel realization and a higher throughput than a serial LDPC decoder implementation. The question of flexibility determines the implementation style of functional nodes which are presented in the following section.

7.4.2 Check Node Implementation

Depending on the basic architecture parallelism we have to instantiate either parallel functional nodes, partially parallel or serial functional nodes. The hardware realization of the variable node processing is straight forward and not further addressed.

As seen in the design space different check node implementations are mandatory which may result in a parallel, serial, or partially parallel CN architecture. Only two architectures are derived here. The fully parallel CN implementation and the serial CN implementation.

Fully Parallel CN Implementation

For a fully parallel hardware realization a CN implementation becomes mandatory which processes all incoming messages within one clock cycle. This is shown in Fig. 7.13. For realizing such a parallel functional unit the tanh equation of the CN can be utilized, which is:

$$\tanh\left(\frac{\lambda_l}{2}\right) = \prod_{i=0, i\neq l}^{k-1} \tanh\left(\frac{\lambda_i}{2}\right). \tag{7.9}$$

This check node calculation can be divided in two steps. Step one is the calculation of an intermediate result I:

$$I = \prod_{i=0}^{k-1} \tanh\left(\frac{\lambda_i}{2}\right). \tag{7.10}$$

I comprises the full tanh product of all incoming messages. For the output calculation a second processing step is required. For each output each l we have to evaluate:

$$\lambda_l = 2 \times \operatorname{arctanh}\left(\frac{I}{\tanh\left(\frac{\lambda_l^{old}}{2}\right)}\right). \tag{7.11}$$

During hardware realization we should avoid divisions for complexity issues. Thus, the two processing steps are transformed to the logarithm domain.

One problem arises during the CN processing in the logarithm domain. The treatment of the sign posses a problem and can be solved by splitting the CN calculation in a magnitude processing and a separate sign processing. The sign processing performs a parity check (XOR) calculation of the signs of the input messages and checks if a parity check is fulfilled. The output sign of a corresponding edge is the XOR of the sign of the incoming message and the parity check result. For a detailed architecture see [25].

In the following only the processing of the magnitudes is shown, see Fig. 7.15. In the figure an input quantization of 4 bits is assumed (only magnitude part), while the intermediate bit width is larger. For an efficient VLSI realization, each data bit width of each intermediate result has to be optimized with respect to an adequate communications performance. For example, the adder which sums up 3 values, each value quantized with 7 bits, would have in worst case a 9 bit output value. However,

Step one calculate intermediate result I (full sum):

$$I = \sum_{i=0,}^{k-1} \ln\left(\tanh\left(\frac{\lambda_i}{2}\right)\right).$$

Step two calculate output result for each output k:

$$\lambda_l = 2 \cdot \operatorname{arctanh}\left(\exp\left(I - \ln\left(\tanh\left(\frac{\lambda_l^{old}}{2}\right)\right)\right)\right)$$

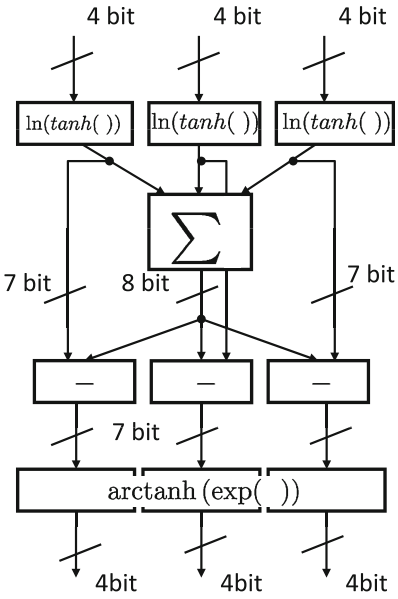


Fig. 7.15 Data path of parallel CN architecture with three edges. Here, only the magnitude part is processed, the sign bits are processed in a separate flow, see [25]

we can saturate the sum to 8 bits without any influence on the resulting communications performance.

For the data path of Fig. 7.15 we have to implement the function $ln(tanh(x))$. Since the input bit width is quite small we can realize the function in hardware by a look-up table. A look-up table stores pre-computed values for each possible input combination. Deriving such a look-up table was shown in Sect. 5.2.2.

Serial CN Implementation

A serial CN implementation is often used for partly parallel decoder implementations. Here, we do not implement a serial CN utilizing the tanh realization of the previous section. The reason for that is the larger dynamics for supporting different code rates. In the case of fully parallel implementation the look-up tables can be efficiently derived since only one code rate and thus a limited SNR range is used with the architecture. As mentioned the serial CN implementation has to provide a code rate flexibility which ranges for the second generation DVB class from a CN degree of 4 to 30.

A generic architecture of a serial CN is shown in Fig. 7.16. Only one input edge is processed per clock cycle. Most of the serial CN architectures are based on minimum searches with correction at the output stage or without correction term. All grey boxes and the dotted lines in Fig. 7.16 are optional and only mandatory when we apply a

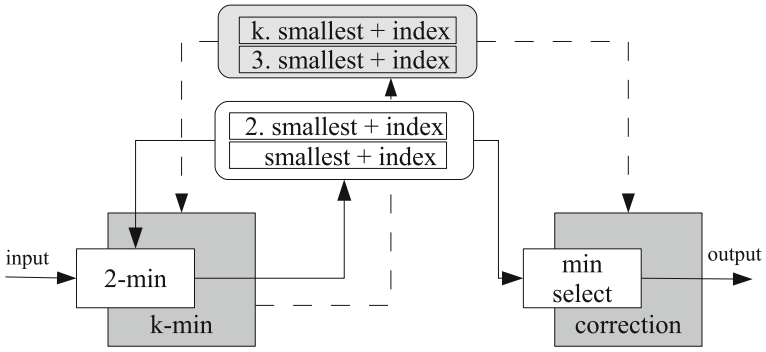


Fig. 7.16 Generic serial CN architecture based on searching the minimum. The *white boxes* are part of the Min-Sum realization, the *gray parts* are optional for a better CN functionality approximation

correction term to approximate the optimal output processing. The entire processing can be split in an input processing and an output processing. The input stage has the task to search the minimum of all input edges and to store the corresponding index. Depending on the output approximation we need only the two smallest values, which would result in the so called Min-Sum algorithm where the minimum value is the result for all output messages. However, we have to ensure not to pass back the minimum to its origin. Thus, for this message we select the second smallest value as output result. Correction terms to approximate the correct result can be derived from further values, which is then often denoted as $k - Min$ approximation. The value k indicates how many values are utilized for the approximation. An approximation function which can be efficiently realized within this architecture was already shown in Sect. 5.2.3.

7.4.3 Mapping from Parity Check Matrix to Hardware

During the hardware design of the first LDPC decoders it turned out that a parity check matrix with random entries will result in memory access problems. These memory access problems are identical to that discussed for parallel interleaving, Sect. 4.3. However, it is possible to design an LDPC code which jointly takes into account the communications constraints and hardware constraints which is denoted as joint code-decoder design in the following. The resulting codes are quasi-cyclic which were introduced in Sect. 7.5. As mentioned, many LDPC codes defined in wireless communication standards are quasi-cyclic, i.e. the parity check matrix is composed of permuted identity matrices.

Figure 7.17 shows the first step of the joint code-decoder design. The joint code-decoder design was already shown in Sect. 4.3 to design interleavers which can be implemented without memory access conflicts. An identical design philosophy is

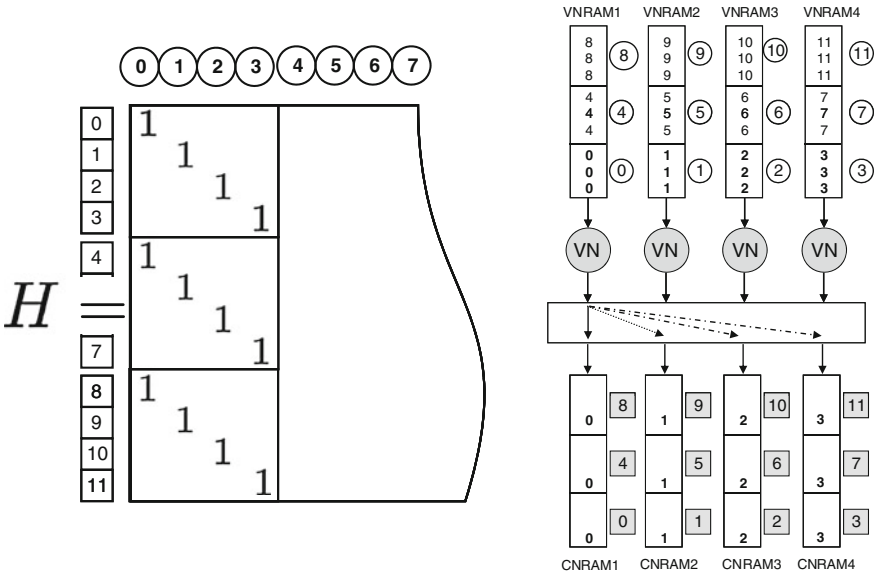


Fig. 7.17 Mapping step of the first three permuted identity matrices to hardware

shown in this section. Starting point is an empty parity check matrix and an architecture hardware template for the decoder. On the right side we have an assumption of four functional nodes with a serial processing of the variable nodes. Thus, each functional VN can accept exactly one message per clock cycle. Four different memories are allocated to the input. The output of the VN processing elements is connected to a shifting network. This concept of functional processing node and afterwards a permutation network was already derived in the interleaver chapter. Again four different memories are assumed for storage which results in a maximum of four processed edges per clock cycle. In this small example we would like to design a regular LDPC code with variable nodes of degree three. Thus we have to reserve three memory spaces for the messages of each variable node. The numbering inside the VN RAMs indicates the three messages which will be allocated to the first 12 variable nodes, labeled from 0 to 11.

On the left side of Fig. 7.17 part of a parity check matrix is shown with a possible allocation of columns to variable and check nodes. Here the first column (VN) is allocated to the first variable node, the second column to the second VN and so on. For the check nodes the same allocation holds, first row is allocated to the first CN, indicated with the square with the zero entry.

We have started to fill the parity check matrix with entries. Here three identity matrices are already placed which shows directly which variable node is connected to which check node. On the hardware side this may result in the shown memory space allocation. The three messages of variable node 0 are stored at three distinct

addresses in the CN RAM1. The same storage location exists for all four processed messages.

Figure 7.18 shows the mapping of the next two identity matrices of a parity check matrix. One identity matrix has already one permutation. This permutation was mandatory to avoid short cycles which hampers the convergence of the utilized message passing algorithm. The resulting messages in the architecture template are changing the storage location to ensure the correct connectivity between VNs and CNs. Messages are cyclic shifted to right, which is exactly the functionality a shifting network can provide. The entire procedure to fill the parity check matrix with P messages by P messages was already introduced in the context of designing structured codes, see Sect. 7.3. The mapping of the parity check matrix to an architecture template as shown in Figs. 7.17 and 7.18 is one possibility to utilize a joint code-decoder design. Many papers exist which utilize the joint code-decoder design technique, like [27–29]. Sometimes serial processing units are assumed, and sometimes parallel processing units. However, all publications have one basic part in common: all utilize a permutation network which allows a parallel message distribution without memory access conflicts.

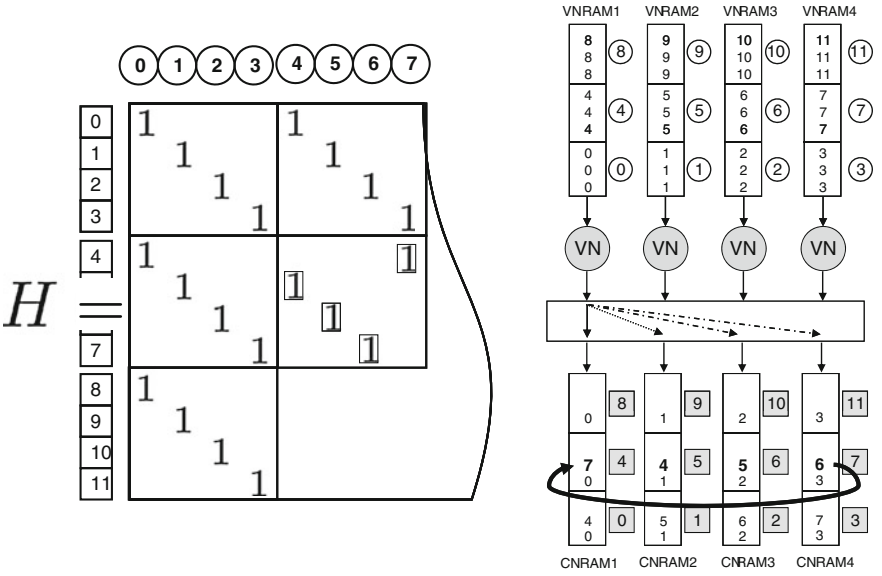


Fig. 7.18 Mapping step of further permuted identity matrices to hardware

7.4.4 Advanced Topics for LDPC Decoder Realization

The hardware realization of LDPC decoders is already quite elaborated. Efficient LDPC decoder designs for various standards can be found in [30–35]. The list is far from being complete. As mentioned before, we have to solve three basic problems: the realization of the computational units, the exchange of messages between the computational units and the storage of the messages. In the following a very compact review list for each of the topics is given.

Iterative Scheduling Method

For the LDPC decoder hardware realization we have to utilize an iterative scheduling method, as described in Sect. 7.2: either a two-phase scheduling or a layered scheduling.

- The two-phase scheduling updates first the VNs and then the CNs.
  - Pro: It is the most simple scheduling method and can always be implemented. The two-phase scheduling method is mandatory for a fully parallel decoder realization.
  - Con: The convergence speed is slower than that of a layered processing.
- The layered scheduling (horizontal) continuously updates the a posteriori information of the VNs after processing one or a group of CNs. This scheduling method utilized for a hardware realization was first presented by M. Mansour [28].
  - Pro: The convergence speed is faster than the two-phase scheduling. For a hardware realization this can be of great advantage since for a limited number of iterations a better communications performance can be achieved. Whenever possible this scheduling method should be utilized for a hardware realization, see [13, 33, 36].
  - Con: For hardware realization the layered update scheme can cause memory access problems which is sometimes difficult to resolve, e.g., in the case of DVB-S2 processing [34].
- The layered scheduling (vertically) continuously updates the information of the CNs after processing one or a group of VNs. This scheduling method utilized for hardware realization was first presented in [37]
  - Pro: The convergence speed is faster compared to that of a two-phase scheduling. Especially for very high throughputs and a partly parallel decoder design this solution can have advantages, see [38]
  - Con: The scheduling method has many restrictions for a joint code-decoder design approach. Realizing a layered scheduling for high code rates is difficult. The higher the code rate, the less is the advantage of a faster convergence compared to a two-phase scheduling.

Quantization Issues

For the hardware realization we have to quantize the input information and the messages which are exchanged in the iterative process. Often the input quantization and the quantization of the exchange messages are chosen to be identical.

- The input LLRs and the exchanged informations are typically quantized with 6 bits. The quantization directly influences the overall area of a chip.
  - Pro: 6 bits results in a robust communications performance for wireless channels. Especially for wireless communication different applications and thus code rates and block sizes have to be supported. Thus, channel conditions will vary. 6 bits are enough to obtain a communications performance with an acceptable loss compared to a floating point implementation. The loss is smaller than 0.2 dB for all cases. Fixed point explorations can be found in [39, 40].
  - Con: When only one (high) code rate has to be supported, we can obtain very good communications performance results with 4 input bits. 6 bits message quantization would be a high routing overhead for a fully parallel decoder realizations.
- The messages can also be encoded by stochastic messages. Many bits are used to encode a message, while the statistics of ones and zeros within a message determines the corresponding confidence level of the message. These Bernoulli sequences are passed bit by bit to the corresponding processing units. An entire decoder with these messages is presented in [41].
  - Pro: Stochastic messages allows a high clock frequency of the design since the arithmetic units for VN and CN processing are very simple. Furthermore it has advantages for the routing overhead due to bit serial message exchange.
  - Con: The approach itself comes with a small performance degradation and makes only sense for a fully parallel approach.

Sub-optimal CN Processing

For LDPC decoding only the processing of the check node functionality has different solutions [39, 40, 42]. The variable node processing is trivial and thus not considered here. The CN functional unit can either compute a sub-optimal solution or an optimal CN calculation. Note, that in all cases the optimal sign value is calculated which indicates if the parity check condition is fulfilled. Only the calculation of the corresponding message reliability differs.

- The most common sub-optimal realization is the Min-Sum realization. The CN functional unit searches the smallest and second smallest magnitudes of all input messages and passes only two different magnitude values back to the VNs.
  - Pro: The realization of the CN functional unit will result in a very small area and the communications performance is good for high code rates.

  - Con: The smaller the code rate, the worse will be the Min approximation. For smaller code rates the performance degradation is quite large compared to an optimal realization. E.g., DVB-S2 features rates of $R = 1/5$. With a simple Min-Sum approximation the required communications performance of the standard can not be achieved.

- The direct $tanh$ realization results in an implementation close to the optimum. The optimality depends of course on the quantization level.
  - Pro: The $tanh$ realization enables a full parallel processing of a check node and is nearly optimal from a communications point of view.
  - Con: The $tanh$ realization is relatively unstable for SNR mismatches, i.e. if the input LLR values are not correctly scaled with respect to the noise variance. This kind of CN realization should not be implemented for decoders featuring wireless communication standards. The non-linear functions are realized by look-up tables which may increase fast for larger bit width

- Many different sub-optimal variants which show a communications performance between an optimal CN computation and the Min-Sum realization. One of the best trade-off possibilities is the so called λ-Min CN realization. For the APP calculation of CN messages only a reduced number of messages are considered for calculating a correction factor [43]
  - Pro: This sub-optimal variant offers a good communications performance and it is adjustable with a decent implementation complexity.
  - Con: The technique is problematic for fully parallel CN processing.

Realizing the VN to CN Connections

For the iterative message exchange we have to provide the connectivity pattern, i.e., the routing between physically instantiated VNs and CNs. Different solutions exist with respect to the required throughput and flexibility.

- Hardwired: the connectivity can be hardwired which makes sense for extremely high throughputs [25].
  - Pro: This results in a one-to-one mapping of the underlying Tanner graph. Hard-wired is one possible solution to realize the 802.3an standard (10 GBASE-T) [10].
  - Con: Routing can be complex and the decoding is restricted to one (or few) dedicated LDPC codes.

- Shuffling networks: these are simple networks which can realize for P input and P output ports exactly P permutation possibilities. These are always cyclic shifts of the input sequence.
  - Pro: Shuffling networks are simple to realize in hardware. The connectivity pattern of quasi-cyclic LDPC codes can be efficiently mapped on these networks.

  - Con: Shuffling networks are restricted to quasi-cyclic LDPC codes, e.g. the connectivity pattern of 802.3an [10] could not be realized with these networks.

- Benes networks: these are networks which can realize exactly $P!$ permutation possibilities for P input and P output ports. The network can realize all conflict free permutation possibilities, which means P input messages can be routed to P distinct output ports.
  - Pro: Benes networks provide a huge flexibility and are often utilized for flexible solutions.
  - Con: The storage amount of the control bits can be very large. $\frac{P}{2} \times (2 \times \log_2(P) - 1)$ control bits are mandatory for one permutation of P messages. Since at each clock cycle a new permutation pattern is required the total storage demand is often higher than the storage of the exchanged messages.

- Network-on-chip solutions: NoCs uses a kind of packet based solution [44–46]. Each packet is associated with a header to identify a destination port.
  - Pro: In theory any connectivity pattern can be realized, demonstrators exist to prove the theoretical capability for a fully flexible LDPC decoder.
  - Con: For providing only the connectivity pattern of an LDPC decoder this solution is cumbersome and often quite large. We are not aware of a full NoC approach for an LDPC decoder product.

Storage of the Exchanged Messages

For packet based transmission every LDPC decoder implementation has to store the input LLR information. Besides the quantization level, the optimization possibilities for the input storages is quite restricted. Most LDPC decoders for wireless transceivers are realized by utilizing a partly parallel architecture. For these architecture different optimization possibilities exist to store the exchanged messages.

- Edge compression: it is possible to store a compressed versions of the exchanged messages. Edge compression is only possible together with sub-optimal CN realizations at which the number of different magnitude values are limited. E.g. for the Min-Sum realization only two different magnitude values are considered.
  - Pro: The higher the CN degree the higher the achieved compression level and thus the storage benefit. This results in power savings, due to less memory accesses compared to architectures without compression.
  - Con: For lower code rates and thus small CN degrees, the compression overhead is getting larger and has to be switched off.

7.5 Joint Code Construction and Architecture Design

This section shows a design study for a high throughput LDPC decoder. It highlights again the fact that a joint know-how of architecture design and code construction is mandatory to obtain an efficient LDPC decoder design.

In the previous sections we have shown the constraints to map a parity check matrix to a decoder architecture, see Sect. 7.4.3. The presented code to architecture mapping results in standard compliant LDPC decoders which support the processing of up to P parity checks in parallel which belong to the same sub-matrix of size $z \times z$.

In this case, the check nodes work serially, i.e. on one edge per clock cycle. This was shown for the hardware mapping assuming $P = z$, see Fig. 7.18. The one-to-one mapping of sub-matrices to serial processing units results in a decoder throughput which is sufficient for WiFi [9] , WiMAX [8] or DVB-S2/T2/C2 [5–7]. This approach is described in many publications e.g. [30, 31, 34]. However, future communications systems will require a higher throughput while providing flexibility. Examples are Utra Wide-Band (UWB) communication systems [11], or systems operating in the 60GHz range [12]. A further application with a large throughput demand will be iterative BICM systems, throughput considerations for feedback loops via MIMO receivers will be presented in Sect. 8.2.

This section shows an advanced LDPC decoder design study which tackles the upcoming high throughput demands, parts of this sections are presented in [47]. One key feature of a new LDPC decoder is always the support of quasi-cycle LDPC codes which was highlighted in Sect. 7.3.1. Further key points which have to be considered for the LDPC code and architecture design are summarized in the following.

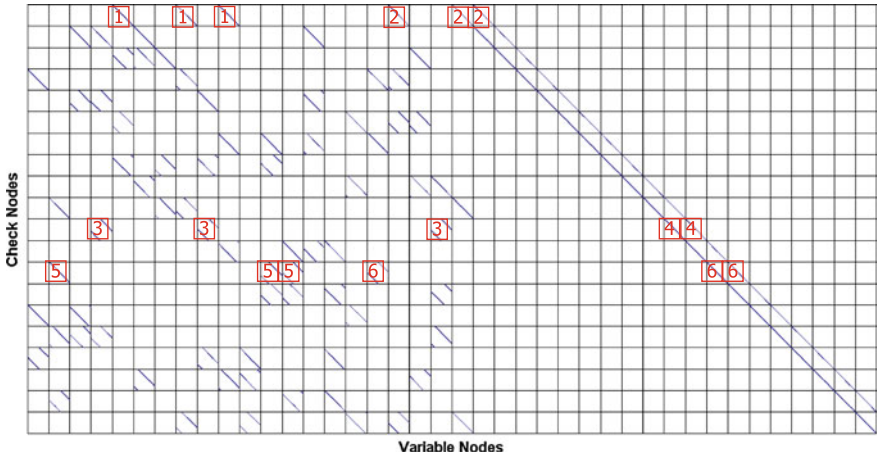


Fig. 7.19 A parity check matrix which enables layered decoding with a three slot architecture. The numbers indicate the clock cycles at which the corresponding sub-matrices will be processed

- Flexibility: The decoder architecture should support block sizes ranging from $N = 1{,}000$ bits to several thousands of bits while supporting various code rates.
- Throughput: The decoder has to provide a throughput within the Gbit/s range.
- Layered decoding: The decoder should support layered decoding to obtain the best communications performance result at each iteration.

All three constraints heavily influence the design of the parity check matrix of the LDPC code and the final decoder architecture. Fulfilling the three constraints as good as possible at the same time can only be achieved by joint channel code construction and architecture design.

As highlighted in Sect. 7.4.1, numerous parallel check nodes can be instantiated in an LDPC decoder to increase its throughput. The memory access patterns to supply all check nodes with data are crucial for the code design to avoid access conflicts. A quasi-cyclic code with sub-matrices of size $z \times z$ can avoid memory conflicts and simplify the connectivity network between variable nodes and check nodes.

Meeting the target throughput in the Gbit/s range on a standard decoder architecture with $P = z$ would require a large size for the identity matrices. However, this contradicts with the given constraints on flexibility and layered decoding: For a small block length, an assumption of $P = z$ results in sub-matrices with multiple diagonals. These multi-diagonals lead to severe memory access problems for layered decoding architectures which are well-known from literature [33, 48]. Thus, increasing the sub-matrix size z in order to reach the required parallelism on a standard architecture is infeasible. Instead, we need to process several sub-matrices in parallel.

In the following a design approach is presented at which 3 sub-matrices can be processed in parallel, resulting in $P = 3 \times z$. The resulting architecture is based on the concepts presented in [36] and shown in Fig. 7.20. Each set of z Variable Nodes is attributed to one of the three slots. One slot processes one sub-matrix per clock cycle. In contrast to standard architectures, each of the z check node blocks (CNB) reads and writes three edges in parallel, with each edge coming from a different slot. The minimum search within the CFU is performed according to [31]. The variable node operations are processed within the CNBs and do not require an explicit functional unit. This architecture fulfills the constraints on flexibility, throughput and layered decoding.

However, the architecture imposes hard restrictions on the parity check matrix. In layered decoding, variable nodes are partly updated as soon as a connected check node is processed. Processing is done in a pipeline with three stages including one cycle from APP RAM through Network and CNB and back to the APP RAM. Writing back the data can only be started after the whole check node has been processed. A variable node can only be accessed again after being updated and is thus blocked for several clock cycles. The length of the blocking interval B does not only depend on the length of the pipeline but also on the check node degree d_{CN}.

$$B = 3 + \left\lceil \frac{d_{CN}}{3} \right\rceil \tag{7.12}$$

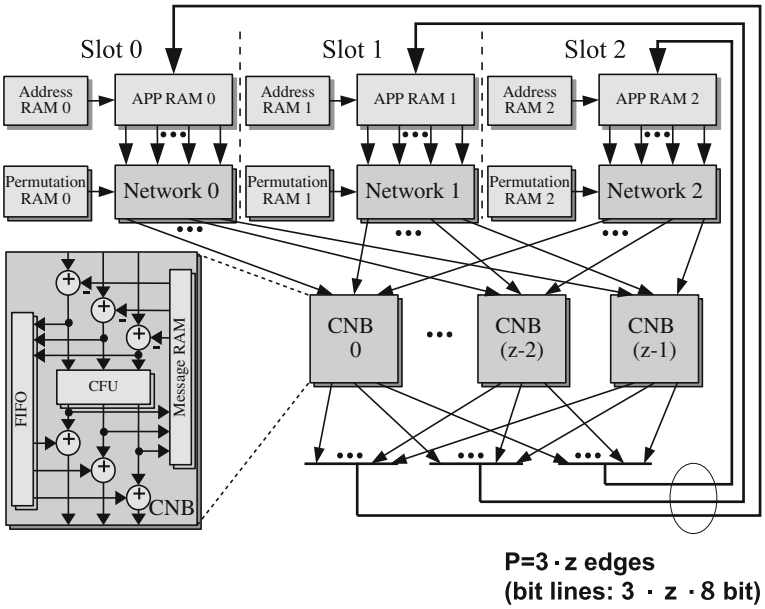


Fig. 7.20 Layered decoder architecture with partly parallel check nodes processing three sub-matrices in parallel each of size $z \times z$. The architecture can process ($P = 3 \times z$) edges per clock cycle

The challenge of the code design lies in the placement of non-zero sub-matrices. Continuous processing has to be ensured which is not delayed by blocked variable node columns, i.e. there always has to be a row of sub-matrices available which does not contain any of the currently blocked variable node columns.

For example, consider Fig. 7.6: We start by processing the first group of z check nodes (the first z rows of the parity check matrix). In the first clock cycle the three sub-matrices labeled with '1' are processed in parallel. Next, the sub-matrices '2' are processed and all values for the first z parity checks have been read. But only 3 clock cycles later the results of the sub-matrices '1' will be written back. Thus, all variable node columns containing '1' are blocked until clock cycle 6, all columns containing '2' until clock cycle 7. When we select the next group of check nodes, we have to find a row of sub-matrices which does not contain any blocked columns.

The layered constraint mainly influences the placement of identity matrices and poses an obstacle especially for high code rates and/or small block sizes. For the design study we chose z to be $z = 93$ due to the derived throughput constraints in [47], see as well Sect. 8.2. The architecture can process $P = 3 \times z = 279$ edges in parallel. After deciding for a detailed size of z we can lift the identity matrix as described in Sect. 7.3.1.

Constructing a structured LDPC code for layered decoding and a sub-matrix size of $z = 93$ allows for a minimum block length of $N = 3{,}720$. Furthermore, the

maximum variable node degree for the shortest blocks is restricted to $VN_{max} = 5$. The block size and variable node degree restrictions are due to the layered constraint as described above. The larger the block the more freedom to place sub-matrices, i.e. the size of the macro matrix will be changed. Starting from a block size of 7,440 bits a maximum VN degree of $VN_{max} = 15$ can be supported.

The final implemented LDPC decoder architecture supports block sizes between 3,720 and 14,880 bits with code rates between $1/2$ and $4/5$. The maximum block size is only restricted due the used sizes of the SRAMs.

The LDPC decoder is implemented on a 65 nm low power bulk CMOS library. We considered the following PVT parameters: Worst Case (WC, 1.1V, 125 °C), Nominal Case (NOM, 1.2V, 25 °C) and Best Case (BC, 1.3V, −40 °C). Synthesis was performed with Synopsis Design Compiler in topographical mode, P&R with Synopsys IC Compiler.

Synthesis as well as P&R were performed with worst sase PVT settings of the 65 nm library. PVT defines the process, voltage, and temparature as decribid for SRAM memories in Sect. 4.2. Table 7.3 shows the results for area and maximum clock frequency. The differences between Synthesis (2.932 mm^2, 322 MHz) and P&R (4.588 mm^2, 275 MHz) mainly result from the huge internal bandwidth inside the LDPC decoder. Each message is quantized with 8 bits, thus we have to route $z \times 3 \times 8 = 2,232$ wires. This results in routing congestion problems due to the huge number of wire connections between memory blocks and logic. For Nominal Case, the maximum clock frequency increases to 465 MHz.

The final physical design shown in Fig. 7.21 contains 59 SRAMs which occupy more than 40 % of the area (2.244 mm^2). An area utilization (cell/core area ratio) of 67 % was achieved.

The power consumption of the physical design is evaluated with Synopsys PrimeTime-PX. Table 7.4 summarizes the results. At a low frequency of 100 MHz, the implemented design consumes 481 mW. The power consumption scales linearly with the clock frequency.

The LDPC decoder has a net throughput of 4.6 Gbit/s assuming five layered decoding iterations. Depending on the code rate the information bit throughput will be smaller. The evaluation of this result and the comparison with other channel decoder realizations is done in Chap. 9.

Table 7.3 Synthesis and P&R results

Design step	Case	Max. clock freq. (MHz)	Area (mm^2)
Synthesis	Worst	322	2.932
	Worst	275	
P&R	Nominal	465	4.588

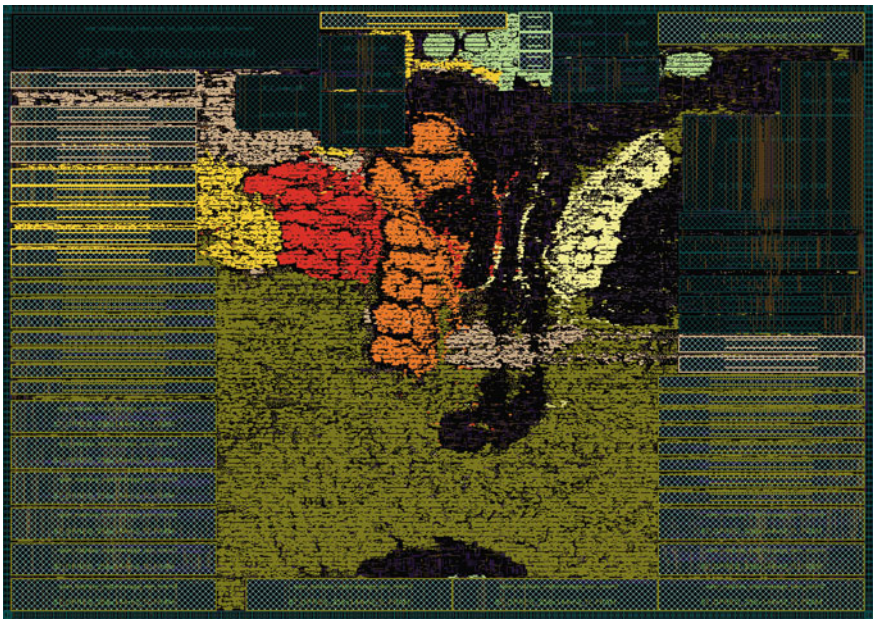


Fig. 7.21 Physical design of the LDPC decoder. The barrel-shifter networks are in the *top middle* (0: light yellow, 1: orange, 2: red). The processing unit containing the 93 check node blocks covers the lower half of the chip (olive)

Table 7.4 Power consumption of the implemented LDPC decoder after P&R

Clock freq. (MHz)	Process case	Power after P&R (mW)
100	Nominal ($V_{dd} = 1.2V$)	481
265	Nominal ($V_{dd} = 1.2V$)	1,271
265	Worst ($V_{dd} = 1.1V$)	1,070
265	Best ($V_{dd} = 1.3V$)	1,378

References

1. Gallager, R.G.: Low-Density Parity-Check Codes. M.I.T. Press, Cambridge (1963)
2. Elias, P.: Error-free coding. IEEE Trans. Inf. Theory **4**(4), 29–39 (1954)
3. Tanner, R.M.: A Recursive approach to low complexity codes. IEEE Trans. Inf. Theory IT-27, 533–547 (1981)
4. MacKay, D., Neal, R.: Near shannon limit performance of low-density parity-check codes. Electron. Lett. **32**, 1645–1646 (1996)
5. European Telecommunications Standards Institude (ETSI): Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2), ETSI EN 302 755 V1.1.1. www.dvb.org

6. European Telecommunications Standards Institude (ETSI): Digital Video Broadcasting (DVB) Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; EN 302 307 V1.1.2. www.dvb.org
7. European Telecommunications Standards Institude (ETSI): Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2); ETSI EN 302 769 V0.9.9. www.dvb.org
8. IEEE 802.16e: Air interface for fixed and mobile broadband wireless access systems. IEEE P802.16e/D12 Draft (2005)
9. IEEE 802.11n: Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput. IEEE P802.11n/D3.0 (2007)
10. IEEE 802.3: LAN/MAN CSMA/CD Access Method. http://standards.ieee.org/getieee802/802.3.html
11. WiMedia Alliance: Multiband OFDM Physical Layer Specification, Release Candidate 1.5 (2009)
12. IEEE 802.15.3c: Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs). IEEE 802.15.3c-2009 (2009)
13. Alles, M.: Implementation Aspects of Advanced Channel Decoding. Ph.D. thesis, University of Kaiserslautern (2010)
14. Luby, M., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Improved low-density parity-check codes using irregular graphs. IEEE Trans. Inf. Theory **42**(2), 585–598 (2001)
15. Schläfer, P., Alles, M., Weis, C., WehnW, N.: Design space of flexible multi-gigabit LDPC decoders. VLSI Des. J. 2012 (2012)
16. Richardson, T., Urbanke, R.: The renaissance of gallager's low-density pariy-check codes. IEEE Commun. Mag. **41**, 126–131 (2003)
17. Tanatmis, A., Ruzika, S., Hamacher, H.W., Punekar, M., Kienle, F., Wehn, N.: A separation algorithm for improved LP-decoding of linear clock codes. IEEE Trans. Inf. Theory **56**(7), 3277–3289 (2010). doi:10.1109/TIT.2010.2048489
18. Scholl, S.: Dissertation in preparation. Ph.D. thesis, Department of Electrical Engineering and Information Technology, University of Kaiserslautern
19. Richardson, T.J., Shokrollahi, M.A., Urbanke, R.L.: Design of capacity-approaching irregular low-density parity-check codes. IEEE Trans. Inf. Theory **47**(2), 619–637 (2001)
20. Xiao, H., Banihashemi, A.: Improved Progressive-Edge-Growth (PEG) construction of irregular LDPC codes. IEEE Commun. Lett. **8**(12), 715–717 (2004)
21. Tian, T., Jones, C., Villasenor, J.D., Wesel, R.D.: Selective avoidance of cycles in irregular LDPC code construction. IEEE Trans. Commun. **52**(8), 1242–1247 (2004)
22. Richardson, T., Urbanke, R.: Modern Coding Theory. Cambridge University Press, Cambridge (2008)
23. Divsalar, D., Jones, C., Dolinar, S., Thorpe, J.: Protograph based LDPC codes with minimum distance linearly growing with block size. In: Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE, vol. 3, p. 5 (2005). doi:10.1109/GLOCOM.2005.1577834
24. TrellisWare: THE F LDPC FAMILY. http://www.trellisware.com/ (2007)
25. Blanksby, A., Howland, C.J.: A 690-mW 1-Gb/s, Rate-1/2 low-density parity-check code decoder. IEEE J. Solid-State Circ. **37**(3), 404–412 (2002)
26. Cocco, M., Dielissen, J., Heijligers, M., Hekstra, A., Huisken, J.: A Scalable Architecture for LDPC Decoding. In: Proceedings of 2004 Design, Automation and Test in Europe (DATE '04). Paris, France (2004)
27. Boutillon, E., Castura, J., Kschischang, F.: Decoder-first code design. In: Proceedings of 2nd International Symposium on Turbo Codes & Related Topics, pp. 459–462. Brest, France (2000)
28. Mansour, M., Shanbhag, N.: Architecture-Aware Low-Density Parity-Check Codes. In: Proceedings of 2003 IEEE International Symposium on Circuits and Systems (ISCAS '03). Bangkok, Thailand (2003)

29. Kienle, F., Wehn, N.: Joint Graph-Decoder Design of IRA-Codes on Scalable Architectures. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04), pp. IV-673-676. Montreal, Canada (2004)
30. Urard, P., Paumier, L., Heinrich, V., Raina, N., Chawla, N.: A 360mW 105Mb/s DVB-S2 Compliant Codec based on 64800b LDPC and BCH Codes enabling Satellite-Transmission Portable Devices. In: Proceedings of Digest of Technical Papers. IEEE International Solid-State Circuits Conference ISSCC 2008, pp. 310–311 (2008). doi:10.1109/ISSCC.2008.4523181
31. Shih, X.Y., Zhan, C.Z., Lin, C.H., Wu, A.Y.: An 8.29 mm 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 & CMOS process. IEEE J. Solid-State Circ. **43**(3), 672–683 (2008). doi:10.1109/JSSC.2008.916606
32. Mansour, M., Shanbhag, N.: Low-Power VLSI Decoder Architectures for LDPC Codes. In: Proceedings of 2002 International Symposium on Low Power Electronics and Design (ISLPED '02). Monterey, California, USA (2002)
33. Dielissen, J., Hekstra, A., Berg, V.: Low cost LDPC decoder for DVB-S2. In: Proceedings of 2006 Design, Automation and Test in Europe (DATE '06). Munich, Germany (2006)
34. Mller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., Wehn, N.: A novel LDPC decoder for DVB-S2 IP. In: Proceedings of DATE '09. Design, Automation. Test in Europe Conference. Exhibition, pp. 1308–1313 (2009)
35. Kienle, F., Brack, T., Vogt, T.: Error Control Coding for B3G/4G Wireless Systems, chap. Architecture and Hardware Requirements for Turbo and LDPC decoders, pp. 113–188. WILEY (2011)
36. Alles, M., Berens, F., Wehn, N.: A Synthesizable IP Core for WiMedia 1.5 UWB LDPC Code Decoding. In: Proceedings of IEEE International Conference on Ultra-Wideband ICUWB 2009, pp. 597–601. Vancouver, Canada (2009)
37. Guilloud, F.: Architecture génrique de dcodeur de codes LDPC. Ph.D. thesis, L'École nationale supérieure des télécommunications (2004)
38. Chen, Z., Peng, X., Zhao, X., Xie, Q., Okamura, L., Zhou, D., Goto, S.: A macro-layer level fully parallel layered LDPC decoder SOC for IEEE 802.15.3c application. In: VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on, pp. 1–4 (2011). doi:10.1109/VDAT.2011.5783634
39. Rovini, M., Insalata, N., Rossi, F., Fanucci, L.: LDPC Decoding in Fixed-Point Precision: a Systematic Quantisation Study. In: Proceedings of 2005 SoftCOM. Split, Croatia (2005)
40. Hu, X., Eleftheriou, E., Arnold, D., Dholakia, A.: Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes. In: Proceedings of 2001 Global Telecommunications Conference (GLOBECOM '01), pp. 1036–1041. San Antonio, TX, USA (2001)
41. Sharifi Tehrani, S., Mannor, S., Gross, W.J.: Fully Parallel Stochastic LDPC Decoders. IEEE Trans. Signal Process. **56**(11), 5692–5703 (2008). doi:10.1109/TSP.2008.929671
42. Zhang, T., Wang, Z., Parhi, K.: On finite precision implementation of low-density parity-check codes decoder. In: Proceedings of International Symposium on Circuits and Systems (ISCAS '01). Antwerp, Belgium (2001)
43. Guilloud, F., Boutillon, E., Danger, J.: λ-Min Decoding Algorithm of Regular and Irregular LDPC Codes. In: Proceedings of 3rd International Symposium on Turbo Codes & Related Topics, pp. 451–454. Brest, France (2003)
44. Masera, G., Quaglio, F., Tarable, A., Vacca, F.: Interconnection Structure for a Flexible LDPC Decoder. In: Proceedings of WiRTeP—Wireless Reconfigurable Terminals and Platforms, pp. 58–62 (2006)
45. Moussa, H., Baghdadi, A., Jquel, M.: Binary de Bruijn interconnection network for a flexible LDPC/turbo decoder. In: Proceedings of IEEE International Symposium on Circuits and Systems ISCAS 2008, pp. 97–100 (2008). doi:10.1109/ISCAS.2008.4541363
46. Neeb, C., Thul, M.J., Wehn, N.: Network-on-Chip-Centric Approach to Interleaving in High Throughput Channel Decoders. In: Proceedings of IEEE International Symposium on Circuits and Systems ISCAS 2005, pp. 1766–1769. Kobe, Japan (2005)

47. Gimmler, C., Kienle, F., Weis, C., Wehn, N., Alles, M.: ASIC design of a Gbit/s LDPC decoder for iterative MIMO systems. In: Proceedings of International Computing, Networking and Communications (ICNC) Conference, pp. 192–197 (2012). doi:10.1109/ICCNC.2012.6167409
48. Rovini, M., Rossi, F., Ciao, P., Insalata, N., Fanucci, L.: Layered Decoding of Non-Layered LDPC Codes. In: Proceedings of 9th EUROMICRO Conference on Digital System Design (DSD'06), pp. 537–544. Dubrovnik, Croatia (2006)

Chapter 8
Bit-Interleaved Coded MIMO System

In Chap. 2 we introduced the so called bit-interleaved coded modulation systems. In BICM systems a channel code is followed by an interleaver stage and the modulation. So far, the transmission of information was assumed to be done via one pair of antennas (one on the sender side and one on the receiver side), which means one symbol was sent in each time slot. However, as introduced in Sect. 2.4 it is possible to transmit multiple symbols via multiple antennas, while on the receiver side it is also possible to receive information via multiple antennas. In this chapter we will enhance the outer tranceiver of the BICM system towards so called multiple-input multiple-output (MIMO) antenna systems. The resulting BICM-MIMO system is part of the outer transceiver, as a clear separation of the frequency modulation part is possible.

In a MIMO system a symbol stream is demultiplexed to multiple transmit antennas while the receiver side collects superimposed samples, which are additionally disturbed by channel noise, from multiple receive antennas. There are two reasons to use MIMO antenna systems: increasing the data rate and/or increasing the reliability of the transmission. Many techniques exist to reduce the complexity of the MIMO demodulation process, e.g. by introducing constraints in space and/or time, by trading off diversity gain and multiplexing gain. MIMO demodulator or MIMO detector are used as synonyms in the following. Two of the most famous space-time codes are the original Bell Labs layered space-time (BLAST) technique [1] or the well-known Alamouti scheme [2].

Typically, all of these techniques have to be concatenated with an additional channel code to ensure a desired quality of service. The overall data rate of the transmission—the number of bits transmitted per channel use—is determined by the space-time encoder and the channel encoder independently. The overall complexity depends rather on the individual modules than on the integration of the two.

Transmission rates close to the theoretical capacity of a MIMO channel can be achieved by a simple encoder structure, by a serial concatenation of an outer code, interleaver, and modulator. In this case the modulator performs a spatial multiplexing of the symbol stream without introducing any further constraints and thus data rate

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_8,

loss. This simple concatenation can be seen as a classical bit-interleaved coded-modulation scheme (BICM).

Approaching the MIMO capacity limit can be achieved by an iterative receiver, where probabilistic (soft) information are exchanged between MIMO detector and channel decoder [3, 4]. However, the demodulator has to calculate maximum a posteriori probabilities (APP) for each bit which can be computationally demanding. The complexity of the MIMO-APP demodulator depends on the number of transmit antennas and the size of the modulation alphabet.

Typically, the MIMO detector and the channel decoder are designed independently, while the overall complexity mainly depends on each individual part. There exist many different possibilities for the realization of the required soft-in soft-out MIMO detector. This chapter deals only with the one which can provide the best communications performance. This MIMO detector is based on the so called sphere detection algorithm. This chapter puts focus on a design flow to improve a system in terms of architectural efficiency and communications performance. The three steps are: understanding the system, deriving architectural constraints, and improve the system. The last step requires know-how from the algorithmic domain and the hardware domain to improve the system. The three steps are sketched in Fig. 8.1. Understanding all steps requires the knowledge of the previous chapters.

- First the system set up is explained and the state-of-the art performance of iterative BICM-MIMO systems is introduced. The basic MIMO detection algorithms are explained, but only the sphere-based algorithm is considered further. The presented soft-in soft-out sphere detector can calculate the optimal symbol-by-symbol MAP criterion.
- The second part deals with an architectural evaluation of the BICM-MIMO system. We show how to derive the necessary parallelism for a channel decoder with or without feedback loop. A high level exploration like the presented is always required before realizing an architecture. This is a classical top down approach at which a designer evaluates the parallelism of the data flow. Based on this the number of instances for the individual components can be derived.

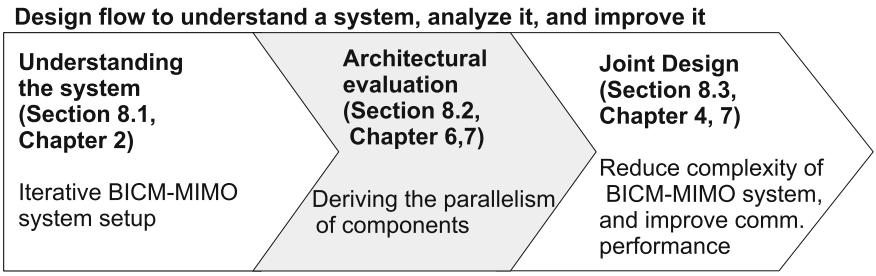


Fig. 8.1 Design steps for system improvement. Parts of this chapter and the related topics which are required for a joint design of algorithm and architecture with respect to BICM-MIMO systems

- Implementing a BICM-MIMO system in a top down approach results in independent implementations of the channel decoder and the MIMO detector. The overall complexity will be lower bounded by the VLSI footprints of the individual components. The basic philosophy of joint MIMO detector and channel code design, which is shown in the third part of the chapter, requires the knowledge of the algorithm and a basic understanding of architectural design. The goal is to reduce the complexity and to increase the communications performance at the same time. This can only be achieved when architectural know-how is taken into consideration in the early phases of system design.

8.1 State-of-the-Art BICM-MIMO Systems

In this section we will revise state-of-the-art BICM-MIMO systems. We assume for all MIMO system that they have a symmetrical number of antenna setup, i.e. $M_T = M_R$. The MIMO encoding and decoding processes are explained in the following paragraphs, furthermore the achievable communications performance is presented.

8.1.1 MIMO Transmitter

The entire encoding procedure is a bit-interleaved coded modulation (BICM) scheme and is shown in Fig. 8.2. The source bits are encoded by an outer channel code of code rate R. The resulting codeword is interleaved and then mapped to symbols. The symbols are then multiplexed to the different antennas and M_T symbols are transmitted simultaneously at each time step. In the following we will explain the notation, which differs slightly from the notation used in previous chapters. Instead of scalars each time slot now holds a vector of transmitted data. The source generates a random information word $\boldsymbol{u}$ of length K which is encoded by the channel encoder. The resulting codeword $\boldsymbol{x}$ consists of N bits which are grouped into N_s subblocks $\boldsymbol{x}_n$. In the following we combine the interleaver stage and this grouping in one stage with the resulting codeword matrix $\boldsymbol{X}$

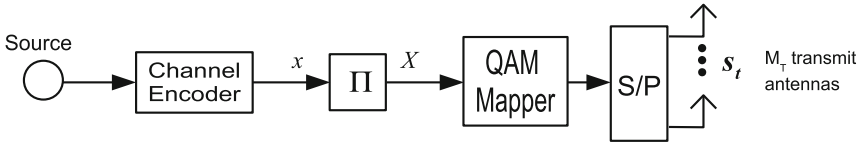


Fig. 8.2 Typical MIMO transmitter with the encoded codeword $\boldsymbol{x}$ and the transmisson vector $\boldsymbol{s}_t$. The serial to parallel de-multiplexing stage is denoted as S/P

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n, \ldots, \boldsymbol{x}_{N_s}). \tag{8.1}$$

Each subblock consists of Q coded bits (Q being the modulation size).

$$\boldsymbol{x}_n = (x_{1,n}, x_{2,n}, \ldots, x_{q,n}, \ldots, x_{Q,n}) \tag{8.2}$$

Each subblock $\boldsymbol{x}_n$ is mapped to one complex symbol s chosen from a 2^Qary QAM modulation scheme (Gray mapping). The advantage of this matrix notation is that it combines the interleaver and the allocation of the bit positions to symbol positions. At any given time M_T consecutive symbols are combined in one transmitted vector $\boldsymbol{s}_t$.

$$\boldsymbol{s}_t = (s_{1,t}, s_{2,t}, \ldots, s_{M_T,t}) \tag{8.3}$$

The whole modulated sequence is represented by

$$\boldsymbol{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_t, \ldots, \boldsymbol{s}_T) \tag{8.4}$$

T time slots are needed to transmit all symbols of one codeword. The transmission of one transmission vector $\boldsymbol{s}_t$ in time step t is modeled by multiplying it with the channel matrix $\boldsymbol{H}_t$ and adding Gaussian noise $\boldsymbol{n}_t$:

$$\boldsymbol{y}_t = \boldsymbol{H}_t \cdot \boldsymbol{s}_t + \boldsymbol{n}_t \tag{8.5}$$

The channel modeling and the difference between a quasi-static channel and an ergodic channel was already introduced in Sect. 2.4. For all presented communications performance curves in this chapter the type of channel model is stated explicitly.

The overall data rate of the presented transmission is $\eta = R M_T Q$ which reflects the number of information bits per time slot. Often, the used channel codes in BICM-MIMO system are either convolutional codes, or turbo codes, or LDPC codes respectively.

8.1.2 BICM-MIMO Receiver

We have to distinguish between BICM-MIMO receivers with an open loop structure and a closed loop structure. The different receiver types are shown in Figs. 8.3 and 8.4. We denote the information received via the M_R received antennas as a matrix $\boldsymbol{Y}$,

$$\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t, \ldots, \boldsymbol{y}_T) \tag{8.6}$$

with $\boldsymbol{y}_t$ being the received samples in time slot t,

$$\boldsymbol{y}_t = (y_{1,t}, y_{2,t}, \ldots, y_{M_R,t}) \tag{8.7}$$

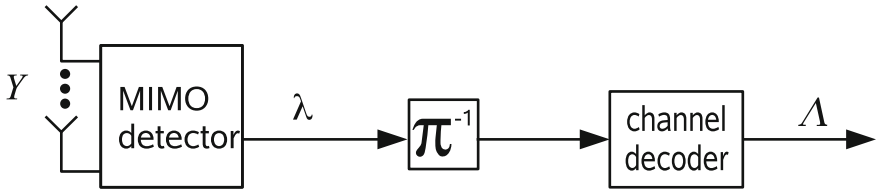


Fig. 8.3 TBICM-MIMO receiver with **open loop** structure. The MIMO detector transforms the received information $\boldsymbol{Y}$ into LLRs ($\boldsymbol{\lambda}$) for each bit position. The interleaved MIMO detector output is passed to the channel decoder

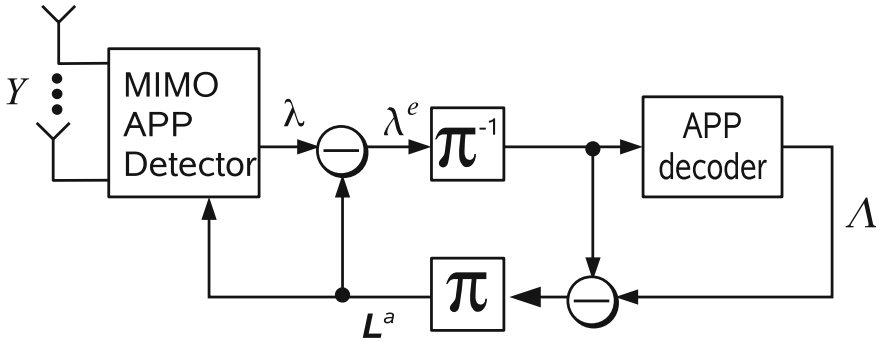


Fig. 8.4 BICM-MIMO receiver with **closed loop** structure, with the a priori information $\boldsymbol{L}^a$ and the extrinsic information $\boldsymbol{\lambda}^e$ passed to the outer decoder

As already stated throughout this manuscript it is always assumed that the channel ($\boldsymbol{H}_t$) is perfectly known by the receiver.

Assuming the open loop case of Fig. 8.3 we can calculate the detector output information using different criteria. The corresponding MIMO detectors are outlined shortly in the following. Note, that only systems with spatial multiplexing are assumed.

- Zero forcing detector: The received vector is multiplied by $\boldsymbol{H}^{\dagger}$ the pseudo-inverse of the channel matrix

$$\hat{\boldsymbol{z}}^{ZF} = \boldsymbol{H}^{\dagger}\boldsymbol{y}_t = (\boldsymbol{H}^H\boldsymbol{H})^{-1}\boldsymbol{H}^H\boldsymbol{y}_t. \tag{8.8}$$

  The major problem of this approach is the amplification of the noise which results in a large degradation in communications performance. The zero forcing solution $\hat{\boldsymbol{z}}^{ZF}$ has a maximum diversity order of $M_R - M_T + 1$ [5]. The hard decision symbols are obtained by quantizing the result to the closest constellation points.
- MMSE detector: The minimum mean square estimator calculates a filter matrix $\boldsymbol{W}$ which minimizes the following condition

$$\boldsymbol{W}^{MMSE} = \arg\min_{\boldsymbol{W}} \left\{ E\{||\boldsymbol{W}^H\boldsymbol{y}_t - \boldsymbol{s}_t||^2\} \right\} \tag{8.9}$$

The resulting filter output $\hat{\boldsymbol{z}}^{MMSE}$ evaluates to

$$\hat{\boldsymbol{z}}^{MMSE} = \boldsymbol{W}^{MMSE}\boldsymbol{y}_t = \left(\boldsymbol{H}^H\boldsymbol{H} + \frac{M_T}{SNR}\boldsymbol{I}\right)^{-1}\boldsymbol{H}^H\boldsymbol{y}_t, \tag{8.10}$$

with $\boldsymbol{I}$ representing a diagonal matrix which is weighted by the corresponding noise. For large SNR values the MMSE solution approximates the ZF solution, thus a diversity order of $M_R - M_T + 1$ is obtained [5].

- ML detector: ZF and MMSE are so called linear detectors while the ML detector is not. The ML detector calculates the maximum likelihood symbol estimation $\hat{\boldsymbol{s}}^{ML}$ which is defined as:

$$\hat{\boldsymbol{s}}^{ML} = \arg\min_{\boldsymbol{s}} \left\{||\boldsymbol{y}_t - \boldsymbol{H}_t\boldsymbol{s}||^2\right\} \tag{8.11}$$

  The ML detector has a diversity order of M_R [5] and provides hard-output values. Though correct, we can improve further the BICM system performance by calculating soft-output values.
- APP detector: Soft-output values can be obtained by applying an a posteriori probability (APP) criterion. The results is denoted as MIMO-APP to distinguish it from the APP detectors for single antenna systems introduced already in Sect. 2.2. The major difference for MIMO-APP detection is the conditional probability on a received vector $\boldsymbol{y}_t$ which comprises the information of M_R symbols. The LLR value on each individual bit can be calculated by

$$\lambda(x_{t,q,m}) = \ln \frac{P(x_{t,q,m} = 0|\boldsymbol{y}_t)}{P(x_{t,q,m} = 1|\boldsymbol{y}_t)} \tag{8.12}$$

Since the channel decoder has its maximum achievable coding gain with APP information at its input we only concentrate on detectors which can provide APP information.

As mentioned, for the MIMO detection and channel decoding we have to distinguish between open loop and closed loop receiver structures.

- **Open loop**: Figure 8.3 shows a receiver structure with a demodulator concatenated with an outer channel decoder. The APP information of the MIMO detector is interleaved and directly passed to the channel code.
- **Closed loop**: Figure 8.4 shows a structure in which the demodulator and the channel decoder pass information back and forth. During the iterative message exchange between detector and outer decoder the input messages we have to ensure the extrinsic information principle, similar as done for turbo or LDPC decoding. $\boldsymbol{L}^a$ is the a priori information which is passed to the MIMO-APP detector. $\boldsymbol{\lambda}^e = \boldsymbol{\lambda} + \boldsymbol{L}^e$ comprises the information $\boldsymbol{\lambda}$ extracted from the received information and the additional gain $\boldsymbol{L}^e$ obtained due to a priori input information. During the first demodulation there exist no a priori information, thus $\boldsymbol{L}^a = 0$.

Closed loop BICM-MIMO receivers can gain more than 3db in communications performance compared to open loop receivers [3]. The final gain depends on many system parameters like the used antenna system, modulation type, number of iterations, channel model and the channel code. Communications performance results are shown in Sect. 8.1.3.

In the following only the MIMO-APP demodulator capable for iterative processing is further considered. A MIMO-APP detector computes logarithmic likelihood values (LLRs) on each bit according to

$$\lambda(x_{q,m}) = \ln \frac{P(x_{q,m} = +1|\boldsymbol{y})}{P(x_{q,m} = -1|\boldsymbol{y})} \tag{8.13}$$

We have to evaluate this equation for each transmission time slot t, however, the index for the time slot t is skipped from now on. q is the index within a modulated symbol with $q \in 1, ..., Q$ and m the index with respect to the antenna layer with $m \in 1, ..., M$. For independent $x_{q,m}$, the probability $P(x_{q,m} = 0|\boldsymbol{y})$ is obtained by summing up the probabilities of all possible symbol vectors $\boldsymbol{s}$ which contain $x_{q,m} = 0$.

$$P(x_{q,m} = 0|\boldsymbol{y}) = \sum_{\forall \boldsymbol{s}|x_{q,m}=0} P(\boldsymbol{s}|\boldsymbol{y}) \tag{8.14}$$

$\boldsymbol{s}|x_{q,m} = 0$ determines the symbol vector conditioned that the corresponding bit position is 0. This calculation is related to the demodulator example of Eq. D.9.

Using Bayes theorem, $P(\boldsymbol{s}|\boldsymbol{y})$ can be expressed as

$$P(\boldsymbol{s}|\boldsymbol{y}) = \frac{P(\boldsymbol{s}) \cdot P(\boldsymbol{y}|\boldsymbol{s})}{P(\boldsymbol{y})} \tag{8.15}$$

We can observe that the analyzed probability consists of three parts. $P(\boldsymbol{s})$ takes into account that not every $\boldsymbol{s}$ is equally likely given the a-priori information $\boldsymbol{L}^a$ from the channel decoder. As the codeword is interleaved before the QAM mapping the bits $x_{q,m}$ are assumed to be independent from each other. Therefore, $P(\boldsymbol{s})$ is the product of the probabilities of the individual bits that were mapped into $\boldsymbol{s}$:

$$P(\boldsymbol{s}) = \prod_{\forall q,m} P(x_{q,m}) \tag{8.16}$$

The term $P(\boldsymbol{y}|\boldsymbol{s})$ is the probability of receiving $\boldsymbol{y}$ under the condition that the vector $\boldsymbol{s}$ was sent. $P(\boldsymbol{y}|\boldsymbol{s})$ can be calculated via the corresponding Gaussian function, as an additive noise is assumed. The third part $P(\boldsymbol{y})$ is constant during the detection of $\boldsymbol{y}$ and is canceled out when applying (8.15) to calculate the LLRs of (8.13). Finally for the soft-input soft-output processing we have to evaluate

$$\lambda(x_{q,m}) = \ln \frac{\sum_{\forall \boldsymbol{s}|x_{q,m}=0} P(\boldsymbol{s}) \cdot e^{-||\boldsymbol{y}-\boldsymbol{H}\boldsymbol{s}||^2/N_0}}{\sum_{\forall \boldsymbol{s}|x_{q,m}=1} P(\boldsymbol{s}) \cdot e^{-||\boldsymbol{y}-\boldsymbol{H}\boldsymbol{s}||^2/N_0}} \tag{8.17}$$

Applying the Jacobian logarithm and ignoring the correction term results in the Max-Log-Map approximation. The detailed discussion can be found in [3, 6].

$$\begin{aligned}\lambda(x_{q,m}) \approx & \min_{\forall \boldsymbol{s}|x_{q,m}=0} \left\{ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}) \right\} \\ & - \min_{\forall \boldsymbol{s}|x_{q,m}=1} \left\{ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}) \right\}\end{aligned} \tag{8.18}$$

An interpretation for (8.18) is that we derive the LLR value $\lambda(x_{q,m})$ from the most likely symbol vectors $\boldsymbol{s}$ with one bit $x_{q,m}$ being 0 or 1 respectively. The expression $N_0 \sum_{\forall q',m'} \ln P(x_{q',m'})$ determines the a priori information under the constraint of $\forall \boldsymbol{s}|x_{q,m} = \pm 1$.

The metric $d(\boldsymbol{s})$ measures the likelihood that a specific vector $\boldsymbol{s}$ has been sent:

$$d(\boldsymbol{s}) = \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}). \tag{8.19}$$

Small metrics $d(\boldsymbol{s})$ relate to a high probability of $\boldsymbol{s}$ having been sent.

Calculating all possible $d(\boldsymbol{s})$ to determine Eq. 8.18 quickly grows infeasible for higher antenna constellations and/or higher order modulations as the complexity grows with 2^{QM}. Therefore, many sub-optimal algorithms with lower were devised. Most of them are based on a tree search. In order to map the metric calculations Eq. 8.19 on a tree, the channel matrix $\boldsymbol{H}$ is decomposed into an unitary matrix $\boldsymbol{Q}$ and an upper-triangular matrix $\boldsymbol{R}$. The Euclidean distance is rewritten as

$$\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 = \left\|\boldsymbol{y}' - \boldsymbol{R}\boldsymbol{s}\right\|^2 \tag{8.20}$$

with $\boldsymbol{y}' = \boldsymbol{Q}^H \boldsymbol{y}$. Equation 8.19 is replaced by the equivalent metric

$$d(\boldsymbol{s}) = \left\|\boldsymbol{y}' - \boldsymbol{R}\boldsymbol{s}\right\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}) \tag{8.21}$$

The triangular structure of $\boldsymbol{R}$ allows the recursive calculation of $d(\boldsymbol{s})$ which can be seen when we fully extend the term for the Euclidean distance in the equation:

$$\left\|\boldsymbol{y'}-\boldsymbol{Rs}\right\|^2=\left\|\begin{pmatrix}y'_1\\y'_2\\\vdots\\y'_M\end{pmatrix}-\begin{pmatrix}r_{1,1}&0&\cdots&0\\r_{2,1}&r_{2,2}&0&\vdots\\\vdots&\vdots&\ddots&0\\r_{M,1}&r_{M,2}&\cdots&r_{M,M}\end{pmatrix}\begin{pmatrix}s_1\\s_2\\\vdots\\s_M\end{pmatrix}\right\|^2$$
$$=\sum_{m=1}^{M}\left|y'_m-\sum_{j=1}^{m}r_{m,j}s_m\right|^2. \tag{8.22}$$

Using the partial symbol vector $s^{(m)}=(s_1,s_2,\ldots,s_m)$ the recursive calculation for each antenna layer m can be written as

$$d_m=d_{m-1}+\gamma_m\left(s^{(m)}\right). \tag{8.23}$$

$d_0=0$ is used for initialization. Including a priori information the partial distance metric of an antenna layer $\gamma_m(s^{(m)})$ evaluates to:

$$\gamma_m\left(s^{(m)}\right)=\left|y'_m-\sum_{j=1}^{m}r_{m,j}s_j\right|^2-N_0\sum_{q=1}^{Q}\ln P(x_{q,m}). \tag{8.24}$$

The recursive calculation can be represented by a tree with $M+1$ layers as shown in Fig. 8.5 for two different cases. The top figure represents a four antenna system with BPSK modulation (one bit per symbol), while the lower tree represents two antennas with two bits per symbol.

The root node corresponds to d_0 and each leaf node corresponds to the metric $d_M=d(\boldsymbol{s})$ of one possible vector $\boldsymbol{s}$. Each layer corresponds to the detection of one symbol s_m. Branches are labeled with the corresponding bit pattern of the symbol. Each branch in the tree can be associated with a certain weight $\gamma_m(s^{(m)})$ which depends on the path from the root node to the corresponding edge. Thus, when advancing from a parent node to a child node, the metric of the child node d_m is calculated from the metric of the parent node d_{m+1} and the branch metric $\gamma_m(s^{(m)})$, see Eq. 8.23.

Evaluating all possibilities of $d(\boldsymbol{s})$ results in $P=2^{MQ}$ possibilities. E.g. for a 4×4 antenna system with a 16-QAM modulation this results in $P=2^{MQ}=65536$ possibilities reflecting all bit possibilities which are decoded within one transmission vector $\boldsymbol{s}_t$.

Based on this tree search, many different MIMO detection algorithms exist. The main differences between the algorithms can be described by how the tree is traversed, e.g. breadth-first or depth-first, and how branches of the tree are pruned. In general, those algorithms achieve different results in terms of communications performance and implementation complexities.

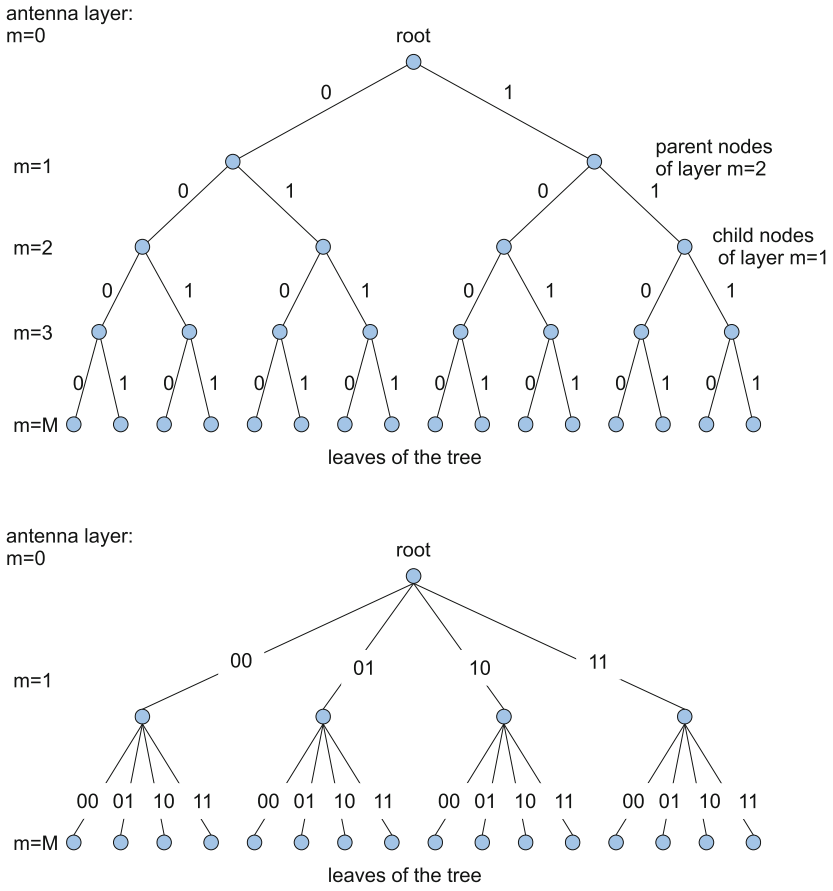


Fig. 8.5 Decision tree for MIMO detection, upper M = 4 antennas and one bit per modulated symbol, e.g. BPSK. Lower figure with 2 antennas and a two bit modulation, e.g. 4-AM

8.1.3 Communications Performance of State-of-the-Art Systems

All following communications performance results assume a BICM system with rate $R = \frac{1}{2}$ LDPC code as channel coding scheme. We distinguish between the **closed loop** system with feedback between channel decoder and MIMO demapper and the **open loop** system without feedback. Figure 8.6 shows the communications performance for a 16-QAM 4×4 system (8 bits/channel use). Each E_b/N_0 point is simulated with 100k transmitted codewords of length $N = 1920$. Here we assume a quasi-static channel, i.e., the channel matrix $\boldsymbol{H}_t$ remains constant within one transmitted codeword (120 channel uses). Four different performance curves are shown. The left most curve shows the outage capacity which is the theoretical lower bound for reliable transmission for this system. The curves are now explained starting

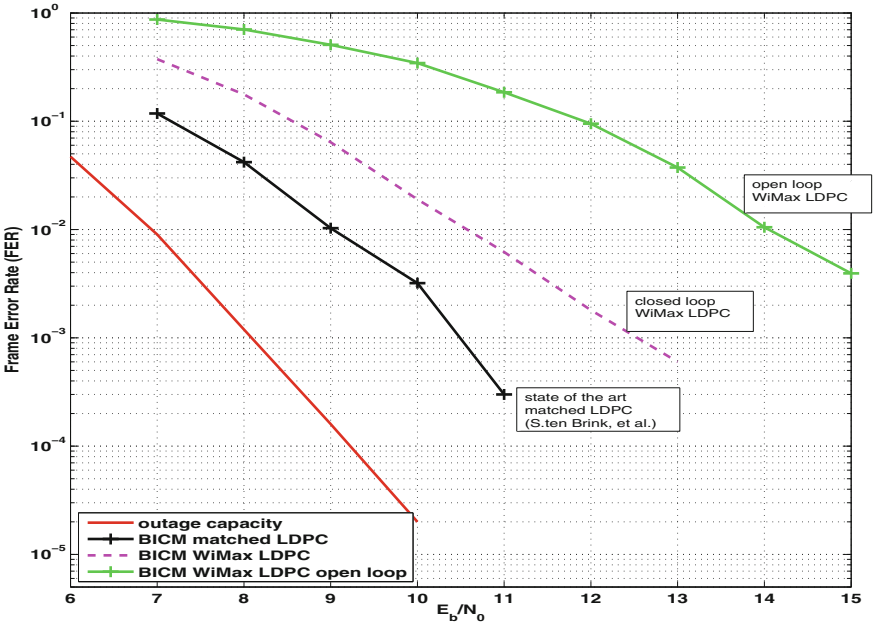


Fig. 8.6 State-of-the-art communications performance for quasi-static MIMO channel

with the one with worst performance. The right curve is achieved by performing the MIMO-APP demodulation and 40 iterations of an LDPC code, thus building an open loop system. A WiMAX type LDPC code is used with a degree distribution of $f_{[6,3,2]} = \left\{\frac{5}{24}, \frac{1}{3}, \frac{11}{24}\right\}$, $g_{[7,6]} = \left\{\frac{1}{3}, \frac{2}{3}\right\}$. The next performance improvement step is to close the loop and do 5 outer loop iterations by evaluating Eq. 8.18. Iterations within an LDPC decoder are denoted as inner iterations, the feedback via MIMO-APP detector is denoted as outer iterations. For LDPC codes we can adjust the communications performance by its degree distribution as seen in Chap. 7. An LDPC code can be adjusted for iterative feedback loops by utilizing EXIT chart techniques, according to [7]. The new LDPC code is thinned out to obtain a different convergence performance. The resulting degree distribution here is $f_{[6,3,2]} = \left\{\frac{1}{8}, \frac{2}{8}, \frac{5}{8}\right\}$, $g_{[6,5]} = \left\{\frac{1}{2}, \frac{1}{2}\right\}$. All LDPC codes presented here are quasi-cyclic to facilitate their implementation in hardware, as explained in Sect. 7.3.1. Especially for the matched LDPC design many variable nodes of degree two are present which gives a higher error floor. This degree distribution is a good trade-off between good convergence and a reasonable error floor. The performance gain of a closed loop system compared to an open loop system can be above 4 dB as seen in Fig. 8.6. The BICM system with matched LDPC codes is denoted as matched BICM in the following, the BICM with WiMAX LDPC code is denoted as WiMAX BICM, respectively.

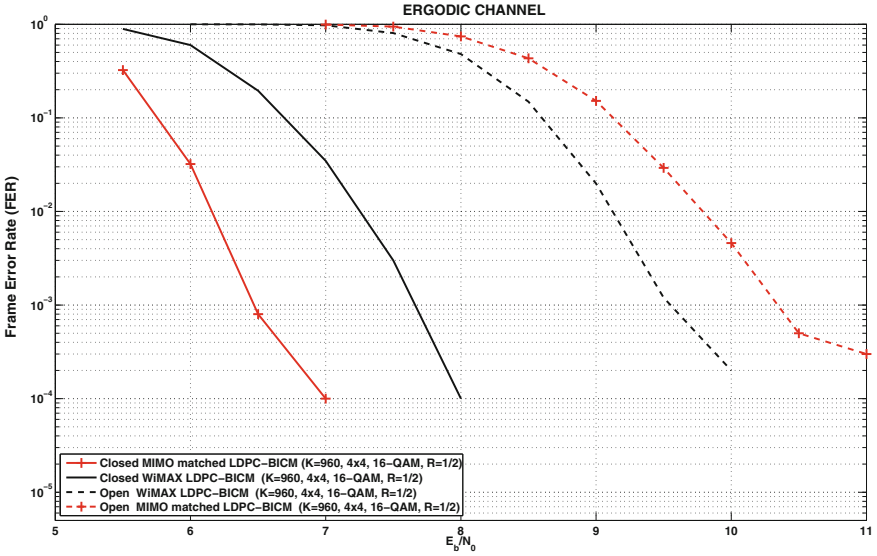


Fig. 8.7 Communications performance of open loop systems and a closed loop systems using two LDPC codes. One system utilizes an LDPC code which is matched with respect to the 4×4 MIMO system, the other system uses the standard WiMAX code

Figure 8.7 shows the communications performance with the same set up (4×4 antennas, 16-QAM), but for an ergodic channel, i.e., $\boldsymbol{H}_t$ changes for each time slot. The graph shows the results for the open loop system and the closed loop system for the matched BICM and WiMAX BICM system. For the closed loop performance, using five outer iterations, we can see a 1 dB performance gain of the matched BICM system at $FER = 10^{-2}$. This gain between matched BICM and WiMAX BICM system is identical to that obtained under quasi-static channel conditions. However, for an open loop performance the WiMAX BICM system outperforms the matched BICM system. In summary two important aspects are highlighted:

- The gain in communications performance for different BICM-MIMO systems depends on the number of outer iterations. Achieving always the best communications performance within all outer iterations can not be achieved by a single code.
- Second important aspect which should be further discussed is the realistic number of outer loop iterations which can be performed in hardware designs. The important analysis of outer loop iterations for a BICM receiver architecture is done in the next section.

8.2 Architecture Feasibility BICM-MIMO

In this section we analyze complexity aspects for the BICM receiver system shown in Fig. 8.4. Again we distinguish between the closed loop system with feedback between channel decoder and MIMO demapper and the open loop system without feedback. For the open loop system we assume that the MIMO demapper provides soft-output information to the channel decoder.

For the realization of the closed loop system several possibilities for the architecture exist. Assuming a fixed number of outer iteration, the iterations can be unrolled and pipelined. The corresponding architecture is shown in Fig. 8.8 for three iterations. Now three blocks are processed simultaneously in this pipeline which implies the instantiation of three hardware instances of the APP demodulator and the APP channel decoder respectively. Unrolling the loop will result in a linear increase in terms of area, since all memories and the logic will be duplicated. The architecture is inflexible with respect to number of performed closed loop iterations.

In the following we assume that one MIMO demodulator instance and one channel decoder instance are used which operate on one coded block. This is shown in Fig. 8.4. Only one codeword of the channel code is decoded, while an equal balancing of the processing time between these two instances is assumed. Thus, each of them is 50 percent of the overall time in an idle mode. This equal balancing relates to the typical iterative turbo code processing where information is exchanged between two MAP components, see Chap. 6. Here, in Fig. 8.4, the two components are the demodulator and outer channel decoder which are separated by an interleaver. We could also process two blocks concurrently in this engine, while one is processed by the demodulator and the other is processed by the APP decoder. However, the different number of iterations of the channel decoders and the feedback loop respectively will result in a difficult scheduling problem which is not in the scope of this analysis.

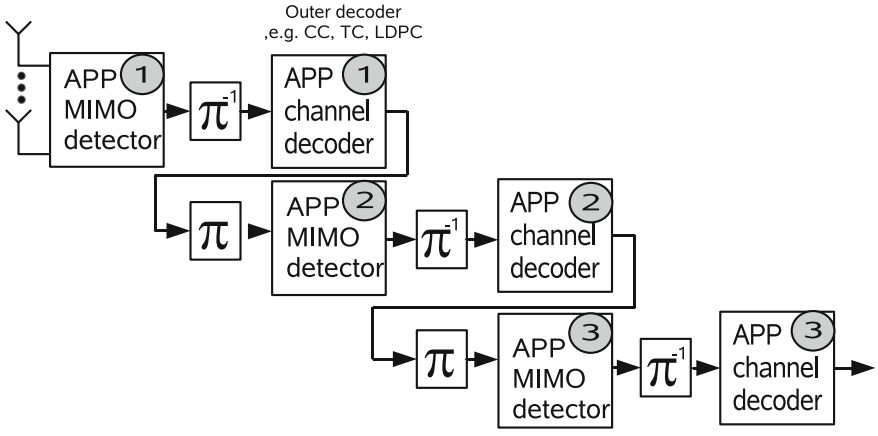


Fig. 8.8 Unrolled architecture for three outer iterations

A pragmatic solution for processing two blocks simultaneously is the instantiation of two independent closed loop receivers. An appropriate allocation of the codeword to be processed has thus to be done at a higher architectural level.

For the following discussions we assume one instantiated MIMO detector and one channel decoder where only one codeword is processed. We consider the throughput constraints for the outer channel decoder utilizing turbo decoders and LDPC decoders, respectively. The parameters to derive the throughput constraints are shown in following:

$\#cycles$	number of cycles required to process one block
$P_{I/O}$	parallelization of the input/output
$iter$	number of iterations of the channel decoder (half iterations for turbo decoding)
P	parallelization of the decoder architecture: for turbo codes parallelization of the MAP architecture, for LDPC codes the number of concurrently processed edges
$\overline{d_{VN}}$	average variable node degree in the Tanner graph (LDPC codes)
N, K	block length, number of information bits
R	code rate of the channel code
f_{clk}	clock frequency
$\delta_{overhead}$	additional fixed architectural overhead (e.g. for flushing the pipeline)
$\#\frac{bits}{cycle}$	normalized throughput: number of information bits decoded per clock cycle

The expected normalized throughput for a given architecture is

$$\#\frac{bits}{cycle} = \frac{K}{\#cycles} = \frac{N \cdot R}{\#cycles}. \tag{8.25}$$

The normalized throughput is a good performance metric of an architecture. For example, LTE advanced will require a turbo decoder architecture with $\#\frac{bits}{cycle} \sim 2$. For a typical frequency of $f_{cyc} = 300\,\text{MHz}$ this yields a payload (information bit) throughput of $T_{payload} = \#\frac{bits}{cycle} \cdot f_{cyc} = 600\,\text{Mbit/s}$.

Low-Density Parity-Check Decoder

The degree of parallelism for LDPC codes is defined as the number of simultaneously processed edges. The normalized throughput of an LDPC decoder can be approximated by:

$$\#\frac{bits}{cycle} \approx \frac{N \cdot R}{iter \cdot \frac{N \cdot \overline{d_{VN}}}{P}} = \frac{P \cdot R}{iter \cdot \overline{d_{VN}}} \tag{8.26}$$

Most of the current partly parallel architecture use the layered architecture where a nearly continuous processing takes place. No overhead cycles ($\delta_{overhead}$) are present within the iterative loop, for more details see Chap. 7 and [8, 9]. An average variable node degree of $\overline{d_{VN}} = 3.2$ (WiMAX LDPC) is assumed to derive the parallelism of a decoder architecture.

$$P = \left(\# \frac{bits}{cycle} \right) \cdot iter \cdot 3.2 \cdot \frac{1}{R} \tag{8.27}$$

Turbo Code Decoder

For turbo code decoders we define the parallelism P of the architecture as the number of LLRs which are exchanged per clock cycle between the component decoders. For turbo decoding a normalized throughput can be expressed as

$$\# \frac{bits}{cycle} \approx \frac{P}{2 \cdot iter \cdot \left(1 + \delta_{overhead} \cdot \frac{P}{K}\right)} \tag{8.28}$$

Turbo decoding needs two half iterations to process the two component codes, thus we need the term $2 \cdot iter$. The overhead $\delta_{overhead}$ is a big obstacle to a further parallelization of turbo decoders. Reducing them is a research topic that is receiving a lot of attention. The LTE Release 8 standard supports very high code rates ($R > 0.9$) which hampers the reduction of these $\delta_{overhead}$ cycles. The problem for high throughput turbo decoder architectures is the limited throughput increase for moderate length $K \sim 5000$ and increasing architecture parallelism P. In this case the term $\delta_{overhead} \cdot \frac{P}{K}$ is significant. For the following calculations the turbo decoder parallelization is calculated with an overhead of $\delta_{overhead} = 32$ and evaluates to:

$$P = \frac{\left(\# \frac{bits}{cycle} \right) \cdot 2 \cdot iter}{1 - \frac{\left(\# \frac{bits}{cycle} \right) \cdot 2 \cdot iter \cdot 32}{K}}. \tag{8.29}$$

Channel Coding Architecture in BICM-MIMO Systems

Table 8.1 shows the required parallelism for turbo decoder architectures and LDPC decoder architectures for an open loop system. A normalized throughput of $\# \frac{bits}{cycle} = 1$ and $\# \frac{bits}{cycle} = 2$ is assumed.

The required architecture parallelism depends on the number of iterations. For example, the presented turbo decoder of [10] has a throughput of 150 Mbit/s at $f_{cyc} = 300$ MHz. This turbo decoder uses an architecture of $P = 8$ and results in a normalized throughput of $\# \frac{bits}{cycle} = 0.5$ at 6.5 iterations. State-of-the-art turbo decoder architectures are already targeting a parallelism up to $P = 32$ [11]. However, the resulting chip size is very large and a further increase in parallelism is inefficient due to the overhead cycles. Thus, a further increase of the throughput off turbo decoders can best be achieved on block level, which means by instantiating multiple turbo decoder instances. For LDPC decoders a parallelism of $P = 360$ was already presented in 2005 [12], a larger degree of parallelism is possible. In summary for the open loop case we can say that there seems to be no practical obstacle to increasing

Table 8.1 Parallelization (P) of an **open loop** architecture for the given iterations and normalized throughput

Normalized throughput	Turbo codes $K = 6000$		LDPC codes $R = 1/2$	
	Iterations	P	Iterations	P
$\#\frac{bits}{cycle} = 1$	4 iter	9	5 iter	32
	6 iter	13	10 iter	64
	8 iter	18	20 iter	128
$\#\frac{bits}{cycle} = 2$	4 iter	18	5 iter	64
	6 iter	28	10 iter	128
	8 iter	39	20 iter	256

the throughput. This can always be done by multiple decoder instances, if the latency constraints can be fulfilled.

Now we analyze the required parallelism of turbo decoder or LDPC decoders used within an iterative BICM-MIMO receiver. Table 8.2 shows the parallelization of a turbo decoder or LDPC decoder for a given normalized throughput assuming a closed loop system. The normalized throughput is defined for the BICM-MIMO receiver while we have now a double iterative system with inner channel code iterations and outer feedback iterations. For example, the notation '2 outer–3 inner' means that the demodulator and channel code is active two times for each block, while the channel code performs three channel code iterations for each of these two times.

The parallelism for, e.g., a $\#\frac{bits}{cycle} = 1$ closed loop system with 2 outer–3 inner iterations translates to a 6 channel decoder iteration in the open loop system. However, since the decoder is assumed to be idle 50 the parallelism to achieve the desired system throughput. This results in large parallelism of P = 26 for the simplest case of Table 8.2. The number of outer iterations and inner channel code iterations are rather small in these examples. The parallelization even has to be increased in order to achieve the best possible communications performance. Note, that the normalized throughput assumption of $\#\frac{bits}{cycle} = 1$ or even higher required by upcoming standards, e.g. LTE advanced.

Table 8.2 Parallelization (P) of a **closed loop** architecture for the given outer and (inner) code iterations with respect to a normalized throughput

Normalized throughput	Turbo codes $K = 6000$		LDPC codes $R = 1/2$	
	Iterations	P	Iterations	P
$\#\frac{bits}{cycle} = 1$	2 outer–3 inner	26	2 outer–5 inner	128
	3 outer–3 inner	35	3 outer–5 inner	192
	4 outer–3 inner	52	4 outer–5 inner	256
$\#\frac{bits}{cycle} = 2$	2 outer–3 inner	55	2 outer–5 inner	256
	3 outer–3 inner	77	3 outer–5 inner	384
	4 outer–3 inner	110	4 outer–5 inner	512

As mentioned, for turbo decoder architectures we can increase the throughput further by creating multiple instances. However, for a closed loop system this requires to handle multiple blocks within an iterative BICM-MIMO system. In our opinion this is currently a strong argument against a double iterative scheme with targets of $\#\frac{bits}{cycle} = 2$, especially in the case of turbo codes. For LDPC codes achieving the required architecture parallelism seems to be easier.

The MIMO demodulator in the outer loop has to provide a normalized throughout of $\#\frac{bits}{cycle} = 1$ or $\#\frac{bits}{cycle} = 2$. The advantage of the MIMO detector is that each received vector $\boldsymbol{y}_t$ can be decoded independently. Thus, the high throughput requirements for the MIMO demodulation can always be achieved by multiple instances.

In summary: the double iterative structure poses a big challenge for the architectural realization. We can extract two options to limit the architectural overhead.

- Either we should get rid of the double iterative scheme,
- or we should ensure a very good communications performance with a limited number of outer iterations, e.g. just 2 outer iterations.

Reducing the number of closed loop iterations, while providing a good communications performance requires a joint consideration of the MIMO detector and channel code design. One possible joint design is presented in the next section.

8.3 Joint Architecture-Algorithm Design

Implementing an iterative BICM-MIMO system in a straight forward manner results in an independent implementation of the channel decoder and the MIMO detector. This straight forward approach was treated in the previous section. A lower bound for the overall area is given by the sum of the independent realizations. In fact, additional memories for the iterative data exchange are required [13].

The goal of the techniques presented in this section is to reduce the complexity of the MIMO-APP detection without sacrificing the overall data rate or the capacity approaching communications performance. The goal is to reduce complexity while increasing communications performance. This can only be achieved when architectural know-how is taken into consideration in early phases of the system design.

The basic idea of the joint design approach is to reduce the search possibilities of the MIMO-APP detection. This can be achieved by a special design of the bit interleaver (Sect. 8.3.1) or by a dedicated code design (Sect. 8.3.2), which in part was published in [14, 15]. All channel codes used for the examples here are LDPC codes, however, it is possible to use the presented idea as well for turbo codes and convolutional codes.

The LDPC codes used in this section are also described by a parity check matrix $\boldsymbol{H}_c$ and fulfill $\boldsymbol{H_c x^T = 0}$. Note that the parity check matrix is here denoted with subscript c to make it distinguishable from $\boldsymbol{H}$, the channel matrix. The parity check matrix $\boldsymbol{H}_c$ has N_c columns and M_c rows and has to be of full rank. The parity check matrix can be described by two layers with

$$\boldsymbol{H}_c = \begin{pmatrix} \boldsymbol{H}_g \\ \boldsymbol{H}_e \end{pmatrix} = \begin{pmatrix} \boldsymbol{H}_g & & \\ \boldsymbol{H}'_e & \cdots & 0 \\ & \ddots & \\ 0 & & 0 \\ 0 & \cdots & \boldsymbol{H}'_e \end{pmatrix} \tag{8.30}$$

The first layer $\boldsymbol{H}_g$ is a sparse parity check matrix, while the second layer $\boldsymbol{H}_e$ defines multiple, unconnected sub-codes. Each sub-code $\boldsymbol{H}'_e$ has a codeword length of $N'_e \leq M_T Q$.

As mentioned before, each transmission vector $\boldsymbol{s}_t$ carries the information of $M_T Q$ bits. For the transmission it has to be guaranteed that all bits of a sub-code $\boldsymbol{H}'_e$ are transmitted within one transmission vector. Thus each transmission vector carries an embedded code $\boldsymbol{H}'_e$. Embedded code or sub-code are used as synonyms in the following.

$\boldsymbol{H}_g$ has the task to connect all embedded codes. The sparse layer $\boldsymbol{H}_g$ can be described by a degree distribution (f_g,g_g). Where f_g represents the degree distribution of the variable nodes of the layer $\boldsymbol{H}_g$ and g_g defines the degree distribution of the check nodes respectively. The description of the second layer $\boldsymbol{H}_e$ can be done by defining one embedded code $\boldsymbol{H}'_e$.

The graph structure of such an LDPC code is shown in Fig. 8.9. In this graph 4 symbol nodes of the transmission vector are connected to 8 variable nodes. These are linked to one embedded code $\boldsymbol{H}'_e$, here, two check nodes. Each symbol node represents the information of one modulated symbol. Assuming a 4×4 antenna system the received transmission vector $\boldsymbol{y}_t$ comprises the information of four symbols. The major advantage will be that the embedded constraints reduce the complexity of the

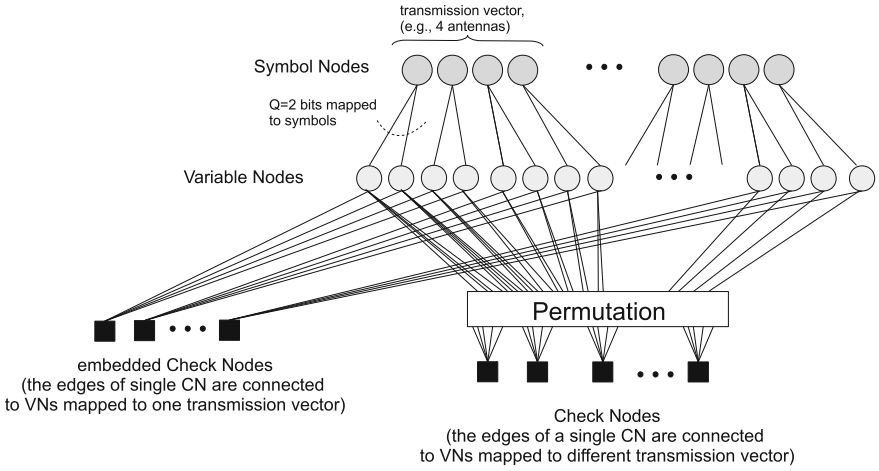


Fig. 8.9 Generic graph structure for an LDPC code with symbol nodes connected to embedded codes. One symbol node represents the information of one modulated symbol

MIMO-APP detection while implicitly solving parts of the channel code. We will see that this will reduce the overall complexity while even a better overall communications performance can be achieved.

The most simple constraint on a sequence of bits is a single parity check constraint, which means $\boldsymbol{H}'_e$ results in a single parity check code. For MIMO-APP detection a new decision tree with one embedded check node results, which is shown in Fig. 8.10. Again two different decision trees are shown, both with one embedded single parity check node constraint. Figure 8.10 top represents four transmit antennas ($M = 4$) and BPSK modulation ($Q = 1$), the lower figure represents the decision tree for two transmit antennas and $Q = 2$ bits per symbol.

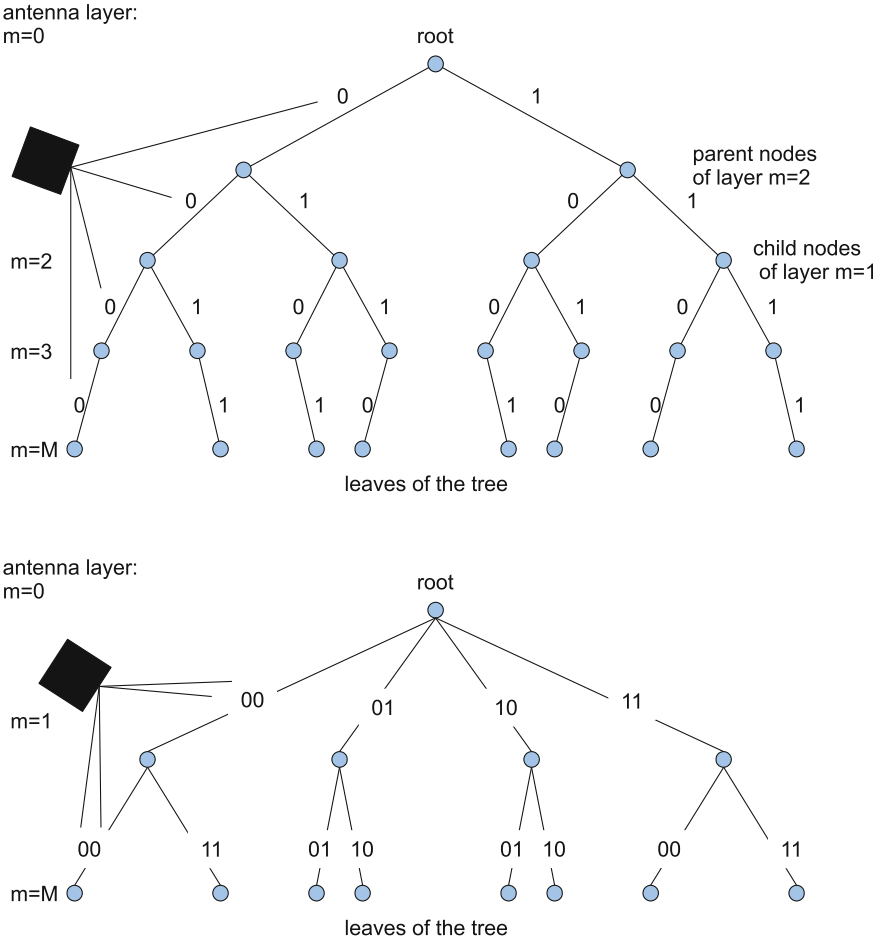


Fig. 8.10 Reduced decision tree with one embedded single parity check node

The check constraint (black square) in both cases is linked to all 4 bits which are simultaneously transmitted. This check eliminates paths in the decision tree, since the last bit has to fulfill the parity check equation. Thus the MIMO-APP demodulation Eq. 8.18 changes to

$$\lambda(x_{q,m}) \approx \min_{\forall \boldsymbol{s}|\boldsymbol{c_e}, x_{q,m}=0} \left\{ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}) \right\} - \min_{\forall \boldsymbol{s}|\boldsymbol{c_e}, x_{q,m}=1} \left\{ \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{s}\|^2 - N_0 \sum_{\forall q',m'} \ln P(x_{q',m'}) \right\} \tag{8.31}$$

with the major difference of the term $\boldsymbol{s}|\boldsymbol{c_e}, x_{q,m} = 0$, which means, the currently observed $\boldsymbol{s}$ is conditioned on bit $x_{q,m}$ and $\boldsymbol{c_e}$. Each observed bit has to be an element of a valid codeword $\boldsymbol{c_e}$, while a valid codeword is defined via the embedded code constraint $\boldsymbol{H}'_e \boldsymbol{c_e}^T = 0$. Thus, we reduce the search space of the MIMO-APP demodulation while implicitly solving the second layer of $\boldsymbol{H}_c$ during the MIMO-APP demodulation. If we embed one single parity check equation, the overall possibilities for demodulation downscales to $P = 2^{MQ-1}$. With q parity checks embedded within one transmission vector the number of possibilities for MIMO demodulation is reduced to $P = 2^{MQ}/2^q = P = 2^{MQ-q}$.

It is important to distinguish the complexity reduction of the presented codes and the complexity reduction caused by algorithmic techniques. All algorithmic transformations, which are published for tree based MIMO-APP decoding, can be applied for the presented approach as well. In the following we will present two examples of how to enable the embedding of code constraints.

- Example one utilizes a standard WiMAX LDPC code (Sect. 8.3.1). By defining a well chosen bit interleaver we can embed parts of the defined parity checks within the transmission vector. The resulting BICM-MIMO will show a better communications performance while reducing the complexity of sphere decoding.
- Example two describes the design of quasi-cyclic LDPC codes which can be decoded by a standard compliant LDPC code decoder. Furthermore, the presented LDPC codes can largely decrease the search space of a sphere detector (Sect. 8.3.2).

8.3.1 Sphere Decoder Aware Bit Interleaver Design

The joint design approach as presented in the previous section enables us to design an elaborated interleaver for the BICM-MIMO system which will decrease the complexity of the sphere detection while improving the communications performance.

This approach works for all LDPC codes which are quasi-cyclic. The resulting interleaver will be a quasi-cyclic interleaver. The motivation of this section is that we can reduce complexity and improve communications performance by simple

derivations from existing communications standards. The basic design method is explained by using one specific WiMAX LDPC code as an example. The following parity check matrix represents a WiMAX LDPC code with codeword length of $N = 576$ bits and code rate $R = 1/2$:

$$H^{Macro} = \left(\begin{array}{cccccccccccccccccccccc}
0 & 24 & 19 & 0 & 0 & 0 & 0 & 0 & 14 & 21 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 7 & 0 & 0 & 0 & 6 & 20 & 3 & 0 & 0 & 0 & 4 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 6 & 21 & 0 & 9 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
16 & 0 & 12 & 0 & 0 & 0 & 0 & 0 & 17 & 7 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 10 & 0 & 0 & 0 & 22 & 0 & 0 & 11 & 19 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 12 & 11 & 0 & 21 & 0 & 0 & 0 & 20 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 24 & 14 & 0 & 0 & 0 & 0 & 0 & 4 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 3 & 19 & 0 & 0 & 0 & 1 & 0 & 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
4 & 0 & 0 & 0 & 21 & 7 & 0 & 11 & 0 & 0 & 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 24 & 0 & 15 & 0 & 0 & 18 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 2 & 17 & 0 & 0 & 0 & 0 & 10 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
11 & 0 & 0 & 0 & 0 & 17 & 0 & 11 & 0 & 0 & 0 & 7 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right) \tag{8.32}$$

As described in Chap. 7 each entry in this matrix indicates a $z \times z$ sub-matrix, here with $z = 24$. In case an entry is one or greater it gives the amount of cyclic right shifts of a permuted identity matrix. Zero sub-matrices are indicated by the zero entries.

Goal is to ensure the mapping of parity check constraints to transmission vectors which can be achieved by the bit interleaver of the BICM system. Here, the design of one interleaver is presented for a 4×4, 64-QAM system since this fits well to the 24 columns of the WiMAX macro matrix.

In this example we can ensure that always two check nodes are mapped to one transmission vector by using a cyclic block interleaver. A cyclic block interleaver can be described by a vector in which each entry defines the offset value for writing, see Sect. 4.3. The idea of the cyclic block interleaver is to reverse the permutation index of the last two groups, such that, the last two rows are rotated back to identity matrices. Thus, embedded sub-codes $\boldsymbol{H}'_e$ will be obtained each consisting of two parity checks and each of these are allocated to transmission vectors. Note, that is possible with any type of quasi-cyclic LDPC code to ensure at least one single parity check code to be embedded in a transmission vector. In this example we cannot ensure a third parity bit within a transmission vector since it is not possible to guaranty that the resulting sub-codes $\boldsymbol{H}'_e$ are unconnected across sphere detectors.

The resulting cyclic block interleaver has a dimension of $C_1 = 24$ columns and $C_2 = 24$ rows. The corresponding (negative) offset values for writing the columns are:

$$\boldsymbol{I}^{offset} = \begin{pmatrix} -11 & 1 & 1 & -17 & 1 & -17 & 1 & -11 & -10 & -13 & 1 & -7 & -2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{8.33}$$

The negative entries here indicate the negative offset values at which position a column is started to be filled. A 1 indicates that we fill the corresponding column of the block interleaver regularly, i.e., from top to bottom. The corresponding offset

values in $\boldsymbol{I}^{offset}$ are the reversed permutation of the quasi-cyclic entries with respect to the last two groups of Eq. 8.32.

Figure 8.11 shows the open loop communications performance of a 64-QAM, 4×4 antennas system. All curves use WiMAX LDPC codes, either with 10 or 30 iterations. The communications performance labeled with default sphere uses the bit interleaver defined in the WiMAX communications standard. The sphere decoder has to search through 2^{24} branches of the tree. The improved communications performance is obtained by using the described bit to transmission vector mapping. The performance gain is up to 0.5 dB. The search space of the sphere detector is reduced by a factor of four (2 parity checks embedded).

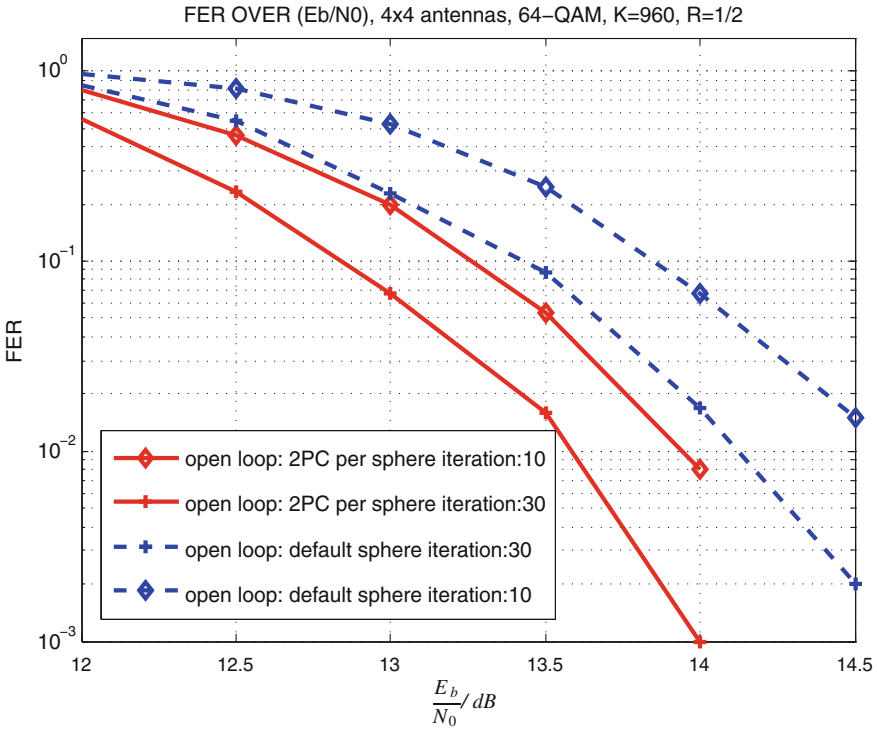


Fig. 8.11 Open loop communications performance, ergodic channel 64-QAM, 4×4 antennas, both utilizing WiMAX LDPC codes. Improved communications performance and reduced complexity by a well chosen bit to transmission vector mapping. Note that the channel code is equivalent in both cases, while the number of leaves was reduced by a factor of four by embedding two parity checks (PC)

8.3.2 Sphere Decoder Aware LDPC Code Design

The second design example shows the design of LDPC codes with a special focus on the complexity reduction of the MIMO detector when using a tree search algorithm. Whenever considering a new design of an LDPC code it is beneficial when the new channel code can be processed by standard LDPC decoder architectures. Since nearly all wireless communications standards rely on quasi-cyclic LDPC codes we restrict the design example of this section to this type of codes. Here, only the basic idea is presented to introduce the potential of the approach of joint design of algorithm and architecture. Further results and how to derive the parity check matrix of the channel code are presented in [14, 15].

The transmission system assumed for the design example is a 4×4 antenna system using a 16-QAM transmission scheme. The bit interleaver used in the example is a classical block interleaver with 16 columns and z rows. The number of rows of the block interleaver depends on the size of the identity matrix. In the following we assume one particular LDPC code with a block length of $N = 1920$ bits and a code rate of $R = 0.5$.

$$H^{Macro} = \left(\begin{array}{cccccccccccccccc} 64 & 0 & 0 & 119 & 50 & 53 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 34 & 70 & 0 & 0 & 66 & 27 & 49 & 63 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 74 & 82 & 0 & 86 & 64 & 80 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 76 & 71 & 15 & 117 & 111 & 101 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right) \tag{8.34}$$

The macro matrix with (z = 120) shown here is one realization which allows to embed four check nodes within each transmission vector. This is indicated by the separator between the top four and bottom four rows of the macro matrix. With the four rows all using identity matrices without permutation we can directly identify the embedded sub-code $\boldsymbol{H}'_e$.

The block interleaver ensures always the correct codeword bit to transmission vector mapping. The bit positions which have to be mapped to the first transmission vector are $[0 \quad z-1 \quad 2z-1 \quad \ldots \quad 15z-1]$ for the second transmission vector $[1 \quad z \quad 2z \quad \ldots \quad 15z]$, and so on.

The communications performance results with respect to an ergodic channel and 16QAM 4×4 system are shown in Fig. 8.12. The figure shows the open loop and the closed loop performance of the new LDPC code using embedded codes in comparison to the WiMAX simulations already presented in Fig. 8.7. Both schemes—WiMAX LDPC codes and joint LDPC code design—use 4 outer and 5 inner channel code iterations in the closed loop case. In the open loop case 20 LDPC iterations (layered) are performed. In both cases, the simulated performance of the open loop system as well as that of the closed loop system is better when using the joint design approach compared to the original WiMAX scheme.

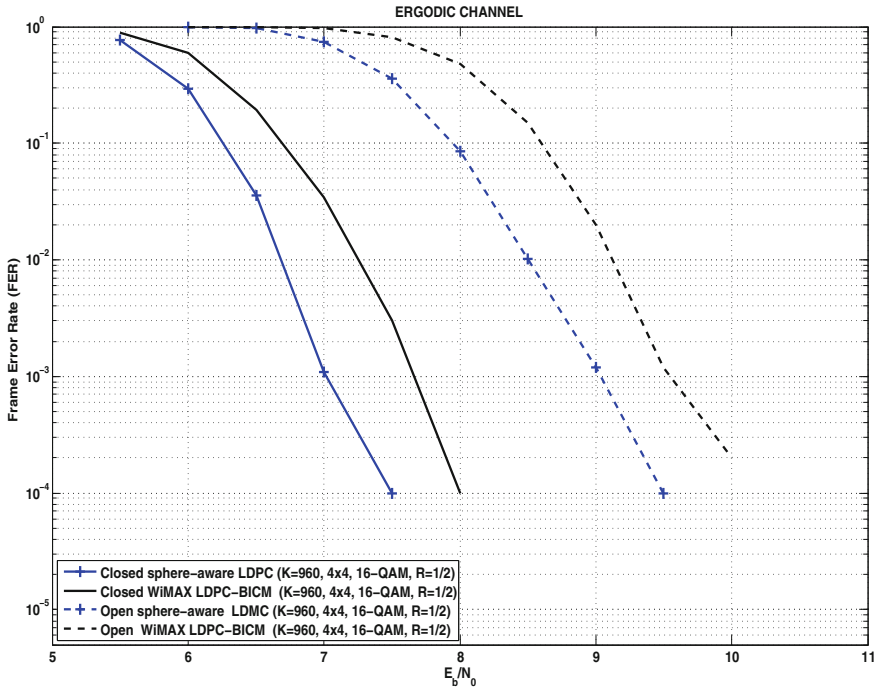


Fig. 8.12 Simulated communications performance (ergodic, 4×4, 16QAM) of an open loop system and a closed-loop system. The graph compares the original WiMAX code against a new LDPC code design. The new LDPC code can be processed by a WiMAX LDPC decoder architecture while reducing the size of the search tree for MIMO-APP detection b a factor of 16

In addition, when utilizing the new LDPC code design and a block interleaver the resulting tree for MIMO detection has only 4096 branches. Thus, the search tree for the sphere detector is reduced by a factor of 16 compared to standard case using WiMAX LDPC codes.

In summary the most important points when designing LDPC codes which use the knowledge of the sphere detector are:

- The properties of quasi-cyclic LDPC codes are used which enables the processing by standard decoder architectures.
- Parts of the channel code are implicitly solved during the MIMO-APP demodulation.
- The size of the search tree for MIMO-APP demodulation can be reduced by over a factor of ten.

- The achieved communications performance can be better than that of state-of-the-art BICM-MIMO schemes for open loop and closed loop simulations.

References

1. Gans, M.J., Amitay, N., Yeh, Y.S., Xu, H., Valenzuela, R.A., Sizer, T., Storz, R., Taylor, D., MacDonald, W.M., Tran, C., Adamiecki, A.: BLAST system capacity measurements at 2.44 GHz in suburban outdoor environment. In: Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd, vol. 1, pp. 288–292 (2001). doi:10.1109/VETECS.2001.944850
2. Alamouti, S.M.: A simple transmit diversity technique for wireless communications. IEEE J. Sel. Areas Commun. **16**(8), 1451–1458 (1998)
3. Hochwald, B., ten Brink, S.: Achieving near-capacity on a multiple-antenna channel. IEEE Trans. Commun. **51**(3), 389–399 (2003). doi:10.1109/TCOMM.2003.809789
4. Vikalo, H., Hassibi, B., Kailath, T.: Iterative decoding for MIMO channels via modified sphere decoding. IEEE Trans. Wirel. Commun. **3**(6), 2299–2311 (2004)
5. Paulraj, A.J., Gore, D.A., Nabar, R.U., Bölcskei, H.: An overview of MIMO communications—a key to Gigabit wireless. In: Proceedings of IEEE **92**(2), 198–218 (2004). doi:10.1109/JPROC.2003.821915
6. Gimmler, C.: Dissertation in preparation. Ph.D. thesis, Department of Electrical Engineering and Information Technology, University of Kaiserslautern (2013)
7. ten Brink, S., Kramer, G., Ashikhmin, A.: Design of low-density parity-check codes for modulation and detection. IEEE Trans. Commun. **52**(4), 670–678 (2004). doi:10.1109/TCOMM.2004.826370
8. Mller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., Wehn, N.: A novel LDPC decoder for DVB-S2 IP. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09), pp. 1308–1313 (2009)
9. Alles, M., Berens, F., Wehn, N.: A synthesizable IP core for WiMedia 1.5 UWB LDPC code decoding. In: Proceedings of IEEE International Conference on Ultra-Wideband ICUWB 2009, pp. 597–601. Vancouver, Canada (2009)
10. May, M., Ilnseher, T., Wehn, N., Raab, W.: A 150Mbit/s 3GPP LTE turbo code decoder. In: Proceedings of Design, Automation and Test in Europe, 2010 (DATE '10), pp. 1420–1425 (2010)
11. Sun, Y., Cavallaro, J.: Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder. Integr. VLSI J. (2010). doi:10.1016/j.vlsi.2010.07.001
12. Urard, P., Yeo, E., Paumier, L., Georgelin, P., Michel, T., Lebars, V., Lantreibecq, E., Gupta, B.: A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes. In: Proceedings of IEEE International Conference on Digest of Technical Papers Solid-State Circuits ISSCC, 2005, pp. 446–609 (2005). doi:10.1109/ISSCC.2005.1494061
13. Gimmler-Dumont, C., Kienle, F., Wu, B., Masera, G.: A system view on iterative MIMO detection: dynamic sphere detection versus fixed effort list detection. VLSI. Des. J. Article ID 826350 (2012). doi:10.1155/2012/826350
14. Kienle, F.: Low-density MIMO codes. In: Proceedings of 5th International Symposium on Turbo Codes and Related Topics, pp. 107–112. Lausanne, Switzerland (2008)
15. Kienle, F.: On low-density MIMO codes. In: Proceedings of IEEE International Conference on Communications ICC '09, pp. 1–6 (2009). doi:10.1109/ICC.2009.5199242

Chapter 9
Comparing Architectures

As mentioned in the previous chapters, todays high-end smart phones have to support multiple radio standards, advanced graphic- and media applications and many other applications resulting in a workload of about 100 giga operations per second in a power budget of 1 W [1]. The baseband processing in the radio part (mainly front-end processing, demodulation and decoding) requires more than 50 % of the overall workload in a state-of-the-art 3.5G smart phone. To achieve higher spectral efficiency new transmission techniques like MIMO will be established, see Chap. 8. However, this will increase the workload even further. Thus, there is a strong need for *efficient* wireless baseband receivers. The overall efficiency of a baseband receiver depends on

- *communications efficiency*: expressed by the spectral efficiency and signal-to-noise ratio (SNR). The requirements on the communications efficiency have the largest impact on the selected baseband processing algorithms.
- *implementation efficiency*: related to silicon area, power and energy. Here, the energy efficiency is the biggest challenge due to the limited available battery power in many devices.
- *flexibility*: in software defined radio, receivers should be flexible/configurable at run-time since they have to support multiple standards. There are various silicon implementation styles ranging from general purpose architectures, over DSPs and ASIPs down to fully physically optimized IP blocks which strongly differ in their implementation efficiency but also in their flexibility. Thus, for each building block of the receiver a detailed analysis of algorithmic flexibility and service parameter flexibility requirements has to be carried out. Service parameter flexibility for a specific channel code is for example varying code rates, block sizes with respect to a given throughput. The goal is to find an optimal implementation style with respect to a flexibility/cost trade-off. Thus, advanced baseband receivers are heterogeneous multi-core architectures implemented in different design styles.

System requirements are very often specified by communication standards such as UMTS, LTE and WiMAX, which define different services in terms of required communications performance and system data throughput, i.e. information bits per

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3_9, © Springer Science+Business Media New York 2014

second. To obtain an efficient baseband implementation, a careful and elaborate *design space exploration* has to be performed. This is a very challenging task due to the size and the multi-dimensionality of the design space as shown for turbo decoding and LDPC decoding in Sects. 6.3.2 and 7.4 respectively. Therefore it is mandatory to prune the design space in an early stage of the design process. In this process the algorithms have to be selected, optimized, and quantitatively compared to each other with respect to their system performance and implementation efficiency.

Appropriate metrics are key for efficient design space exploration to measure the algorithmic and the implementation complexity, respectively. Parts of this chapter was presented in [2].

Algorithmic Complexity

There exists no universal measure for complexity. The huge field of complexity theory and the resulting 'O' notation is mainly focused towards software implementation and cannot be adapted in a straight forward fashion to hardware implementation. The algorithmic complexity, with respect to hardware implementation, has to be tailored to its application space, in this case the communication system.

Thus, it is important to understand the application under consideration. From a communication system point of view we can separate digital processing in the baseband into two parts: the so called 'inner modem' and 'outer modem' [3], respectively. The task of the inner modem is the extraction of symbols from the received signal waveform, i.e., equalization, channel estimation, interference cancellation and synchronization. The outer modem performs demodulation, de-interleaving and channel decoding on the received symbols. A large diversity exists in the various baseband processing algorithms with respect to operation types, operation complexity, and data types especially between the inner and out modem. Thus, algorithmic complexity in baseband processing should be separately analyzed for the inner and the outer modem, respectively.

A useful description of complexity for our purpose is to use the number of 'algorithmic' operations which have to be performed per received samples by the algorithms of a baseband receiver. This complexity metric has the advantage of being independent of a specific implementation of the algorithms. Based on this complexity definition a two-dimensional graph can be set up in which the horizontal axis corresponds to the sample rate of the receiver (which is proportional to the data rate) and the vertical axis corresponds to the operations per sample which have to be carried out. Typically, both axes are scaled logarithmically.

van Berkel determined the complexity of various algorithms in baseband processing. In his remarkable and comprehensive article [1] he focused on the number of 'algorithmic' operations which have to be performed per received bit by the algorithms of a baseband receiver for different communication standards. As a consequence the algorithmic complexity for the inner and outer modem is measured in giga operations per second (GOPs). Using this metric it can be seen that sophisticated decoding schemes like turbo and LDPC codes have a much larger

complexity—measured in GOPs—than the algorithms of the inner modem. Eberli from ETH Zürich uses a similar metric [4] for measuring complexity in baseband processing. The importance of appropriate metrics to compare forward error correcting codes is recently shown in [5]. Two improved metrics are suggested, one metric is based on operations the other metric is based on the required storage demands per decoded bit. The major weakness of these metrics is the missing link between both metrics which prevents an application to the resulting implementation complexity.

Implementation Complexity

Often implementation complexity is derived from algorithmic complexity in a straight forward manner. For example, Eberli [4] uses a cost factor for each algorithmic operation which reflects its implementation cost. In this way we can derive area efficiency and energy efficiency which is even more important for baseband receivers.

Graph representations for energy and area efficiency are commonly used for design space exploration:

- A two dimensional energy efficiency graph: one axis corresponds to the algorithmic complexity, e.g. measured in *GOPs*, and the other axis to the power, e.g. measured in mW, consumed when providing the corresponding operations/second. Each point in this graph describes the *energy efficiency metric* of a given implementation, i.e. *operations/second/power unit*, usually measured in *GOPs/mW*. Since energy corresponds to power multiplied with execution time, each point gives the *operations/energy* measured in $operations/Joule$.
- In a similar way we can set up an area efficiency graph in which one axis represents the needed area. Each point in this graph yields the *area efficiency metric*, i.e. *operations/second/area unit*, usually measured in $GOPs/mm^2$.

Note that the energy and area efficiency for the same algorithmic complexity can vary by several orders of magnitude, depending on the selected implementation style. By far the highest energy efficiency is achieved by physically optimized circuits, however, at the expense of limited algorithmic and service parameter flexibility. The highest flexibility, at the expense of low energy efficiency, is achieved by programmable digital signal processors. As said the designer has to find a compromise between the two conflicting goals by trading off flexibility versus energy efficiency. However, in contrast to area and energy efficiency, flexibility is hard to quantify. The optimum design point has thus to be understood qualitatively. It depends on the application and a large number of economic and technical considerations. We can combine energy and area efficiency in a two dimensional design space in which the two axes correspond to area efficiency and energy efficiency, respectively. This is a well known representation of the design space.

Assessment of Metrics

Area, throughput and especially energy in many system-on-chip implementations are dominated by data-transfers and storage schemes [6] and not by the computations themselves. However, common metrics as described above focus solely on operations, and do not consider data-transfer and storage issues at all. Thus, these metrics are only valid if the operations dominate the implementation complexity. This is the case in data-flow dominated algorithms like an FFT calculation, correlation or filtering which are dominating the algorithms in the inner modem.

However, the *channel decoding algorithms* in the outer modem largely differ from the algorithms used in the inner modem. Here, the operations to be performed are non-standard operations (e.g. tanh) using non-standard data types (e.g. 7 bit fix-point). But more importantly, the overall implementation complexity, especially energy, is dominated by data-transfers and storage schemes.

The transitions from 3G to LTE advanced require an improvement of two orders of magnitude in energy efficiency. This improvement will come from technology scaling to a small extent only [7]. A general trend towards co-design of algorithm and architecture can be seen in new standards. Since the traditional separation of algorithm and architecture design leads to suboptimal results.

In channel decoding the co-design focuses on data-transfer and storage schemes [8–10]. Examples with respect to communication standards are special interleavers for turbo codes (e.g. LTE standard [11]) and special structures of the parity check matrix for LDPC codes (e.g. DVB-S2 standard [12]). These special structures allow an efficient parallel implementation of the decoding algorithm with small overhead in data-transfer and storage. GOPs based metrics do not at all reflect such specific structures.

Another important issue is flexibility. Flexibility on the service parameter side, e.g., code rates and block sizes in the case of channel decoding, have a large impact on the implementation complexity. The cost of the overhead introduced by flexibility is normally not considered by looking only at the operations of the algorithm.

In summary, efficiency metrics based on GOPs only are questionable. Particularly, for non-data flow dominated algorithms, as they entirely neglect important issues like data and storage complexity, algorithm/architecture co-design and flexibility.

In this chapter we focus on channel decoding as application. The contributions of this chapter are:

- We show that metrics based on GOPs can lead to wrong conclusions.
- We introduce meaningful and suitable metrics for energy and area efficiency.
- We present an approach for design space exploration based on these metrics.

9.1 Reference Designs

Reference designs are key to assess various metrics. Many publications on VLSI implementations exist for turbo decoders, e.g. [13–18] and LDPC decoders, e.g. [19–23]. However, performing an objective assessment of metrics requires detailed and complete information about the implementation cost and the communication performance. However, performing an objective assessment of metrics requires all information about the implementation data are mandatory together with the communications performance data. Moreover an identical implementation technology is mandatory since normalizing different technologies is in technologies below 90 nm very error-prone. The difficulty of comparing different publications is shown in Sect. 9.4 after deriving suitable metrics for comparison.

For the sake of demonstration, 5 different channel decoder implementations are selected which were designed by the research group AG Wehn, University of Kaiserslautern in the last couple of years. All test benches, fixed point models, and technology related data are completely available for the comparison. Every decoder is designed with the same design methodology: hand optimized VHDL code and synthesized using Synopsis Design Compiler and in all cases many years of application experience proved via industry projects. The decoders differ in services (throughput, block sizes, code rates), decoding algorithms, and implementation styles. Selected channel codes are convolutional codes, turbo codes and LDPC codes which covers a large spectrum of channel codes used in todays standards. The 5 different decoders are:

- An application specific instruction set processor (ASIP) [24] capable of processing binary turbo codes, duo-binary turbo codes and various convolutional codes (CC) with varying throughputs dependent on code rate and decoding scheme.
- A turbo decoder, which is LTE [11] compliant. The maximum throughput is 150 Mbit/s at 6.5 decoding iterations.
- An LDPC decoder optimized for flexibility, supporting two different decoding algorithms, code rates from R $= 1/4$ to 9/10 and a maximum block length of 16384.
- An LDPC decoder which is WiMedia 1.5 compliant and optimized for throughput, supporting code rates from R $= 1/2$ to 4/5 with two block lengths N $= 1200$ and N $= 1320$ bits [25].
- A convolutional decoder with 64-states which is WiFi [26] compliant.

All decoders are synthesized on a 65 nm CMOS technology under worst case PVT conditions with $V_{dd} = 1.0$ V, 120 °C. Power estimations are based on nominal case $V_{dd} = 1.1$ V. Table 9.1 gives an overview of the key parameters of the different decoders. P&R indicates that the corresponding data are post-layout data. The payload (information bits) throughput depends on the number of decoding iterations for turbo and LDPC codes which also impacts the communications performance. Thus, the throughput is specified depending on the number of iterations.

We chose our own designs as reference designs, as these were the only designs for which we had complete access to all information, which is necessary for a fair

Table 9.1 Reference decoders: service parameters and implementation results in 65 nm technology, including area and dynamic power consumption

Channel decoder	Flexibility	Max. size	Throughput (Mbit/s)	Frequency (MHz)	Area (mm^2)	Power (mW)
ASIP [24]	CC Binary TC Duo-binary	N = 16 k	40 (64-state) 14 (6 iter) 28 (6 iter)	385 (P&R)	0.7 (P&R)	∼100
LTE TC [27]	R = 1/3 to R = 9/10	N = 18 k	150 (6.5 iter)	300 (P&R)	2.1 (P&R)	∼300
Flexible LDPC	R = 1/4 to R = 9/10	N = 16 k	30 (R = 1/3 40 iter) 100 (R = 1/2 20 iter) 300 (R = 5/6 10 iter)	385 (P&R)	1.17 (P&R)	∼389
LDPC [25] WiMedia 1.5	R = 1/2 to R = 4/5	N = 1.3 k	640 (R = 1/2 5 iter) 960 (R = 3/4 5 iter)	265	0.51	∼193
CC decoder	64-state NSC		500	500	0.1	∼37

assessment of metrics. This does not limit the validity of the statements and conclusions made in this chapter.

In Table 9.2 we show the number of algorithmic operations required to process the different types of convolutional codes, turbo codes and LDPC codes. Bit-true C-reference models are used for operation counting. In all algorithms of Table 9.2 the operations are fixed point additions with varying bit width. We normalized all operations to an 8 bit addition. The number of operations is related to information bits which have to be decoded. The total number of operations, which have to be performed per second, depends on the code rate R and throughput, which in turn depends on the number of iterations for LDPC and turbo code decoding. Two different

Table 9.2 Number of normalized algorithmic operations per decoded information bit for different channel decoders dependent on throughput and code rate R

Channel code	# iterations	GOPs (w.r.t. throughput)		
		100 Mbit/s	300 Mbit/s	1 Gbit/s
CC (64-state)		20	60	200
LDPC Min-Sum	5	7.5/R	22.5/R	75/R
	10	15/R	45/R	150/R
	20	30/R	90/R	300/R
	40	60/R	180/R	600/R
LDPC Min-Sum λ-3-Min	10	50/R	150/R	500/R
	20	30/R	90/R	300/R
	40	200/R	600/R	2000/R
3GPP Turbo (Max-Log)	2	28	84	280
	4	56	168	560
	6	84	252	840

algorithms for LDPC codes are utilized in our LDPC decoders. Both algorithms are suboptimal algorithm approximating the belief propagation algorithm: the Min-Sum algorithm with a scaling factor and the λ-3-Min algorithm [28] which is a more accurate approximation. However, the latter needs about 3.3 times more operations. This more accurate approximation is required when lower code rates $R < 0.5$ have to be supported, as for example in DVB-S2 decoders [21]. The flexible LDPC decoder supports both decoding algorithms, the WiMedia LDPC decoder is based on the Min-Sum algorithm only, see Sect. 5.2.3.

9.2 Suitable Metrics

Figure 9.1 shows the two dimensional design space for our channel decoders, covering area and energy efficiency based on operations. One axis represents the area efficiency ($GOPs/mm^2$), and the other axis the energy efficiency (op/pJ). In this graph, efficient architectures w.r.t. area and energy are located in the upper right corner. Less efficient architectures are placed in the lower left corner.

The convolutional decoder seems to be the most efficient decoder while the ASIP shows to be the decoder with the lowest efficiency. An interesting observation is the efficiency of the flexible LDPC decoder. The efficiency largely increases in both directions (area, energy) when replacing the Min-Sum by the λ-3-Min algorithms which is the more complex algorithm in terms of operations. As described in the previous section the GOPs for this algorithm increases by a factor 3.3.

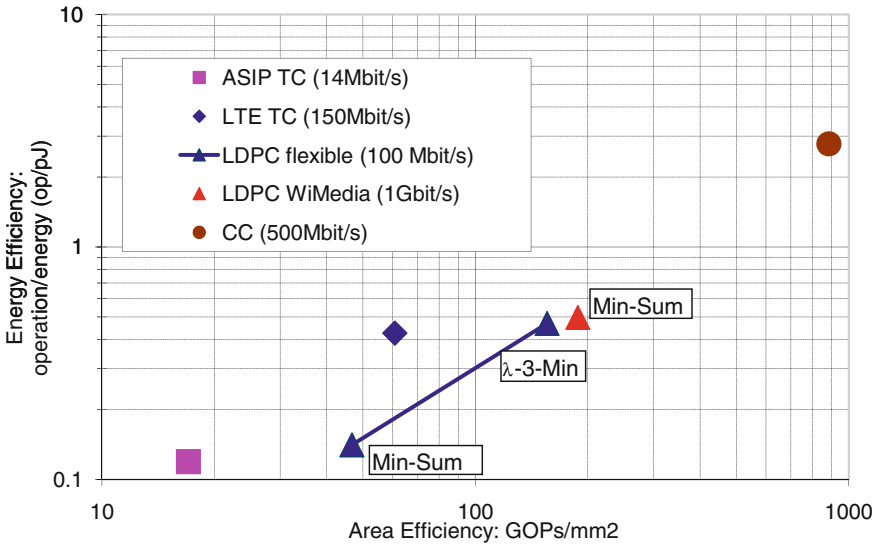


Fig. 9.1 Energy efficiency versus area efficiency based on operations

Furthermore, we see that the λ-3-Min based flexible LDPC decoder has nearly the same efficiency as the less flexible WiMedia decoder which is optimized for high throughputs. This observation seems to be contradictory, we would expect that a less flexible decoder, optimized for throughput, has a much higher efficiency compared to a highly flexible decoder architecture. This aspect shows already the problem with GOPs based metrics.

The problem with GOPs based metrics will become more obvious after introducing appropriate metrics which resolve the aforementioned anomaly. Instead of referring to operations per task we refer to the number of decoded information bits. Metrics normalized to the number of information bits allow to compare competing architectures for a given algorithm since the efficiency metrics are independent of the specific operations and data types used to execute the algorithm. All implementation issues like data-transfer and storage are taken into account since the metrics are oblivious of how the task has been executed. Such a metric considers the result only and not how the result is calculated. Furthermore, a metric normalized to the number of information bits allows to compare different coding schemes as a function of the communication parameters (modulation, signal to noise ratio, bandwidth). In particular, iterative decoding algorithms can be compared in a meaningful way to non-iterative algorithms.

We define two metrics for implementation efficiency as follows

- energy efficiency metric: *decoded information bit per energy* measured in bit/nJ
- area efficiency metric: *information bit throughput per area unit* measured in $Mbit/s/mm^2$

We mapped our decoders to the design space which is based on these metrics. The result is shown in Fig. 9.2. Again efficient architectures are in the upper right corners, inefficient architectures are in the lower left corner.

A large change in the relative and absolute positions can be observed for some decoders, when comparing the graph to Fig. 9.1 which was based on GOPs metrics.

- The difference in the efficiency between the two instances of the flexible LDPC decoder (Min-Sum and the λ-3-Min decoder, respectively) is now much smaller. Moreover the Min-Sum decoder is more efficient than the other ones which was not the case in the conventional design space. This matches our expectations since the data-transfer and storage scheme in both decoders is nearly identical. The increase in computation results in only a small energy and area increase by about 10 %. Both flexible LDPC decoders are targeting the same throughput.
- The efficiency of the WiMedia decoder which is optimized rather for throughput than for flexibility, is now much larger than the efficiency of the flexible LDPC decoder which again matches our expectations.

To summarize, in applications which are dominated by data-transfers and storage schemes, like here in channel coding, the change in the number of operations in the algorithm has only a small impact on area and energy. Comparing Figs. 9.1 and 9.2 demonstrates clearly the problem with GOPs based metrics.

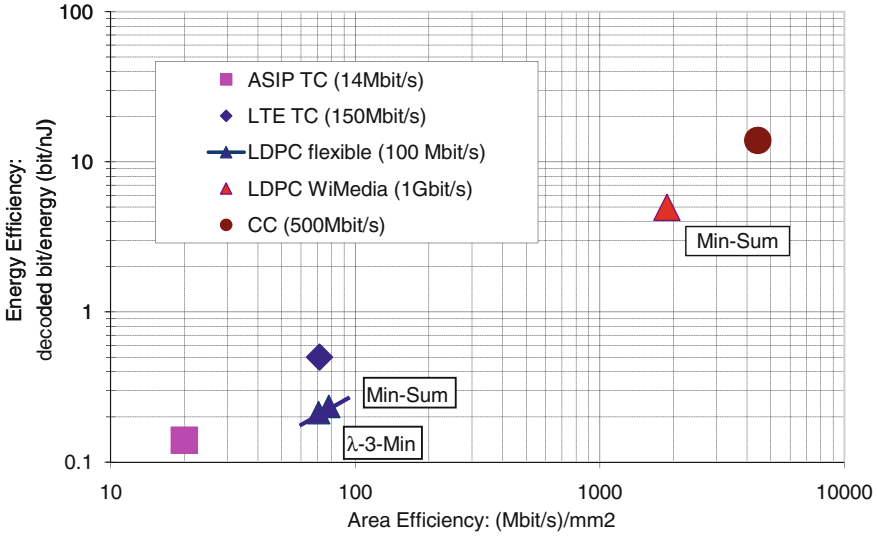


Fig. 9.2 Design space based on suitable metrics. Decoded information bit per energy over information bit throughput per area unit

So far we have focused on the implementation complexity but have not discussed the important aspect of flexibility and communication performance. In the following we will investigate the relationship between communication performance, algorithmic/service parameter flexibility and implementation efficiency.

9.3 Approach for Design Space Exploration

In the previous section we investigated the absolute and the relative positions of the different decoders to each other. However, equally important in this space is the shift of an efficiency point when specific communication parameters are changed.

The following relevant parameters will be considered: the frame error rates (FER), i.e. communications performance, coding techniques, code rates, number of iterations and throughput.

We present two case studies to demonstrate an appropriate approach for design space exploration. The resulting different values for area and energy efficiencies illustrate the strong dependency between communications performance and implementation efficiency.

The two case studies will show that a single quantitative measure to select the best channel code does not exist. The first case study is concerned with implementation efficiency and compares non-iterative decoding techniques (convolutional codes) to iterative decoding techniques (LDPC codes). The second case study compares

two different iterative decoding techniques (LDPC and Turbo codes) with code rate flexibility.

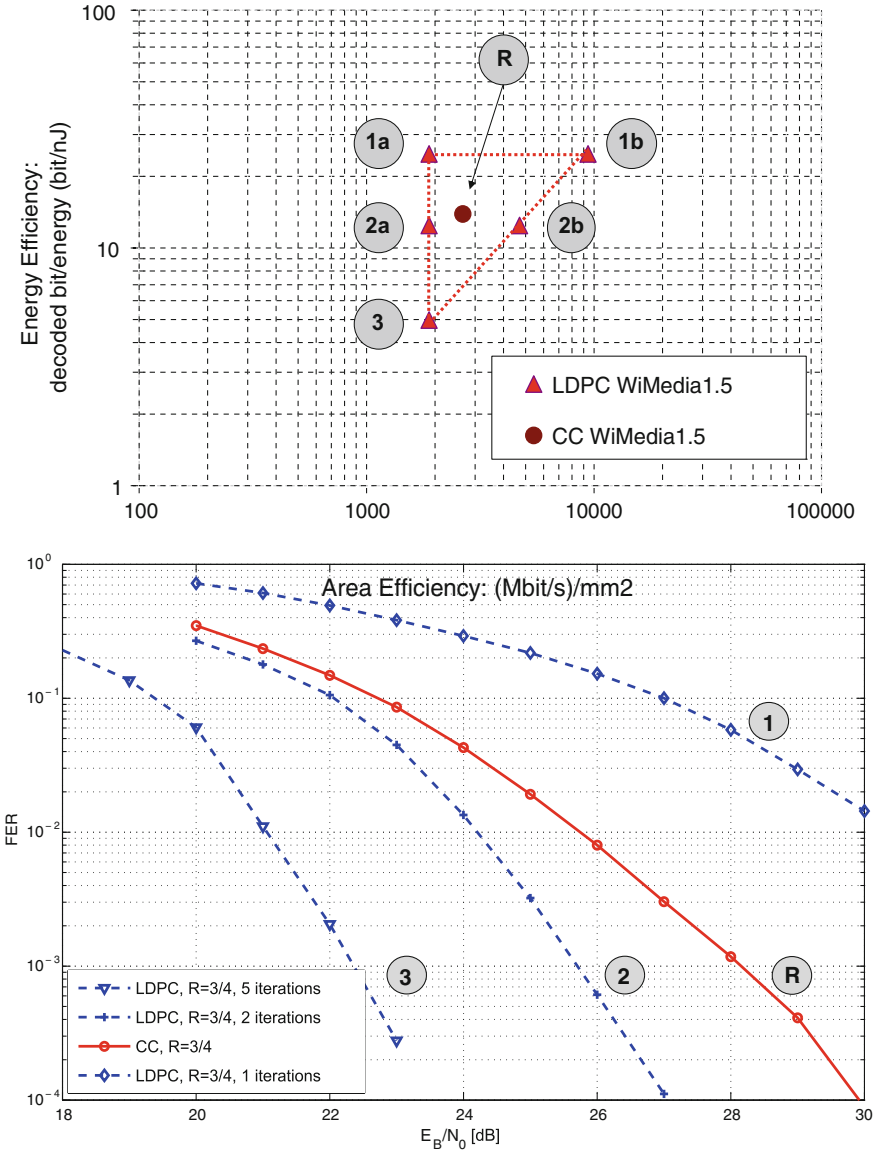
9.3.1 Implementation Driven Design Space Exploration

The first design study reflects the discussion performed during the WiMedia 1.5 standardization process. WiMedia [29] features low complexity devices for UWB communication. The older WiMedia 1.2 standard uses convolutional codes as channel coding technique which served as a reference design. LDPC codes were considered as a promising candidate which had to be compliant to the given service parameters like the throughput of 960 Mbit/s and the code rate $R = 0.75$. Thus, new LDPC codes were developed according to the code/architecture co-design approach [25] resulting in an LDPC decoder which has a much higher efficiency than e.g. a highly flexible LDPC decoder.

As already shown in Fig. 9.2 the area and energy efficiency of the resulting WiMedia 1.5 LDPC decoder is lower compared to a convolutional decoder. However, this comparison does not at all consider the communications performance. At most five decoding iterations can be performed by the LDPC decoder to comply with the throughput requirements of the standard. As already pointed out, the number of iterations strongly impacts the communications performance of the LDPC decoder. The frame error rates as a function of the number of iterations are contrasted with implementation efficiency in Fig. 9.3. Point 3 in the design space figure corresponds to the WiMedia 1.5 decoder when performing 5 iterations (this was the decoder assumption in the previous figures when we referred to the WiMedia LDPC decoder). The communication figure shows that this decoder has a 4 dB better communication performance than the convolutional decoder. The communication performance is comparable to that of the convolutional decoder if the LDPC performs only two iterations instead of five (case 2 in Fig. 9.3). Finally, executing only one iteration in the LDPC decoder results in a communication performance which is about 4 dB worse than the convolutional decoder (case 1 in Fig. 9.3).

Note the resulting set of points in the design space for the different cases. The different efficiency points in Fig. 9.3 are obtained by distinguishing two cases:

- The system throughput is not changed w.r.t. WiMedia 1.5. constraint (scenario a in Fig. 9.3). In this scenario only the energy efficiency is improved (points $3 \rightarrow 2a \rightarrow 1a$). Obviously the decoding time decreases when the decoder executes a smaller number of iterations resulting in a negative time lag. This time lag can be exploited for energy efficiency improvement. For example clock and power supply could be completely switched off when decoding is finished. This reduces energy and leakage current. Another possibility is to slow down the frequency (frequency scaling). This reduces the energy by the same amount as in the previous case but the peak power consumption during decoding instead of leakage is minimized.



R : Reference convolutional decoder with R = 0.75 and fixed throughput of 1 Gbit/s
1 : LDPC code performing 1 iteration
  a) identical throughput (1 Gbit/ s) ~ 4dB worse com. performance
  b) 5 times higher throughput (5 Gbit/ s) ~ 4dB worse com. performance
2 : LDPC code performing 2 iteration
  a) identical throughput (1 Gbit/ s) ~ 1dB better com. performance
  b) 2.5 times higher throughput (2.5 Gbit/ s) ~ 1dB better com. performance
3 : LDPC code performing 5 iteration identical throughput (1 Gbit / s) ~ 4dB better com. performance

Fig. 9.3 Implementation efficiency and communications performance for WiMedia 1.5 standard (16-QAM, CM1 channel according to IEEE 802.15.3a [25])

The most efficient technique is voltage scaling in which the voltage is reduced which results in the highest energy efficiency.

- The system throughput is changed (scenario *b* in Fig. 9.3). In this scenario the area efficiency increases by the same amount as the throughput increases due to smaller number of iterations (points $3 \rightarrow 2b \rightarrow 1b$).

We see that in terms of efficiency the LDPC decoder is increasing with decreasing communication requirements, i.e. number of iterations. Thus, the *decoder efficiency is represented by various efficiency evaluations instead of a single point* in the design space. This varying set of points results from varying communication performance requirements. We also see that the efficiency of the LDPC decoder outperforms the convolutional decoder at the same communication performance.

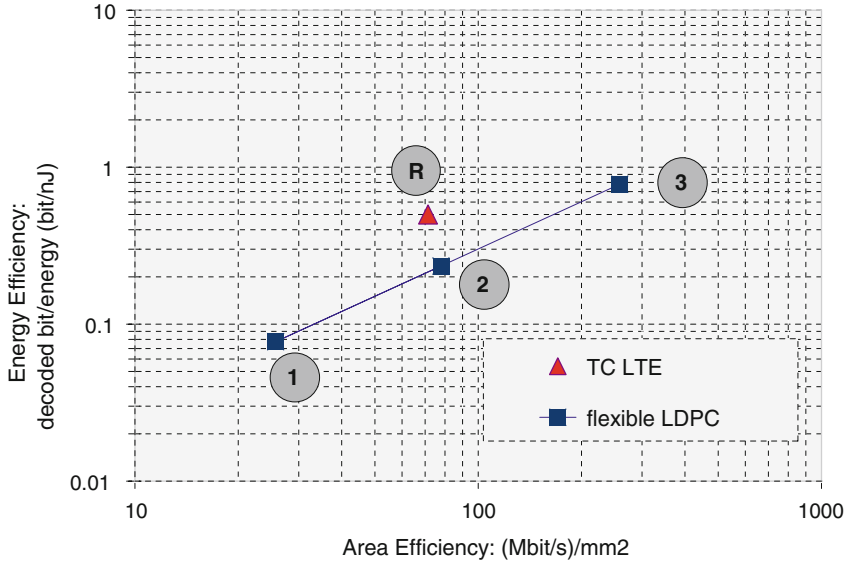
9.3.2 Communications Performance Driven Exploration

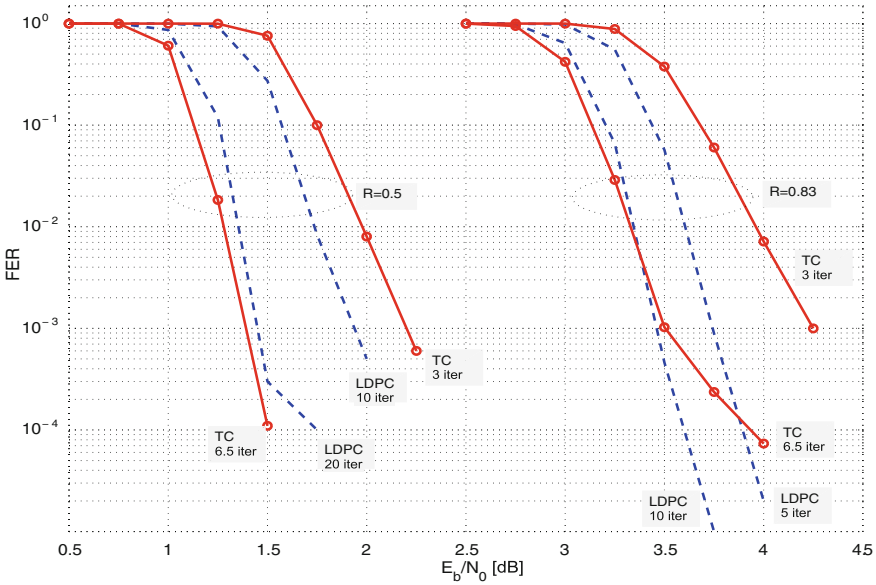
In the second design study we compare two iterative decoding techniques and put emphasis on the resulting communications performance. An LTE turbo decoder is compared with a flexible LDPC decoder which supports code rate flexibility. The right graph in Fig. 9.4 shows the communication performance for the two decoding schemes for different code rates ($R = 0.5$ and $R = 0.83$) and iteration numbers. The number of information bits is $K = 6140$ in all cases. Frame error rates are based on fixed point simulations matching the hardware implementation. We use the communications performance of the turbo decoder with 6.5 iterations as reference point for both code rates. The 6.5 iterations result from the throughput constraint of 150 Mbit/s which is specified in the LTE standard. The 6.5 iterations fulfill the LTE communications performance requirements for all code rates.

It is well known that the communication performance in LDPC decoding depends on the number of iterations *and* the code rate. The LDPC decoder under investigations provide large code rate flexibility, i.e., the hardware can support various code rates. The LDPC decoder requires 10 iterations for $R = 0.83$ and 20 iterations for $R = 0.5$ to match the performance of the turbo decoder. For a code rate of $R = 1/3$ even 40 iterations are mandatory (this is not shown in Fig. 9.4b).

The corresponding results in the implementation space are noteworthy. The turbo decoder efficiency is identical for all code rates (see left graph in Fig. 9.4). Thus, we have only one fixed efficiency point. This is due to the fact that the code rate flexibility is implemented by puncturing which has negligible impact on throughput, area and energy.

However, the situations is completely different for the flexible LDPC decoder. For a given communications performance the code rate has strong impact on the number of required iterations. This iteration number influences the implementations efficiency as we have seen in the previous case study. But beside this impact of the iteration number, there is also a direct impact of the code rate on the implementation efficiency since lower code rates require also a more accurate decoding algorithm





R : Reference of LTE turbo decoder
150 Mbit/s throughput for all code rates reference TC performance at 6.5 iterations
1 : LDPC code at code rate R= 1/3
~ max throughput 30Mbit/s ~ 40 iterations to match TC performance
2 : LDPC code at code rate R = 1/2
~ max throughput 90Mbit/s ~ 20 iterations to match TC performance
3 : LDPC code at code rate R = 0.83
~ max throughput 300Mbit/s ~ 10 iterations to match TC performance

Fig. 9.4 Implementation efficiency and communications performance of LTE turbo code/decoder and a flexible LDPC decoder (BPSK, AWGN channel)

(λ-3-Min algorithm instead of the less complex Min-Sum algorithm). The resulting efficiency points are shown in the left graph of Fig. 9.4. We see that the efficiency increases in both directions with increasing code rate (points $1 \rightarrow 2 \rightarrow 3$).

The important observation in this exploration is the varying implementation efficiency of the flexible LDPC decoder. The different efficiency evaluations result from the required code rate flexibility in the LDPC decoder which is necessary to match the communications performance with respect to a competitive turbo code decoder. We see that analyzing only one code rate, and thus one snap shot, could result in a wrong efficiency conclusions.

The two explorations have shown that implementation efficiency for iterative decoders often results in many points instead of a single point in the design space. The different evaluations result from the strong interrelation between communication performance, flexibility and implementation efficiency.

9.4 Comparison of Published Decoder Architectures

Comparing different decoder architectures objectively, even given proper metrics, is a very challenging task. This is due to the fact that published decoders

- are based on different technology nodes (e.g. 180 nm, 130 nm, 65 nm) and characterization assumptions (e.g. worst case, nominal case or best case operating conditions) and
- differ in their communications performance with respect to the chosen architectural parameters.

To highlight the problem of a fair comparison, we list selected state-of-the-art turbo decoder architectures in Table 9.3. This table shows the implementation parameters of the various decoders, i.e. frequency, area, throughput, technology node, but also parameters which have a strong impact on the communications performance. First, we will discuss technology parameters that influence the achievable throughput followed by architectural parameters affecting the communications performance.

9.4.1 Technology Parameters

Different technologies have different area, timing (frequency) and energy characteristics. In principle, it is possible to normalize the area from one technology node (e.g. 130 nm) analytically to a reference technology node (e.g, 65 nm) [30], but scaling timing and energy is very difficult for advanced technology nodes. We focus here on only some aspects of technology scaling.

In larger technologies, e.g. a 130 nm process, the critical path is normally found in the logic. In turbo decoders, for example, it is typically determined by the add-compare-select unit in the maximum a posteriori (MAP) processing kernel.

Table 9.3 Published 3GPP TC decoder architectures with corresponding architectural and technology parameters

Work	Standards	Results from	Throu. (Mbit/s)	@ iter.	f_{cyc} (MHz)	Technology (nm)	Area (mm^2)	Parameters com. performance
[13]	HSPA	Chip	24	6	145	180	14.5	$Q = 5, WL = 40, AL = 40$
[14]	HSPA	Chip	20.2	5.5	246	130	1.2	$Q = 5, WL = 40, AL = 40$
[15]	HSPA	Synthesis	758	6	256	130	13.1	$Q = 5, WL = 32, AL = 32$
[16]	LTE	Chip	186	8	250	130	10.7	$Q =?, WL = AL = 128$
[17]	LTE	Chip	390	5.5	302	130	3.57	$Q = 5, WL = 30, AL = 30$
[18]	LTE	Place & route	1280	6	400	65	8.3	$Q = 6, WL = 64, AL = 64$
[33]	HSPA	Synthesis	60	6	166	180	16.0	$Q = 6, WL = 20, AL = 20$
[34]	HSPA	Synthesis	184	6	300	130	3.8	$Q = 6, WL = 8, AL = 20$
[27]	LTE	Chip	150	6.5	300	65	2.1	$Q = 6, WL = 32, AL = 96$
[35]	LTE	Synthesis	300	6.5	350	40	1.46	$Q = 6, WL = 64, AL = 64$

The communications performance depends mainly on: input quantization (Q), window length (WL), and acquisition length (AL)

In advanced technologies of 65 nm and less, memories are becoming more and more the limiting factor in terms of timing and energy [30]. Today, we can see large variations in timing and energy efficiency of memories even in the same technology node, depending on the technology provider and the final memory instantiation [31]. Consequently, the assumption that memories scale as good as the logic, when scaling published timing and energy to reference node, is not true [30, 32].

The assumptions under which published frequency and energy data were obtained are another important issue. Timing and energy characterization can be performed under nominal, worst and best case operating conditions. These conditions are related to voltage supply, temperature and technology variations. The achievable frequency can vary up to a factor of two between worst case and best case conditions. For example, under worst case conditions the decoder presented in [27] has a maximum frequency of $f_{cyc} = 300\,\text{MHz}$, but achieves $f_{cyc} = 450\,\text{MHz}$ under nominal case conditions. In Table 9.3 we list the operating conditions, when specified in the publication. Measurements of fabricated chips are often performed under nominal or even best case conditions, whereas some of the listed decoders assume worst case conditions. Depending on the assumed operating conditions, we see large differences in achievable frequencies, which in turn results in large variations in the final throughput, see Sect. 6.3.3.

For comparing different architectures there exist one further problem. The dramatic change of efficiency for post synthesis data or post place and route (P&R) data. Often when designing industry chips we assume a secure margin for the final P&R data of up to 30 %, i.e., we add this overhead to the post synthesis area to enable a fast extrapolation of the final chip area. However, the move to higher throughputs and the mandatory high parallelism of the architectures pose stringent constraints for the wiring. Especially for LDPC codes where randomness has to be ensured the overhead for P&R becomes significant and exceeds the expected 30 % overhead. One example which highlight the problems of place and route is shown in Fig. 9.5.

In this graph the already presented LDPC and turbo decoder architectures are shown (throughput of 150 Mbit/s). In addition advanced LDPC and turbo decoder architectures are shown with throughput numbers in the Gbit/s range. The data of these architectures are shown in Table 9.4 and are taken from [36, 37]. Again the same 65 nm technology for comparison reasons is used.

Two important problems for future channel decoders can be seen:

- The synthesis data of the decoders presented in Table 9.4 show a promising efficiency increase in terms of area and power efficiencies. However, after P&R the efficiency drops significantly due to the large overhead of routing. This can be seen for the LDPC decoder architecture (1 Gbit/s) where the efficiency points are present before and after P&R.
- For the turbo decoders we can see that the efficiency points are in the same range for both implementations. The high throughput turbo decoders features new design techniques to increase the architecture efficiency. However, the increase in terms of area and energy efficiency is saturating, due to the large overhead of P&R.

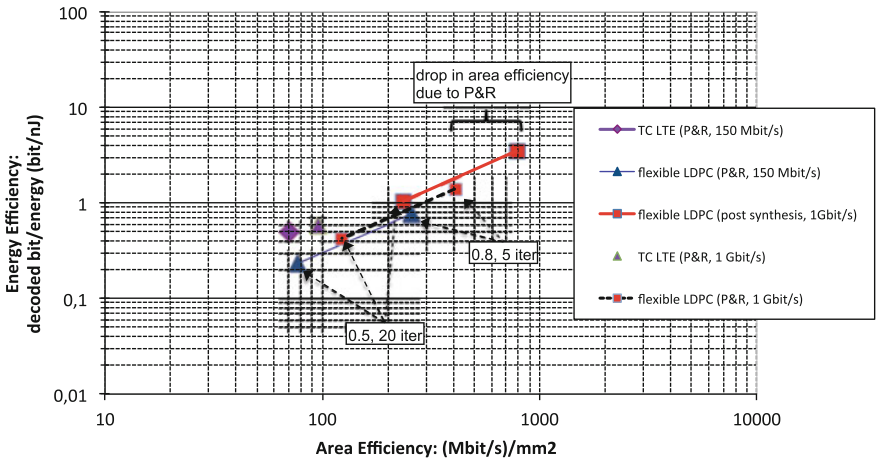


Fig. 9.5 Architecture efficiency trend, all data points are derived with place and route. Only the data for high throughput LDPC decoder shows as well post-synthesis efficiency points

Table 9.4 Gigabit reference decoders: service parameters and implementation results in 65 nm (low power) technology, including area and dynamic power consumption

Channel decoder	Flexibility	Block length	Throughput (Mbit/s)	Frequency (MHz)	Area (mm^2)	Power (mW)
HSPA, LTE	R = 1/3 to	N = 18 k	1200	300	11.2	~2000
TC [36]	R = 9/10		(7 iter)	(P&R)	(P&R)	(P&R)
Flexible [37]	R = 0.5 to	N = 3720 to	1000 (R = 1/2 20 iter)	275	4.6	~1320
LDPC	R = 0.8	N = 14.8 k	4600 (R = 4/5 5 iter)	(P&R)	(P&R)	(P&R)

9.4.2 Communications Performance

The communication performance is mainly affected by three parameters, which also have a strong impact on throughput, area, and energy: input quantization (Q), window length (WL), and acquisition length (AL). Efficient decoder architectures split the input block into several so-called windows, to reduce the memory requirements and to permit parallel processing, see Chap. 6. To counteract the consequent degradation of the communications performance, an acquisition step is introduced at the window borders. Of course, the input quantization has a large and direct effect on the communications performance. The quantization of data internal to the turbo decoder can be derived from the input quantization. Typically, a 5 bit input quantization shows a 0.2 dB performance degradation compared to a 6 bit input quantization, assuming

the same number of iterations and a code rate of $R = 1/3$. Most of the published turbo decoder architectures are instantiated with 5 or 6 bits for input quantization.

To highlight the impact of window and acquisition length on the communications performance, we compare the two turbo decoders [34] and [27]. They share the same quantization, technology, and basic decoder architecture, but strongly differ in the window length and acquisition length ($WL = 8, AL = 20$ compared to $WL = 32, AL = 96$). For a code rate $R = 1/3$ and a given number of iterations, the two decoders show a nearly identical communications performance. However, for $R = 0.95$, the highest code rate specified in the LTE standard, the difference in the communications performance after 6 iterations is about 2 dB. The decoder from [34] achieves the desired communications performance only after 9 iterations, at a much lower throughput, which is first order inverse proportional to the number of iterations, see Sect. 6.3.3.

It is possible to derive some first order analytical equations showing the impact of the parameters (*WL*, *AL*) on throughput and area (Sect. 6.3.3), but there is no analytical equation for the dependency of the communications performance on these parameters. Instead Monte Carlo simulations have to be performed, making it nearly impossible to compare the decoders in Table 9.3 from a communications performance point of view, unless the publications give detailed communications performance graphs for the relevant parameters of the given standards. Although sometimes such graphs are published, they typically cover only a subset of a standard's parameters and do not cover their corner cases. For example, the turbo decoder from [17] is LTE compliant with respect to the supported interleavers and code rates, but it is not clear how many iterations are required to fulfill the LTE communications performance requirements at the code rate of $R = 0.95$.

These examples show that comparing only architectural results of different published decoders may lead to entirely wrong conclusions. The communications performance results have to be published as well, not for one point but for various service parameters, to allow a comparison of implementation efficiencies.

9.4.3 VLSI Efficiency and Communications Performance at Once

The presented approach to plot two different charts for the communications performance and the VLSI performance is not the only possibility to compare architectures. One efficient way to compare different channel coding schemes is to visualize all data in one graph. This is shown in Fig. 9.6. The x-axis shows the signal-to-noise ratio which is mandatory to obtain a desired frame error rate. Here in this chart a fixed frame error rate of $FER = 10^{-3}$ is assumed. The y-axis is split in two parts, the upper part shows the known area efficiency, while the lower part shows the energy efficiency respectively. Thus, one channel coding scheme will have always two lines, one in the top half, the other in the bottom half. In this chart the results for a turbo decoder and an LDPC decoder architecture is shown, according to the architectures of Table 9.1. The resulting efficiencies for two code rates are presented

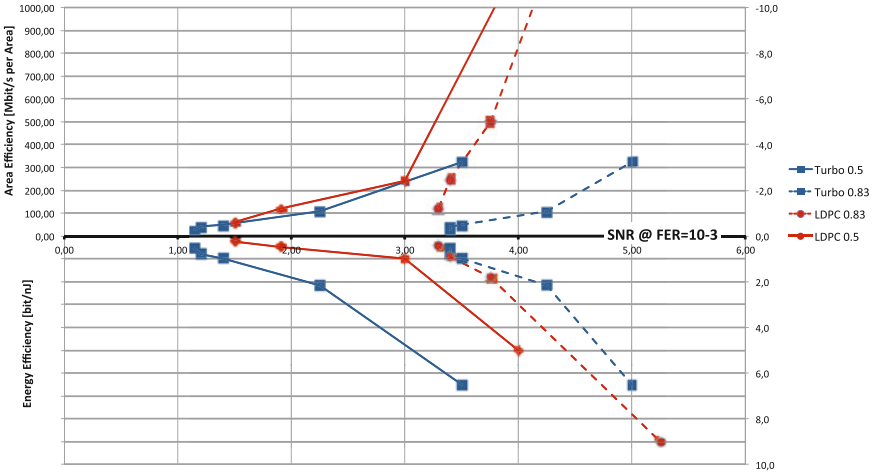


Fig. 9.6 Communications efficiency, area and energy efficiency in one chart, for a fixed FER $= 10^{-3}$

($R = 0.5$ and $R = 0.8$). Depending on the number of iterations we will have different efficiency points with respect to SNR, in terms of area, and in terms of energy efficiency, respectively. The advantage of this chart representation is the possibility to compare communications performance and the resulting VLSI efficiency at once. Here, it can be seen that the turbo decoder architecture has advantages for the $R = 0.5$ scenario, while the architecture of the LDPC decoder shows a better efficiency for the case of a high code rate $R = 0.8$.

In summary of the chapter we analyzed the trade-offs between implementation efficiency, communications performance and flexibility. Meaningful efficiency metrics are mandatory to explore and evaluate the resulting design space. We introduced and discussed suitable energy and area efficiency metrics which are based on decoded information bit per energy and throughput per area unit. Various channel decoder implementations were utilized to examine these efficiency metrics with respect to the achieved communications performance and with respect to the decoder flexibility. The presented approaches allow to systematically compare different realizations by jointly considering: implementation efficiency, communications performance and flexibility.

References

1. van Berkel, C.H.: Multi-core for mobile phones. In: Proceedings of DATE '09. Design, Automation. Test in Europe Conference. Exhibition, pp. 1260–1265 (2009)
2. Kienle, F., Wehn, N., Meyr, H.: On complexity, energy- and implementation-efficiency of channel decoders. IEEE Trans. Commun. **59**(12), 3301–3310 (2011). doi:10.1109/TCOMM.2011.092011.100157

3. Meyr, H., Moeneclaey, M., Fechtel, S.A.: Digital Communication Receivers. Wiley, New York (1998)
4. Eberli, S.C.: Application-Specific Processor for MIMO-OFDM Sofware-Defined Radio. Ph.D. Thesis, ETH Zurich, Integrated Systems Laboratory (2009)
5. Galli, S.: On the fair comparison of FEC schemes. In: Proceedings of IEEE international communications (ICC) conference, pp. 1–6 (2010). doi:10.1109/ICC.2010.5501811
6. Miranda, M., Ghez, C., Brockmeyer, E., Op De Beeck, P., Catthoor, F.: Data transfer and storage exploration for real-time implementation of a digital audio broadcast receiver on a Trimedia processor. In: Proceedings of 15th Symposium on Integrated Circuits and Systems Design, pp. 373–378 (2002). doi:10.1109/SBCCI.2002.1137685
7. Rowen, C.: Energy-efficient LTE baseband with extensible dataplane processor units. In: 9th International Forum on Embedded MPSoC and Multicore, Savanna, USA (2009)
8. Boutillon, E., Castura, J., Kschischang, F.: Decoder-first code design. In: Proceedings of 2nd International Symposium on Turbo Codes & Related Topics, pp. 459–462, Brest, France (2000)
9. Mansour, M., Shanbhag, N.: Architecture-aware low-density parity-check codes. In: Proceedings of 2003 IEEE International Symposium on Circuits and Systems (ISCAS '03), Bangkok, Thailand (2003)
10. Kwak, J., Lee, K.: Design of dividable interleaver for parallel decoding in turbo codes. Electron. Lett. **38**(22), 1362–1364 (2002)
11. Third Generation Partnership Project: 3GPP LTE (Long Term Evolution) Homepage. http://www.3gpp.org/Highlights/LTE/LTE.htm
12. European Telecommunications Standards Institute (ETSI): Digital Video Broadcasting (DVB) Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; TM 2860r1 DVBS2-74r8. www.dvb.org
13. Bickerstaff, M., Davis, L., Thomas, C., Garrett, D., Nicol, C.: A 24 Mb/s Radix-4 LogMAP turbo decoder for 3GPP-HSDPA mobile wireless. In: Proceedings of 2003 IEEE International Solid-State Circuits Conference (ISSCC '03), pp. 150–151, 484, San Francisco, CA, USA (2003)
14. Benkeser, C., Burg, A., Cupaiuolo, T., Huang, Q.: Design and optimization of an HSDPA turbo decoder ASIC. IEEE J. Silid-State Circ. **44**(1), 98–106 (2009). doi:10.1109/JSSC.2008.2007166
15. Prescher, G., Gemmeke, T., Noll, T.: A parameterizable low-power high-throughput turbo-decoder. In: Proceedings of 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005), pp. V-25–V-28, Philadelphia, Pennsylvania, USA (2005)
16. Kim, J.H., Park, I.C.: A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE. In: Proceedings of IEEE Custom Integrated Circuits Conference CICC '09, pp. 487–490 (2009). doi:10.1109/CICC.2009.5280790
17. Studer, C., Benkeser, C., Belfanti, S., Huang, Q.: A 390 Mb/s 3.57 mm^2 3GPP-LTE turbo decoder ASIC in 0.13μm CMOS. In: Proceedings of IEEE International Solid-State Circuits Conference—Digest of Technical Papers, 2010. ISSCC 2010, vol. 53, pp. 274–275, San Francisco, USA (2010)
18. Sun, Y., Cavallaro, J.: Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder. Integr. VLSI J. (2010). doi:10.1016/j.vlsi.2010.07.001
19. Urard, P., Paumier, L., Heinrich, V., Raina, N., Chawla, N.: A 360 mW 105 Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite-transmission portable devices. In: Proceedings of Digest of Technical Papers. IEEE International Solid-State Circuits Conference ISSCC 2008, pp. 310–311 (2008). doi:10.1109/ISSCC.2008.4523181
20. Shih, X.Y., Zhan, C.Z., Lin, C.H., Wu, A.Y.: An 8.29 mm 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 & CMOS process. IEEE J. Solid-State Circ. **43**(3), 672–683 (2008). doi:10.1109/JSSC.2008.916606
21. Mller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., Wehn, N.: A novel LDPC decoder for DVB-S2 IP. In: Proceedings of DATE '09. Design, Automation, Test in Europe Conference. Exhibition, pp. 1308–1313 (2009)

22. Mansour, M., Shanbhag, N.: Low-power VLSI decoder architectures for LDPC codes. In: Proceedings of 2002 International Symposium on Low Power Electronics and Design (ISLPED '02), Monterey, California, USA (2002)
23. Dielissen, J., Hekstra, A., Berg, V.: Low cost LDPC decoder for DVB-S2. In: Proceedings of 2006 Design, Automation and Test in Europe (DATE '06), Munich, Germany (2006)
24. Vogt, T., Wehn, N.: A reconfigurable ASIP for convolutional and turbo decoding in a SDR environment. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **16**, 1309–1320 (2008)
25. Alles, M., Berens, F., Wehn, N.: A synthesizable IP core for WiMedia 1.5 UWB LDPC code decoding. In: Proceedings of IEEE International Conference on Ultra-Wideband ICUWB 2009, pp. 597–601, Vancouver, Canada (2009)
26. IEEE 802.11: Wireless Fidelity (Wireless LAN). http://grouper.ieee.org/groups/802/11/
27. May, M., Ilnseher, T., Wehn, N., Raab, W.: A 150 Mbit/s 3GPP LTE turbo code decoder. In: Proceedings of Design, Automation and Test in Europe, 2010 (DATE '10), pp. 1420–1425 (2010)
28. Guilloud, F., Boutillon, E., Danger, J.: λ-Min decoding algorithm of regular and irregular LDPC codes. In: Proceedings of 3rd International Symposium on Turbo Codes & Related Topics, pp. 451–454, Brest, France (2003)
29. WiMedia Alliance. www.wimedia.org
30. International Technology Roadmap for Semiconductors 2011: ITRS home page. http://public.itrs.net
31. CMP: Circuits Multi-Projets. http://cmp.imag.fr/products/ic/
32. Rabaey, J.M.: Low Power Design Essentials, 1st edn. Springer, Berlin (2009)
33. Thul, M.J., Gilbert, F., Vogt, T., Kreiselmaier, G., Wehn, N.: A scalable system architecture for high-throughput turbo-decoders. J. VLSI Signal Process. Syst. (Special Issue on Signal Processing for Broadband Communications) **39**(1/2), 63–77 (2005). Springer Science and Business Media, Netherlands
34. May, M., Neeb, C., Wehn, N.: Evaluation of high throughput turbo-decoder architectures. In: Proceedings of IEEE International Symposium on Circuits and Systems ISCAS 2007, pp. 2770–2773, New Orleans, USA (2007). doi:10.1109/ISCAS.2007.378627
35. Ilnseher, T., May, M., Wehn, N.: A multi-mode 3GPP-LTE/HSDPA turbo decoder. In: Proceedings of IEEE International Conference on Communication Systems (ICCS 2010), pp. 336–340 (2010)
36. Ilnseher, T., Kienle, F., Weis, C., Wehn, N.: A 2.12 Gbit/s turbo code decoder for LTE advanced base station applications. In: Proceedings of 7th International Symposium on Turbo Codes and Iterative Information Processing (2012)
37. Gimmler, C., Kienle, F., Weis, C., Wehn, N., Alles, M.: ASIC design of a Gbit/s LDPC decoder for iterative MIMO systems. In: Proceedings of International Conference on Computing, Networking and Communications (ICNC), pp. 192–197 (2012). doi:10.1109/ICCNC.2012.6167409

Appendix A
Channel Coding Data

In this appendix various data for communication standards are presented and serves as a parameter look up basis. The related information and parameter descriptions are presented in the corresponding chapters in this manuscript. Data are partially assembled by Creonic GmbH (http://www.creonic.com) (Tables A.1, A.2, A.3, A.4, A.5 and A.6).

Table A.1 Selection of standards and their channel codes, table adapted from [1]

Standard	Codes	States	Code rates	Infobits for CC,TC Codword for LDPC
GSM	CC	16, 64	1/4, 1/3, 1/2	Up to 876
EDGE	CC	64	1/4, 1/3, 1/2	Up to 870
UMTS	CC	256	1/4, 1/3, 1/2	Up to 504
	bTC	8	1/3	Up to 5114
CDMA2000	CC	256	1/6, 1/4, 1/3, 1/2	Up to 744
	bTC	8	1/5, 1/4, 1/3, 1/2	Up to 20730
HSDPA	bTC	8	1/2, 2/3, 3/4	Up to 5114
LTE	bTC	8	1/3–9/10	Up to 6144
DAB	CC	64	1/4	None
DVB-H	CC	64	1/2, 2/3, 3/4, 5/6, 7/8	1624
DVB-T	CC	64	1/2, 2/3, 3/4, 5/6, 7/8	1624
DVB-RCT	dbTC	8	1/2, 3/4	Up to 648
DVB-RCS	dbTC	8	1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7	Up to 1728
IEEE 802.11a/g	CC	64	1/2, 2/3, 3/4	Up to 4095
IEEE 802.11n	CC	64	1/2, 2/3, 3/4	Up to 4095
	LDPC	–	1/2, 2/3, 3/4, 5/6	Up to 1620
IEEE802.16e	CC	64	1/2, 2/3, 3/4, 5/6	Up to 864
(WiMax)	dbTC	8	1/2, 2/3, 3/4	Up to 4800
	LDPC	–	1/2, 2/3, 3/4, 5/6	Up to 1920
DVB-S2/C2/T2	LDPC		1/4–9/10	16200, 64800
WiMedia 1.5 UWB	LDPC		1/2, 5/8, 3/4, 4/5	1200, 1320

(continued)

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3, © Springer Science+Business Media New York 2014

Table A.1 (continued)

Standard	Codes	States	Code rates	Infobits for CC,TC Codword for LDPC
IEEE 802.3an	LDPC		0.841	2048
IEEE 802.15.3c	LDPC		1/2, 5/8, 3/4, 7/8, 14/15	672, 1440
IEEE 802.22	LDPC		1/2, 2/3, 3/4, 5/6	384, 480
CCSDS	LDPC		1/2–4/5	1280–32768
GMR-1	LDPC		1/2–9/10	950–11136

Table A.2 Duo-binary turbo decoding standards, table adapted from [2]

Standard	DVB-RCS [3]	DVB-RCT [4]	WiMax [5]
Codeword sizes	112–5184	192–864	96–9600
Info couples	48–864	72–324	24–2400
Code rates	$\frac{1}{3}-\frac{6}{7}$	$\frac{1}{2}, \frac{3}{4}$	$\frac{1}{2}-\frac{5}{6}$
Trellis states		8	
Input polynomials		$I_0 = 4_8, I_1 = 7_8$	
Feedbackward polynomial		$G_{FB} = 15_8$	
Feedforward polynomials	$G_0 = 13_8$, $G_1 = 11_8$	$G_0 = 13_8$	$G_0 = 13_8$
Number of codes	84	10	24

Table A.3 LDPC code standards with high throughput requirements, adapted from [2]

Standard	802.3 an [6] (10 GBASE-T)	WiMedia 1.5 [7] (UWB)	802.15.3c [8] (60 GHz)
Codeword sizes	2048	1200, 1320	672, 1440
Code rates	$\frac{1723}{2048}$	$\frac{1}{2}-\frac{4}{5}$	$\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \frac{14}{15}$
Sub-matrix sizes	N/A	30	21, 96
Min–Max CN degree	32	5–15	6–32
Min–Max VN degree	6	2–5	1–4
No. of codes	1	8	4
Throughput (Gbit/s)	10	1	$\leq$5

Table A.4 Degree distributions of the WiFi 802.11n LDPC Codes, only three block are defined with N = 648, N = 1248, and N = 1944 bit, table adapted from [9]

	WiFi 802.11n	
R	Variable node degree f	Check node degree g
$\frac{1}{2}$	$[2, 3, 12] = \left\{\frac{11}{24}, \frac{5}{12}, \frac{1}{8}\right\}$	$[7, 8] = \left\{\frac{2}{3}, \frac{1}{3}\right\}$
$\frac{2}{3}$	$[2, 3, 4, 6, 8] = \left\{\frac{7}{24}, \frac{1}{3}, \frac{5}{24}, \frac{1}{24}, \frac{1}{8}\right\}$	$[11] = \{1\}$
$\frac{3}{4}$	$[2, 3, 4, 6] = \left\{\frac{5}{24}, \frac{1}{3}, \frac{1}{4}, \frac{5}{24}\right\}$	$[14, 15] = \left\{\frac{1}{3}, \frac{2}{3}\right\}$

(continued)

Table A.4 (continued)

	WiFi 802.11n	
R	Variable node degree f	Check node degree g
$\frac{5}{6}$	$[2, 3, 4] = \left\{\frac{1}{8}, \frac{1}{12}, \frac{19}{24}\right\}$	$[22] = \{1\}$
$\frac{1}{2}$	$[2, 3, 4, 11] = \left\{\frac{11}{24}, \frac{3}{8}, \frac{1}{24}, \frac{1}{8}\right\}$	$[7, 8] = \left\{\frac{5}{6}, \frac{1}{6}\right\}$
$\frac{2}{3}$	$[2, 3, 7, 8] = \left\{\frac{7}{24}, \frac{1}{2}, \frac{1}{12}, \frac{1}{8}\right\}$	$[11] = \{1\}$
$\frac{3}{4}$	$[2, 3, 6] = \left\{\frac{5}{24}, \frac{1}{2}, \frac{7}{24}\right\}$	$[14, 15] = \left\{\frac{1}{3}, \frac{2}{3}\right\}$
$\frac{5}{6}$	$[2, 3, 4] = \left\{\frac{1}{8}, \frac{5}{24}, \frac{2}{3}\right\}$	$[21, 22] = \left\{\frac{3}{4}, \frac{1}{4}\right\}$
$\frac{1}{2}$	$[2, 3, 4, 11] = \left\{\frac{11}{24}, \frac{3}{8}, \frac{1}{24}, \frac{1}{8}\right\}$	$[7, 8] = \left\{\frac{5}{6}, \frac{1}{6}\right\}$
$\frac{2}{3}$	$[2, 3, 6, 8] = \left\{\frac{7}{24}, \frac{1}{2}, \frac{1}{24}, \frac{1}{6}\right\}$	$[11] = \{1\}$
$\frac{3}{4}$	$[2, 3, 6] = \left\{\frac{5}{24}, \frac{13}{24}, \frac{1}{4}\right\}$	$[14, 15] = \left\{\frac{5}{6}, \frac{1}{6}\right\}$
$\frac{5}{6}$	$[2, 3, 4] = \left\{\frac{1}{8}, \frac{11}{24}, \frac{5}{12}\right\}$	$[19, 20] = \left\{\frac{1}{4}, \frac{3}{4}\right\}$

Table A.5 Degree Distributions of all WiMax 802.16e LDPC Code Classes, the codeword size ranges from N = 576 to N = 2304 bit in steps of 24 bits, table adapted from [9]

	WiMax 802.16e	
R	Variable node degree f	Check node degree g
$\frac{1}{2}$	$f_{[2,3,6]} = \left\{\frac{11}{24}, \frac{1}{3}, \frac{5}{24}\right\}$	$g_{[6,7]} = \left\{\frac{2}{3}, \frac{1}{3}\right\}$
$\frac{2}{3}$ A	$f_{[2,3,6]} = \left\{\frac{7}{24}, \frac{1}{2}, \frac{5}{24}\right\}$	$g_{[10]} = \{1\}$
$\frac{2}{3}$ B	$f_{[2,3,4]} = \left\{\frac{7}{24}, \frac{1}{24}, \frac{2}{3}\right\}$	$g_{[10,11]} = \left\{\frac{7}{8}, \frac{1}{8}\right\}$
$\frac{3}{4}$ A	$f_{[2,3,4]} = \left\{\frac{5}{24}, \frac{1}{24}, \frac{3}{4}\right\}$	$g_{[14,15]} = \left\{\frac{5}{6}, \frac{1}{6}\right\}$
$\frac{3}{4}$ B	$f_{[2,3,6]} = \left\{\frac{5}{24}, \frac{1}{2}, \frac{7}{24}\right\}$	$g_{[14,15]} = \left\{\frac{1}{3}, \frac{2}{3}\right\}$
$\frac{5}{6}$	$f_{[2,3,4]} = \left\{\frac{3}{24}, \frac{5}{12}, \frac{11}{24}\right\}$	$g_{[20]} = \{1\}$

Table A.6 Overview of LDPC codes in DVB standards, table adapted from [2]

Block Size	Code Rate ID	Code Rate	Variable node Degrees	Check node Degrees	Edges	Used in DVB-		
						S2	T2	C2
64800	$\frac{1}{4}$	$\frac{1}{4}$	2, 3, 12	4	194399	×		
64800	$\frac{1}{3}$	$\frac{1}{3}$	2, 3, 12	5	215999	×		
64800	$\frac{2}{5}$	$\frac{2}{5}$	2, 3, 12	6	233279	×		
64800	$\frac{1}{2}$	$\frac{1}{2}$	2, 3, 8	7	226799	×	×	
64800	$\frac{3}{5}$	$\frac{3}{5}$	2, 3, 12	11	285119	×	×	
64800	$\frac{2}{3}$	$\frac{2}{3}$	2, 3, 13	10	215999	×	×	×
64800	$\frac{3}{4}$	$\frac{3}{4}$	2, 3, 12	14	226799	×	×	×
64800	$\frac{4}{5}$	$\frac{4}{5}$	2, 3, 11	18	233279	×	×	×
64800	$\frac{5}{6}$	$\frac{5}{6}$	2, 3, 13	22	237599	×	×	×
64800	$\frac{8}{9}$	$\frac{8}{9}$	2, 3, 4	27	194399	×		

(continued)

Table A.6 (continued)

Block Size	Code Rate ID	Code Rate	Variable node Degrees	Check node Degrees	Edges	Used in DVB-		
						S2	T2	C2
64800	$\frac{9}{10}$	$\frac{9}{10}$	2, 3, 4	30	194399	×		×
16200	$\frac{1}{4}$	$\frac{1}{5}$	2, 3, 12	3, 4	48599	×	×	
16200	$\frac{1}{3}$	$\frac{1}{3}$	2, 3, 12	5	53999	×		
16200	$\frac{2}{5}$	$\frac{2}{5}$	2, 3, 12	6	58319	×		
16200	$\frac{1}{2}$	$\frac{4}{9}$	2, 3, 8	4, 5, 6, 7	48599	×	×	×
16200	$\frac{3}{5}$	$\frac{3}{5}$	2, 3, 12	11	71279	×		
16200	$\frac{3}{5}$	$\frac{3}{5}$	2, 3, 12	9	58319		×	
16200	$\frac{2}{3}$	$\frac{2}{3}$	2, 3, 13	10	53999	×	×	×
16200	$\frac{3}{4}$	$\frac{11}{15}$	2, 3, 12	9, 10, 11, 12, 13	47519	×	×	×
16200	$\frac{4}{5}$	$\frac{7}{9}$	2, 3	11, 12, 13	44999	×	×	×
16200	$\frac{5}{6}$	$\frac{37}{45}$	2, 3, 13	16, 17, 18, 19	49319	×	×	×
16200	$\frac{8}{9}$	$\frac{8}{9}$	2, 3, 4	27	48599	×		×

Appendix B
Communications Performance Results

The communications performance results serves as a reference for comparing simulations. The related information and parameter descriptions are presented in the corresponding chapters in this manuscript. Simulations are partially assembled by Creonic GmbH (http://www.creonic.com). The communications performance data for LTE turbo codes are the floating point reference simulations which were utilized for the chip designs in Chap. 9 (Figs. B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10 and B.11).

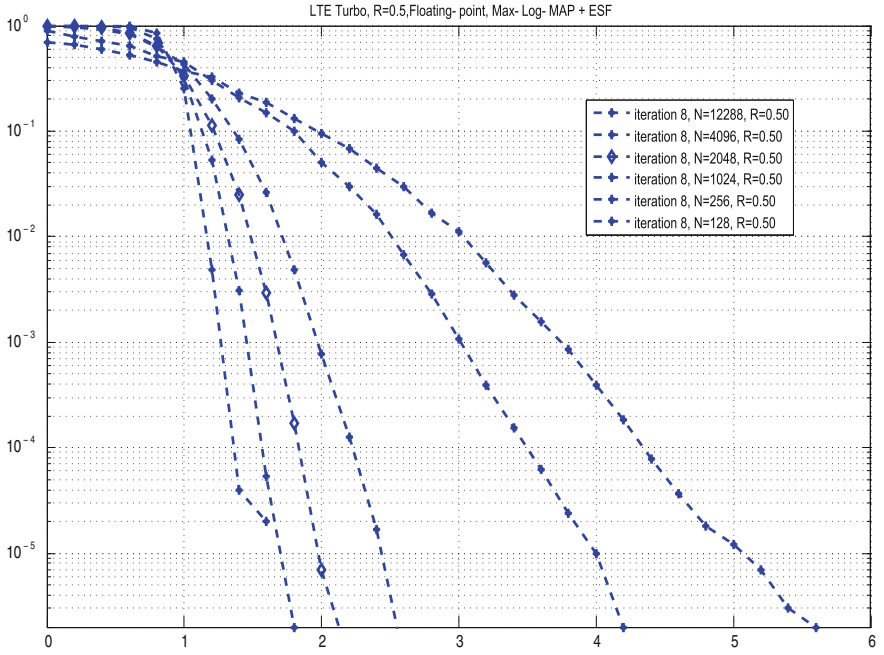


Fig. B.1 Communications performance for LTE turbo decoding for the code rate $R = 1/2$ for different block sizes

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3, © Springer Science+Business Media New York 2014

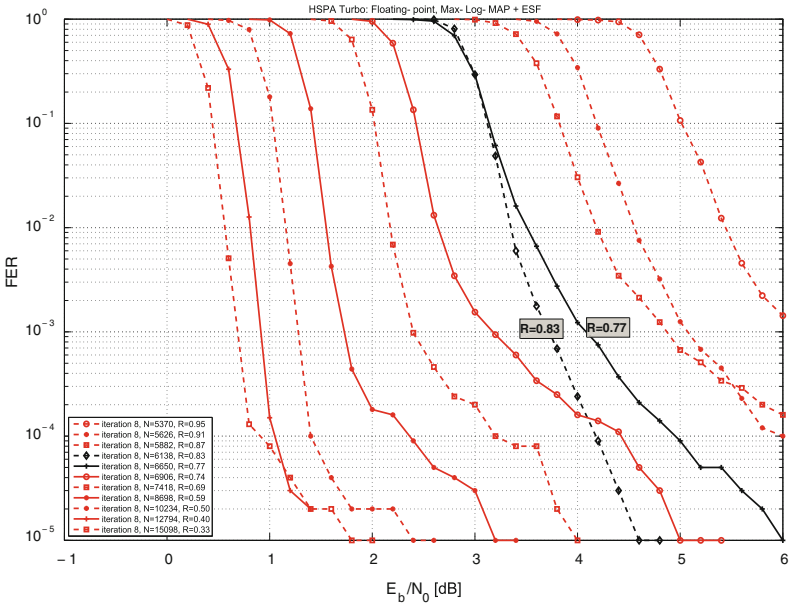


Fig. B.2 Communications performance for HSPA turbo decoding for different code rates, all with maximum infobit size of $K = 5114$

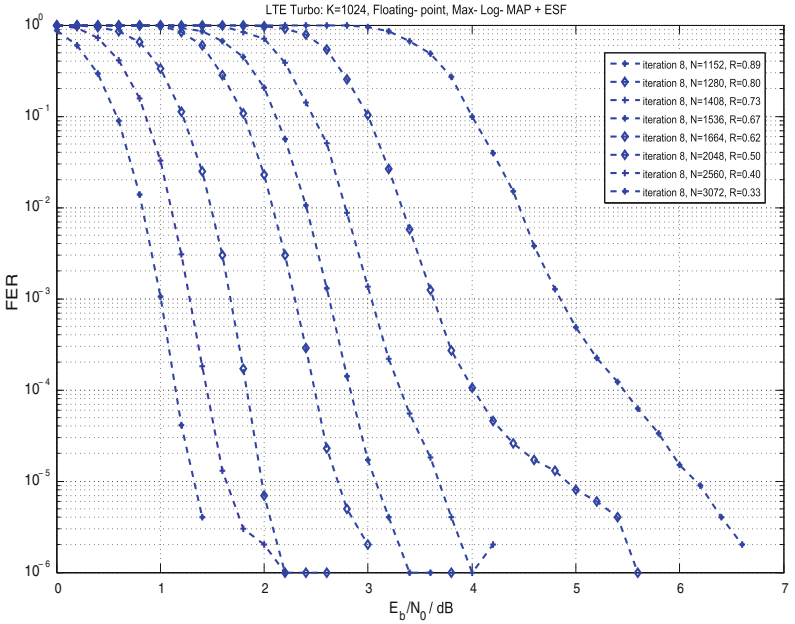


Fig. B.3 Communications performance for LTE turbo decoding for different code rates, all with maximum infobit size of $K = 6144$

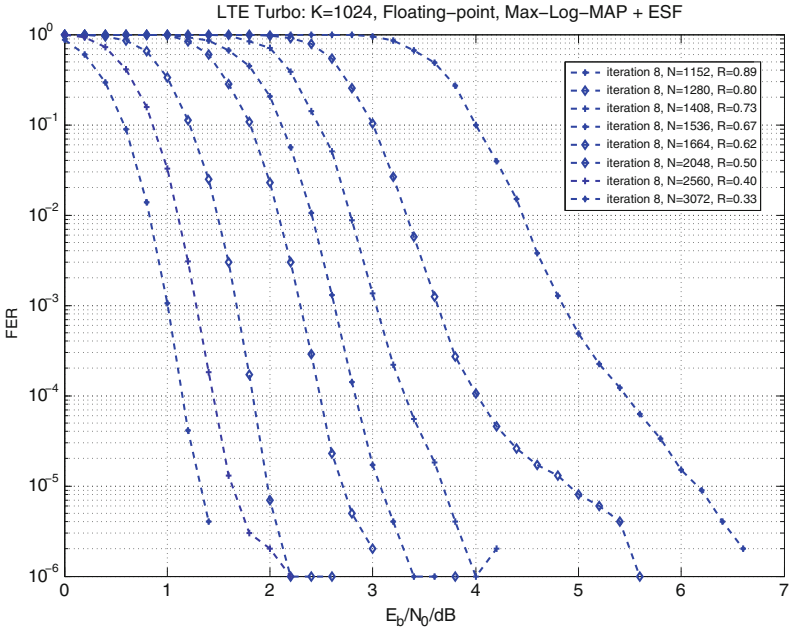


Fig. B.4 Communications performance for LTE turbo decoding for different code rates, all with maximum infobit size of $K = 1024$ bits

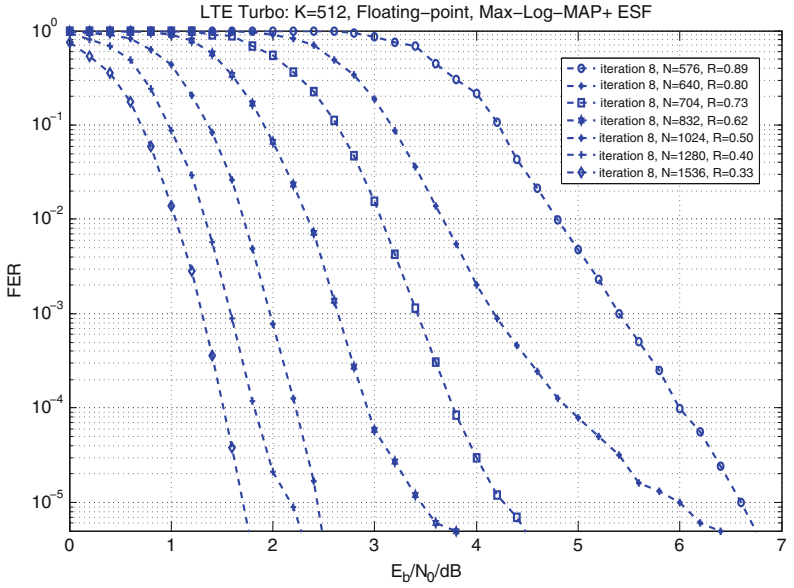


Fig. B.5 Communications performance for LTE turbo decoding for different code rates, all with maximum infobit size of $K = 512$ bits

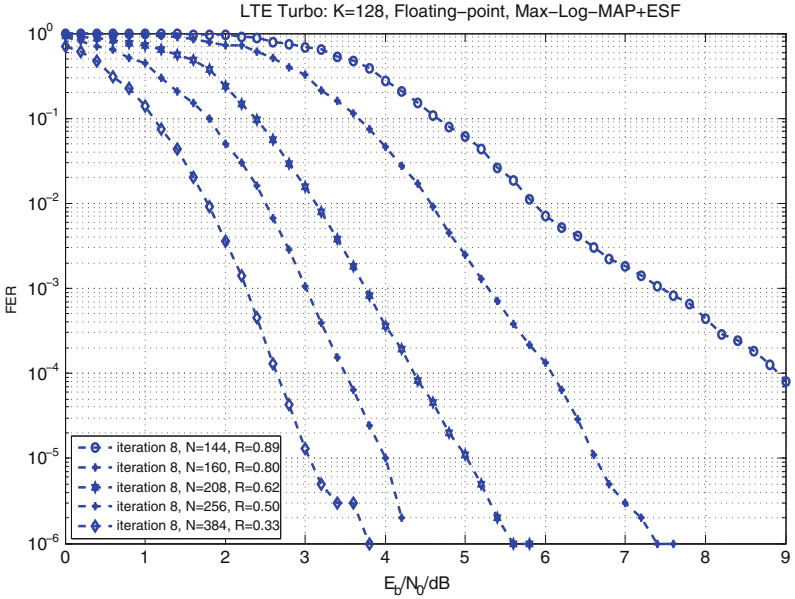


Fig. B.6 Communications performance for LTE turbo decoding for different code rates, all with maximum infobit size of $K = 128$ bits

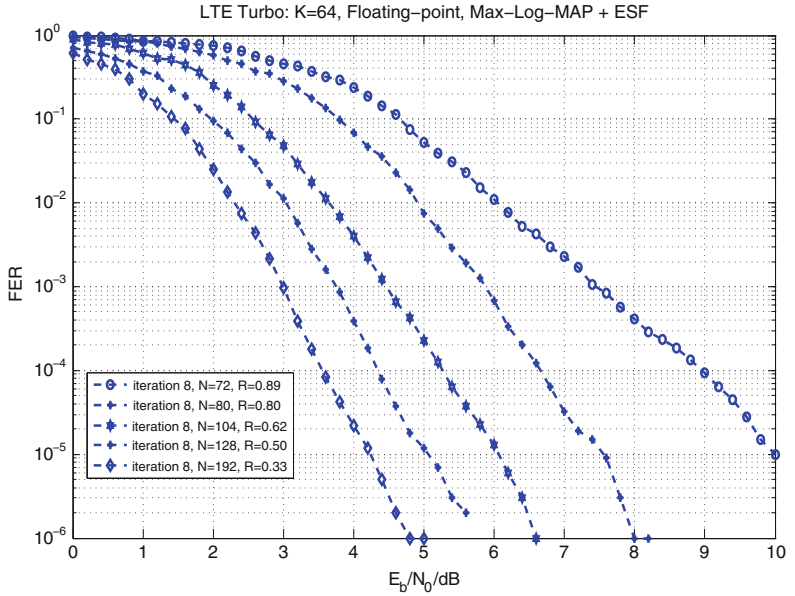


Fig. B.7 Communications performance for LTE turbo decoding for different code rates, all with maximum infobit size of $K = 64$ bits

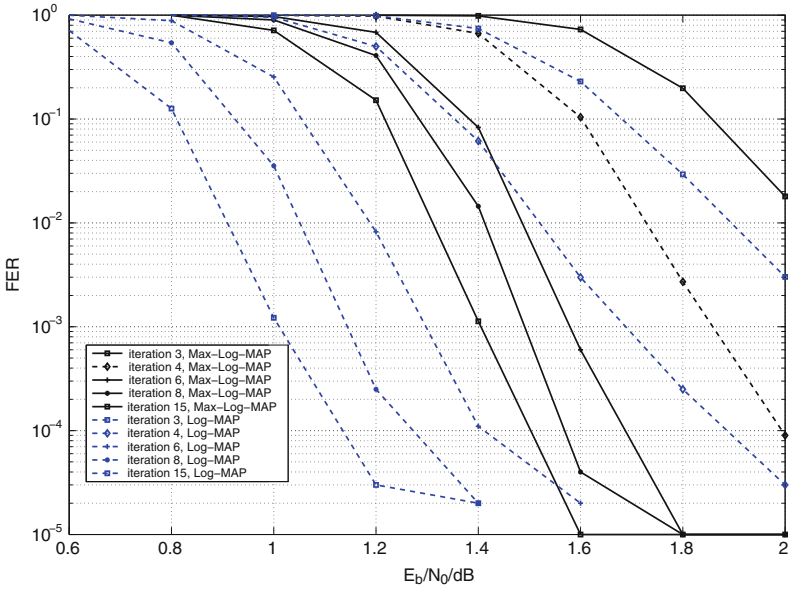


Fig. B.8 Communications performance for LTE turbo decoding of $R = 0.5$ and $K = 6144$ bits for two decoding algorithms. Shown are the performance for different iterations utilizing the Log-MAP and the Max-Log-MAP algorithm

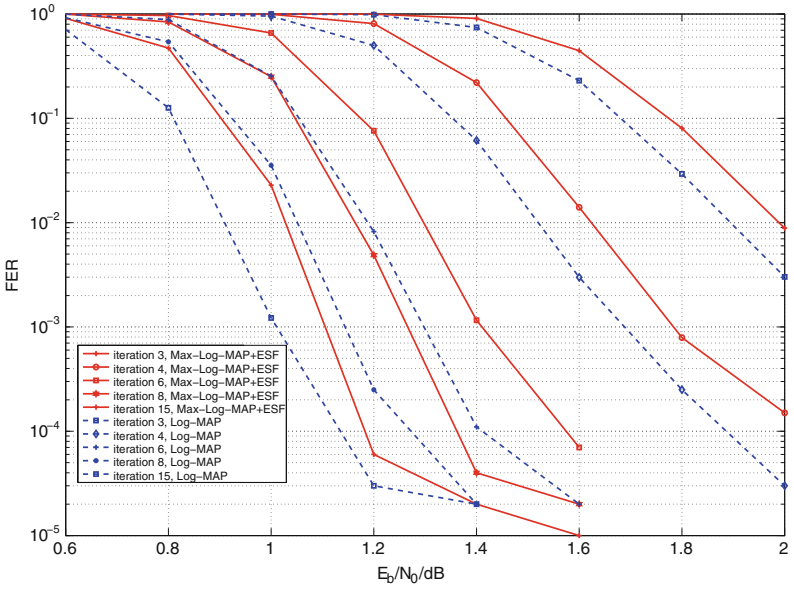


Fig. B.9 Communications performance for LTE turbo decoding of $R = 0.5$ and $K = 6144$ bits for two decoding algorithms. Shown are the performance for different iterations utilizing the Log-MAP and the Max-Log-MAP with extrinsic scaling factor of $ESF = 0.75$

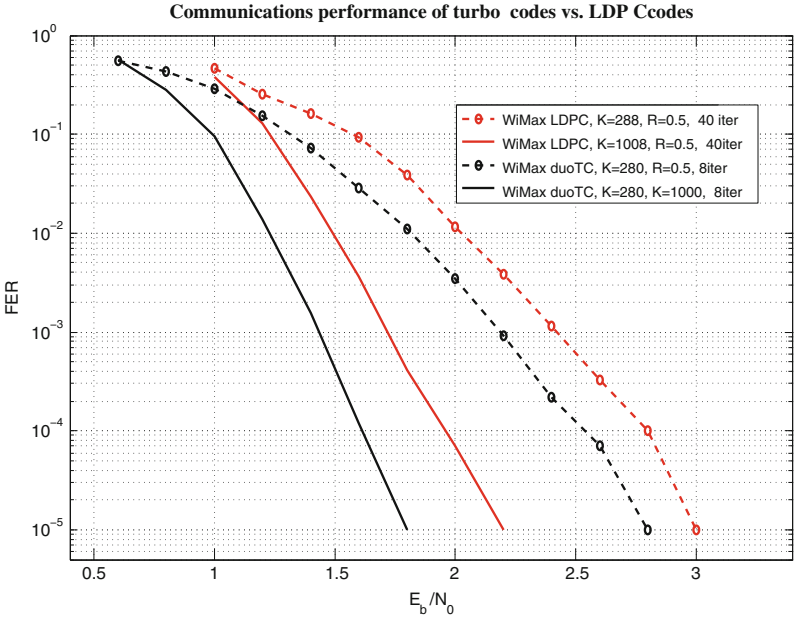


Fig. B.10 Communications performance for duo-binary turbo codes and LDPC codes both specified in the WiMAX standard. The code rate is in all cases $R = 0.5$

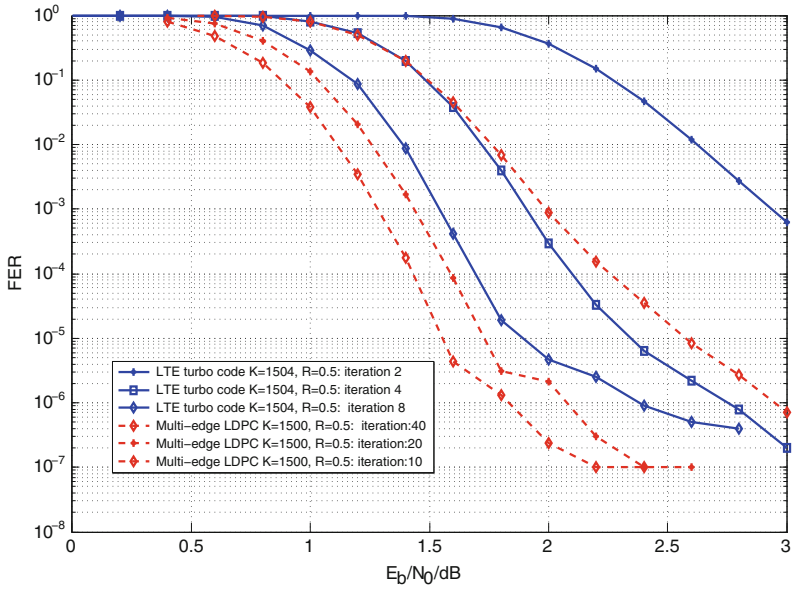


Fig. B.11 Communications performance of LTE turbo codes and multi-edge type LDPC codes, both with $K = 1500$ infobits and $R = 0.5$

Appendix C
Probability Terms

In this section we describe several probability terms by means of examples. For formal definitions of probability terms see for example [10].

One popular example is to calculate the odds in a game of chance, like rolling a die. Rolling dice is a discrete experiment with 6 possible outcomes, which is denoted as sample space of the experiment $\Omega = \{1, 2, 3, 4, 5, 6\}$. The probability of an idealized die showing a particular number after rolling it, can be calculated by

$$P(X = 6) = \frac{\text{Possible outcomes favoring event 6}}{\text{Total number of possible outcomes}}, \tag{C.1}$$

here with rolling a 6 as an example.

When referring to an outcome of an experiment we utilize a capital letter, where X denotes a discrete random variable. A random variable is an expression whose value is the outcome of one particular experiment.

We have to distinguish between the small letter x which reflects always one particular event with one result, i.e. $x = 6$. However, a random variable, with capital X, can have a more complicated probability constraint. For example, we can write the probability showing a number smaller than 4 after rolling it is

$$P(X < 4) = P(x = 3) + P(x = 2) + P(x = 1) = 1/2 \tag{C.2}$$

We can define as well more complected event like the probability to be an odd number $Q \in \{1, 3, 5\}$, the probability of this experiment is

$$P(X = Q) = P(x = 1) + P(x = 3) + P(x = 5) = 1/2 \tag{C.3}$$

The probability of an event is always linked to all possible events, which means here:

$$p_X(x = 1) + p_X(x = 2) + p_X(x = 3) + p_X(x = 4) + p_X(x = 5) + p_X(x = 6) = 1 \tag{C.4}$$

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3, © Springer Science+Business Media New York 2014

$p_X(x)$ is called probability distribution function of a random variable X. The distribution has to satisfy

$$\sum_{x \in \Omega} p_X(x) = 1. \tag{C.5}$$

For a discrete sample space Ω, as shown here, the function $p_X(x)$ is denoted as probability mass function. Assuming a continuous sample space the sum is replaced by an integral, this case is shown later. The probability to obtain an odd number Q can now be evaluated in a more generic way as:

$$P(X = Q) = \sum_{x \in Q} p_X(x) = 1/2 \tag{C.6}$$

The probability of either random variable X or random variable Y occurring is given by:

$$P(X \cup Y) = P(X) + P(Y) \tag{C.7}$$

This equation holds only when the two random variables are independent. For example, the probability to roll a 6 in the first attempt or to roll a 6 in the second attempt is $P(X \cup Y) = P(x = 6) + P(x = 6) = \frac{1}{3}$.

If two events are independent then their joint probability, the probability of both events occurring in the same experiment, is

$$P(X \cap Y) = P(X, Y) = P(X)P(Y). \tag{C.8}$$

The probability of rolling two sixes in a row is $P(X, Y) = P(x = 6) \cdot P(x = 6) = \frac{1}{36}$.

The probability of an random variable X given an occurrence of Y is called the conditional probability. The conditional probability of X given Y is defined as

$$P(X|Y) = \frac{P(X, Y)}{P(Y)}. \tag{C.9}$$

Using this we can calculate the probability to roll a six in the second attempt, when the first attempt was a six, already. $P(X = 6|Y = 6) = \frac{P(X \cap Y)}{P(Y)} = \frac{\frac{1}{36}}{\frac{1}{6}} = \frac{1}{6}$. Note that, since both events are independent, the conditional probability is just $P(X|Y) = P(X)$!

Equation C.9 links the joint probability of two random variable $P(X, Y)$ and the conditional probability. Thus, the relation between $P(X|Y)$ can be derived, which leads to **Bayes**' theorem:

$$P(X|Y) = \frac{P(X, Y)}{P(Y)} = \frac{P(Y|X) \cdot P(X)}{P(Y)} \tag{C.10}$$

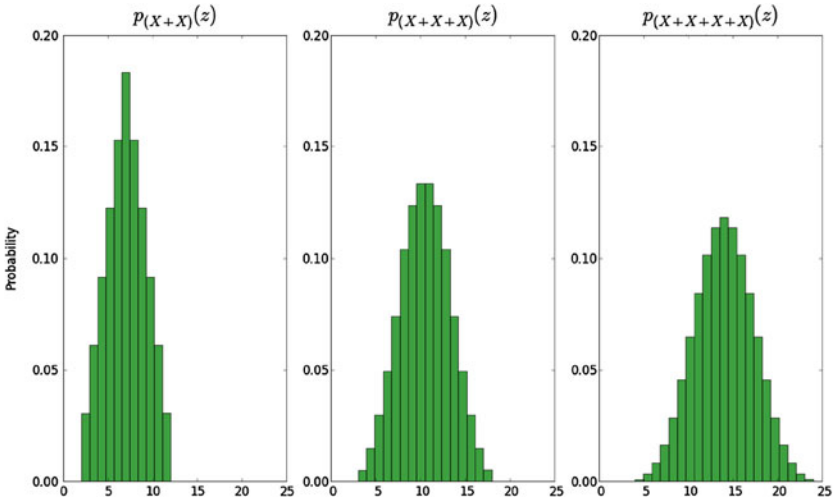


Fig. C.1 Probability distribution of the sum of two, three, and four dice events

The probability mass function in the example of rolling a die describes a so called discrete uniform distribution. The discrete uniform distribution is important, e.g. to model a random binary source, which has an equal probability to generate either a zero or a one event. However, there are many more discrete distributions where different events have a distinct probability of occurrence. Imagine the following experiment: instead of rolling one die, as in the previous examples, now there are two, three, or four dice, which are rolled together. The result of the experiment is the sum of pips all dice show at the end of the experiment. The possible outcomes of this experiment range from 2 to 12 for two dice (or n to 6n for n dice), but not every outcome is equally likely. The probability for each outcome of this experiment can be seen in Fig. C.1.

Assuming the sum of two dice the outcome z is quite clear with the maximum sum to be $z = 12$ and the minimum of $z = 2$. These are rather rare events, as both dice have to show the same value. It can be seen that the $z = 7$ is the most common sum, because various different combinations result in a sum of 7 pips. The resulting probability mass function is denoted with $p_{(X+X)}(z)$ and can be obtained by the convolution $conv(p_X(x), p_X(x))$ of the individual mass functions. The larger the sum of individual experiments, e.g. the sum of 4 experiments $p_{(X+X+X+X)}(z)$, the more we approach the important Gaussian distribution.

In communication systems we typically evaluate specific observations, lower case letter x, within a discrete random variable X. Thus, whenever possible, we skip the capital X, and we will use $p(x)$ or $P(x|y)$ instead.

Gaussian Distribution

So far we have only discussed discrete random variables, the distributions of which can be described by a probability mass function. In the case of continuous random variables the distributions are described by so called probability density functions (pdf). One probability density function of particular importance is the Gaussian or normal distribution, which is given by:

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}(\mathrm{x}-\mu)^2\right\} \tag{C.11}$$

The Gaussian variable $\mathcal{N}(x|\mu,\sigma^2)$ is fully described by μ and σ^2, with μ being the mean or center value and σ^2 being the variance, which gives the width of the distribution. With the probability density function we can evaluate the probability of a random variable $X \sim \mathcal{N}(x|\mu,\sigma^2)$ lying within a certain range:

$$P[a \leq X \leq b] = \int_a^b p(x|\mu,\sigma^2)\,\mathrm{d}x \tag{C.12}$$

The standard deviation σ defines the boundaries at which 68.27 % of all outcomes are in the range of $[-\sigma,\sigma]$, thus $P[-\sigma \leq X \leq \sigma] \approx 0.6827$. 95.45 % of all outcomes X will be in the range of $[-2\sigma \leq X \leq 2\sigma]$ and 99.73 % within the range of $\pm 3\sigma$, respectively.

As mentioned, the probability density is defined for continuous random variables. However, in hardware design we have typically a limited, quantized, range of values and thus discrete events. The probability of a random variable falling in a certain range has to be calculated via the corresponding range integral. Figure C.2 shows the continuous Gaussian function, the ranges for different standard deviation values are highlighted. Figure C.3 shows the corresponding probability mass function with ranges of identical probability; the well known histogram visualization appears. It is worth mentioning that, if the range integrals are not perfectly symmetrical with respect to the mean value, a probability mass function results that shows slightly imbalanced bins, as can be observed in Fig. C.3. In hardware design we always have to keep in mind that this can happen due to limited resolution or incomplete information about μ or σ. An algorithm realized in hardware has to be stable with respect to these inaccuracies.

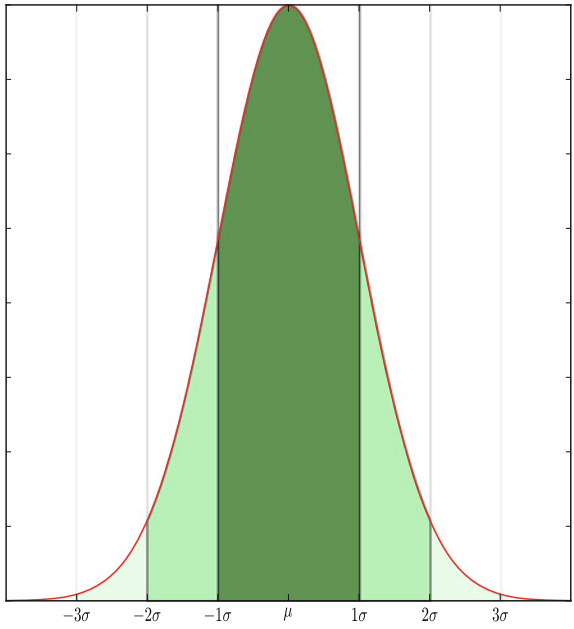


Fig. C.2 Continious Gaussian function

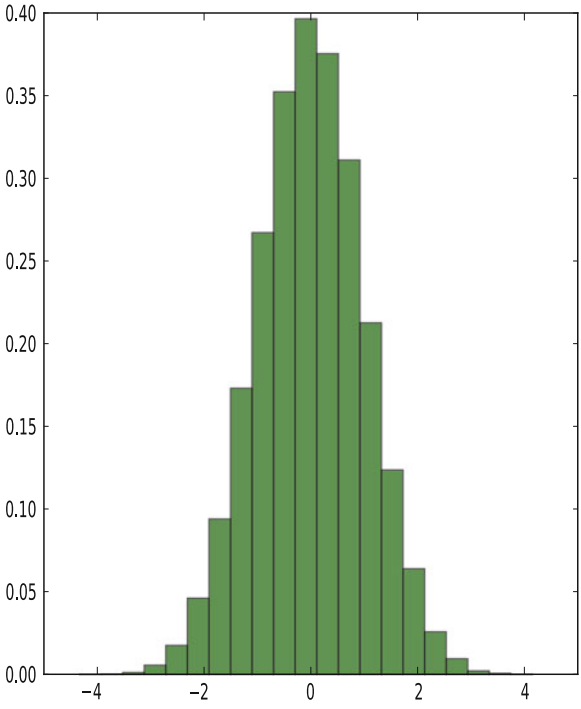


Fig. C.3 Histogram or probability mass function

Appendix D
4-AM Demodulator

In this example we show the mathematical operations done by a demodulator for the 4 amplitude modulation (AM). Within a 4-AM modulation process always $Q = 2$ bits are mapped to one symbol s_j with $s_j \in \{-3, -1, 1, 3\}$. Each value (amplitude) occurs equally likely since we assume an equal distributed binary source for the mapping process. The mapping form binary input variables x_i, x_{i+1} to the resulting symbol s_j and the transmission via an AWGN channel is shown in Fig. D.1. The figure shows only the transmission of one symbol s_j, which will be received as a noise corrupted real valued y_i.

The task of the demodulator is to calculate an a posteriori probability with respect to a received value y_j. The demodulator can calculate the result on symbol level with $P(s_j|y_j)$ or directly on bit level using $P(x_i|y_j)$, $P(x_{i+1}|y_j)$, as introduced in Sect. 2.2. Table D.1 summarizes the different expressions for demodulation on bit and symbol level in the probability and the log-likelihood domain. In order to understand the different possibilities and to derive the calculations of the demodulation,we have to review the 4-AM transmission and receiving process.

In Fig. D.2 we have plotted the transmission statistics for submitting equally distributed 4-AM symbols via an AWGN channel with $\sigma^2 = 1$. The Fig. D.2a–f will help to illustrate the terms joint probability, marginal distribution, and conditional distribution, the understanding of which is necessary to calculate the output of a higher order demodulator, here 4-AM demodulator. All sub-figures represent results of Monte Carlo simulations for a limited number of experiments (10,000), which thus results in inaccuracies of the corresponding histogram, probability mass functions, or scatter plot visualizations. We assume the input to the channel as a random variable S with an equal occurrence or each value $s \in \{-3, -1, +1, +3\}$, which results as well for the output of the AWGN channel a random received variable Y.

- Figure D.2a shows the so called scatter plot of the transmission, with the input data s on the x-scale and the corresponding observed output y data on the y-scale. Visualizing the probability mass function of $p(s, y)$ would require a third dimension to represent the frequency with respect to a quantized y realization.

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3, © Springer Science+Business Media New York 2014

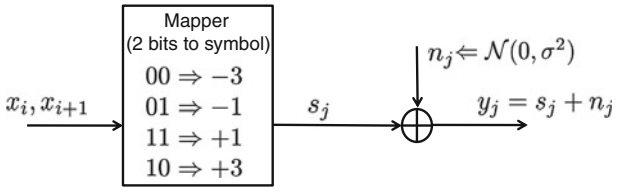


Fig. D.1 Illustration of the 4 level modulation and transmission via an AWGN channel

Table D.1 Demodulator possibilities with respect to higher order modulation, here the 4-AM example

Demodulator	Bit-level	Symbol level
	$P(x_i = 0\|y_j)$	$P(s_j = -3\|y_j)$
Probability domain	$P(x_i = 1\|y_j)$	$P(s_j = -1\|y_j)$
	$P(x_{i+1} = 0\|y_j)$	$P(s_j = +1\|y_j)$
	$P(x_{i+1} = 1\|y_j)$	$P(s_j = +3\|y_j)$
		$ln\frac{P(s_j=-3\|y_j)}{P(s_j=+1\|y_j)}$
Log-likelihood domain	$ln\frac{P(x_i=0\|y_j)}{P(x_i=1\|y_j)}$	$ln\frac{P(s_j=-1\|y_j)}{P(s_j=+1\|y_j)}$
	$ln\frac{P(x_{i+1}=0\|y_j)}{P(x_{i+1}=1\|y_j)}$	$ln\frac{P(s_j=+1\|y_j)}{P(s_j=+1\|y_j)} = 0$
		$ln\frac{P(s_j=+3\|y_j)}{P(s_j=+1\|y_j)}$
Comment	Information loss	full APP information

- Figure D.2b shows the simulated mass distribution of $p(y)$. The probability of each bin is shown on the x-scale, while the received random variable y for small ranges is given on the y-axis. $p(y)$ is the marginal probability of $p(x, y)$ which can be obtained by summing over all possible realizations $p(y) = \sum_{s\in\{-3,-1,1,3\}} p(s, y)$
- Figure D.2c shows the obtained mass distribution of $p(y|s = 1)$. The mean value of this distribution is at $\mu = 1$, exactly the value of the fixed channel input $s = 1$.
- Figure D.2d shows the distribution of the input variable s. Each value should occur equally likely, the inaccuracies in the graph are due to the limited amount of simulated data. $p(s)$ can be seen as well as the marginalization of $p(s, y)$ which is $p(s) = \sum_{y\in\Omega_y} p(s, y)$.
- Figure D.2e, f show the conditional probabilities for each possible channel input under the condition of the received variable $Y < 1$ (Fig. D.2e) or $Y > 1$) (Fig. D.2f) was observed.

Figure D.2a–f are now utilized to explain the demodulator process. First we will derive the demodulator calculations in the probability domain on symbol level.

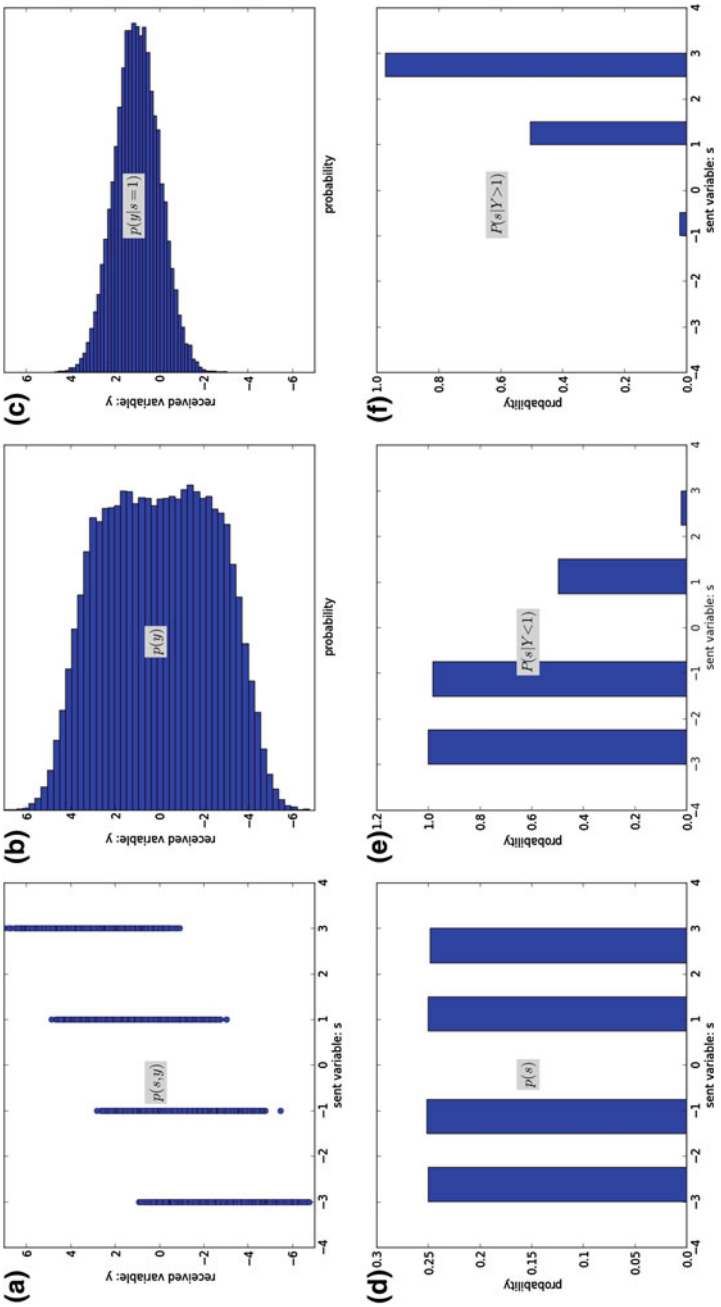


Fig. D.2 Illustration of a distribution over the received variable Y and sent variable S

4-AM Demodulation: Probability Domain on Symbol Level

As an example we assume a received value of $y = 3.4$ and calculate the probability that $s = 3$ was sent. Thus, we evaluate the conditioned probability $P(s = 3|y = 3.4)$. We can not directly calculate this probability and first apply Bayes' theorem, resulting in

$$P(s = 3|y = 3.4) = \frac{P(y = 3.4|s = 3) \cdot P(s = 3)}{P(y = 3.4)}. \tag{D.1}$$

The most difficult part to evaluate in this equation is $P(y = 3.4)$, since we need the full knowledge of the density function $p(Y)$. However, we eliminate this part by introducing a further condition. The sum of probabilities of all possible sent symbols has to be one, i.e. $\sum_{s\in\{-3,-1,1,3\}} P(s|y = 3.4) = 1$. We know that $P(y = 3.4)$ occurs in each term of the sum and is independent of the sent symbol. Furthermore, each symbol is equally likely.

$$P(s = 3|y = 3.4) = \frac{P(y = 3.4|s = 3)}{\sum_{s\in\{-3,-1,1,3\}} P(y = 3.4|s)} \tag{D.2}$$

The sum in the denominator reflects the introduced normalizing condition on each symbol. This equation can now be solved by utilizing the shifted Gaussian pdf of Eq. C.11 with the mean value $\mu = s$ and the received value $y = 3.4$.

$$p(y = 3.4|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(3.4 - s)^2\right) \tag{D.3}$$

For each received value y_j and each symbol $s_j \in \{-3, -1, +1, +3\}$ we have to evaluate this equation. The final corresponding probability results for each are shown in Fig. D.3. The figure shows the demodulation of a 4 amplitude modulation (AM) scheme. Each sub-figure shows the results for 4 different noise levels σ^2. The figures show the probability for each symbol as a function of the received value y. Thus, the left sub-figure shows the probability of $P(y|s = -3)$. On the x-axis the various values for a received y are shown, on the y-axis the corresponding probability conditioned on a sent symbol. It is obvious that the probabilities for $P(y|s = -3)$ and $P(y|s = +3)$ are axially symmetrical.

4-AM Demodulation: Log-Likelihood Ratios on Symbol Level

The log-likelihood ratio was already introduced in the binary case, here we show as well it's advantage on symbol level.

$$\lambda(s|y) = ln\left(\frac{P(s|y)}{P(s_{norm}|y)}\right) \tag{D.4}$$

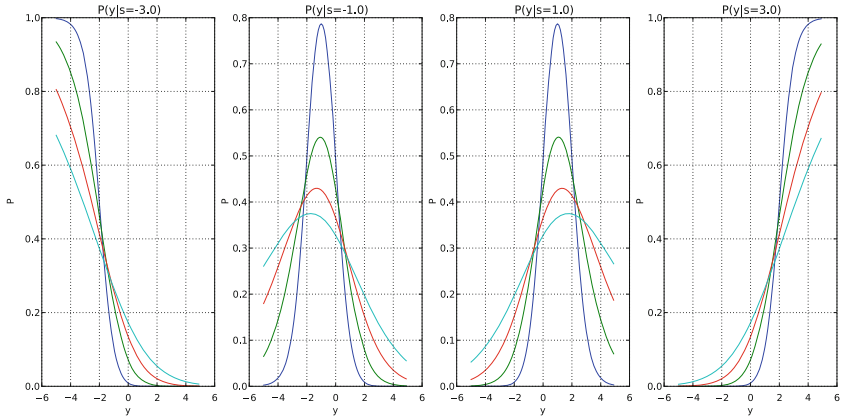


Fig. D.3 Illustration of the demodulation of a 4 amplitude transmission. Each *subfigure* shows the results for four different noise levels σ^2. Each figure shows the probability for one symbol having been the channel input as a function of the received channel output y

Equation D.4 represents the LLR on symbol level, each normalized to the symbol s_{norm}. The normalizer symbol can be chosen arbitrarily, so without loss of generality in the following we assume $s_{norm} = +1$. Starting form Eq. D.2 we can see that the LLR on symbol level results in:

$$\lambda(s|y) = -\frac{1}{2\sigma^2}(y-s)^2 + \frac{1}{2\sigma^2}(y-s_{norm})^2 \tag{D.5}$$

$$\lambda(s|y) = \frac{1}{\sigma^2}y(s-s_{norm}) + \frac{1}{2\sigma^2}((s_{norm})^2 - (s)^2) \tag{D.6}$$

Figure D.4 shows the same information as Fig. D.3, only in the LLR domain. For each symbol and noise level there exists a linear dependency. All lines intersect the LLR $= 0$ value at $\sqrt{s - s_{norm}}$ while the angle of the line depends on σ^2 and the corresponding symbol. This linear dependencies is of big advantage for the hardware realization. As well for the binary LLR value the linear relation exists, as derived in Eq. 2.22.

4-AM Demodulation: Log-Likelihood Ratios on Bit Level

An other possibility for the demodulator is to calculate the bit level LLRs, which can be calculated from symbol probabilities. In many practical applications the LLRs are required on bit level since the channel code itself is working as well on bit level. For independent bits x_i, the probability $P(x_i = 1|y_j)$ is obtained by

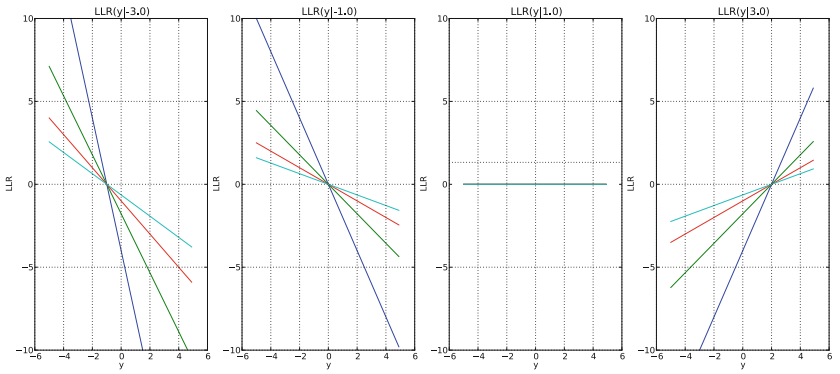


Fig. D.4 Illustration of the demodulation of 4 amplitude transmission. Each *subfigure* shows the results for 4 different noise levels σ^2, the log-likelihood ratios of each symbol are normalized to $s = 1$

$$P(x_i = 1|y_j) = \sum_{\forall s_j|x_i=1} P(s_j|y_j) \tag{D.7}$$

In this section we do not skip the indices to stress the fact that one symbol j comprises the information of two bits at bit position $i, i+1$. $\forall s_j|x_i = 1$ defines all symbols s_j which results from the mapping with the mapped bit at position i to be a one. Using Bayes' theorem, $P(s_j|y_j)$ can be expressed as

$$P(s_j|y_j) = \frac{P(s_j) \cdot P(y_j|s_j)}{P(y_j)} \tag{D.8}$$

Furthermore, by utilizing the log-likelihood ratio:

$$\lambda(x_i|y_j) = ln \frac{\sum_{\forall s_j|x_i=0} P(y_j|s_j) \cdot P(s_j)}{\sum_{\forall s_j|x_i=1} P(y_j|s_j) \cdot P(s_j)} \tag{D.9}$$

If additional and independent a priori information $\boldsymbol{P}(x_i)$ for each bit is available**,** then the symbol information is the product of probabilities of Q bit positions mapped to the symbol.

$$P(s_j) = \prod_{i}^{i+Q-1} P(x_i) \tag{D.10}$$

However, within our example of a 4-AM demodulator we assume the a priori information of $P(s_j)$ to be constant, thus this information cancels out. As already mentioned, every symbol in the 4-AM modulation scheme encodes two bits. The calculation of the bit LLR is different for these two bits, depending on their bit position. Assume the bits x_i and x_{i+1} are mapped to the symbol s_j. If $x_i = 0$, we

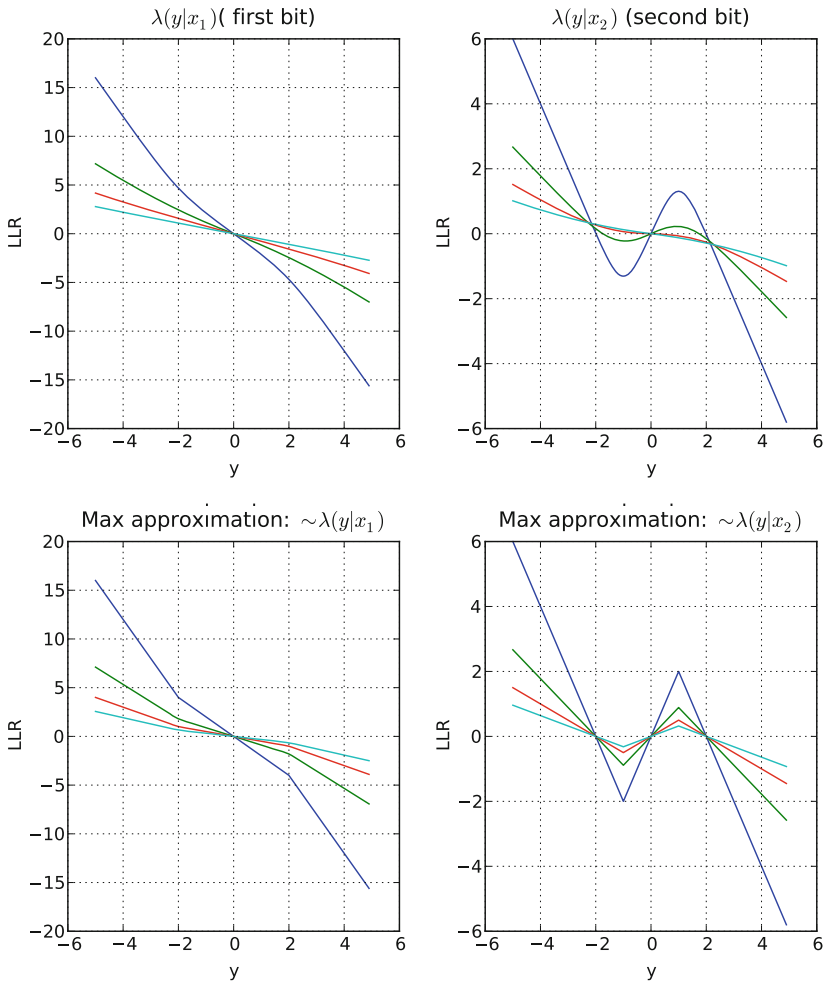


Fig. D.5 Illustration of the bit demodulation of 4-AM transmission. Each *subfigure* shows the results for four different noise levels σ^2. The true bit LLRs are shown as a function of y. Furthermore a possible hardware approximation is shown

know that $s_j = -3$ or $s_j = -1$, while if $x_i = 1$, then $s_j = +1$ or $s_j = +3$. Thus, the LLR for x_i can be calculated by:

$$\lambda(x_i|y_j) = ln\frac{P(y_j|s_j = -3) + P(y_j|s_j = -1)}{P(y_j|s_j = +1) + P(y_j|s_j = +3)} \tag{D.11}$$

The second bit $x_{i+1} = 0$ has a different mapping, see Fig. D.1. The resulting LLR can be evaluated by:

$$\lambda(x_{i+1}|y_j) = ln\frac{P(y_j|s_j=-3)+P(y_j|s_j=+3)}{P(y_j|s_j=-1)+P(y_j|s_j=+1)} \tag{D.12}$$

In Fig. D.5 the LLR values for both bit positions within a symbol are shown. The results clearly show that one bit is protected better by the modulation scheme, since the resulting values are unambiguous and can result in higher magnitude values. For the second bit some LLR values result from three different y values.

Figure D.5 shows as well a possible hardware approximation of the LLR calculation. This approximation takes only the maximum probability within the nominator and denominator calculation into account, i.e.

$$\lambda(x_i|y_j) = ln\frac{\max\limits_{\forall s_j|x_i=0}\left\{P(y_j|s_j)\cdot P(s_j)\right\}}{\max\limits_{\forall s_j|x_i=1}\left\{P(y_j|s_j)\cdot P(s_j)\right\}} \tag{D.13}$$

This kind of approximation technique is typically denoted as Max-Log approximation and is derived in more general context in Sect. 3.3.4.

References

1. Vogt, T.: A reconfigurable application-specific instruction-set processor for Trellis-based channel decoding. Ph.D. thesis, University of Kaiserslautern (2008)
2. Alles, M.: Implementation aspects of advanced channel decoding. Ph.D. thesis, University of Kaiserslautern (2010)
3. European Telecommunications Standards Institute (ETSI): Digital video broadcasting (DVB): interaction channel for satellite distribution systems—ETSI EN 301 790 V1.4.1. http://www.dvb.org
4. European Telecommunications Standards Institute (ETSI): Digital video broadcasting (DVB): interaction channel for digital terrestrial television (RCT) incorporating multiple access OFDM—ETSI EN 301 958 V1.1.1. http://www.dvb.org
5. IEEE 802.16e: Air interface for fixed and mobile broadband wireless access systems. IEEE P802.16e/D12 Draft (2005)
6. IEEE 802.3: LAN/MAN CSMA/CD access method. http://standards.ieee.org/getieee802/802.3.html
7. WiMedia Alliance: Multiband OFDM physical layer specification, release candidate 1.5 (2009)
8. IEEE 802.15.3c: Part 15.3: wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPANs). IEEE 802.15.3c-2009 (2009)
9. Brack, T.: Application and standard driven LDPC code decoder development. Ph.D. thesis, University of Kaiserslautern (2008). ISBN 978-3-939432-72-2
10. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, Heidelberg (2006)

Index

A
4-AM, 249
Address port, 74
Algorithmic complexity, 212
Approximation function, 112
A priori information, 63
Architecture
  comparison, 224
  LDPC, 163
  parallel check node, 166
  serial check node, 168
  turbo codes, 131
Area, 76, 217
ASIC, 4
ASIP, 5
Aspect ratio, 78
AWGN, 18, 19

B
Backward recursion, 59
BICM, 17
  MIMO, 185, 187
Binary phase shift keying, 18
Bit error rate, 22
BPSK, 17
Branching, 45
Butterfly unit, 110

C
Channel code, 13
Channel coding, 37
Channel reliability factor, 129
CMOS, 2
Codeword, 40
Coding gain, 23
Column mux, 78
Communication channel, 9
Communications efficiency, 211
Communications performance, 68
  LDPC, 153
  turbo code, 125
Compare select, 110
Complexity
  algorithmic, 212
  implementation, 213
Conditional probability, 244, 252
Control flow, 72
Convolutional code
  standards, 56
Convolutional codes, 52
  3GPP, 53
Cycle time, 78

D
Data flow, 97
Data path, 72, 97
  parallel processing, 106
  unrolling, 106
Data path: parallel MAP, 106
Data path: serial MAP, 100, 104
Data port, 74
Decoding
  LDPC, 150
  turbo code, 120
Demodulator, 18, 24
  4AM bit level, 253
  4AM symbol level, 252
  bit level, 249
  BPSK, 27
  symbol level, 249
Design flow, 68
Design language, 8
Design space exploration, 68, 76, 164, 219

F. Kienle, *Architectures for Baseband Signal Processing*,
DOI: 10.1007/978-1-4614-8030-3, © Springer Science+Business Media New York 2014

metrics, 213
Downlink, 7
DRAM, 73
DSP, 5

E
E_b/N_0, 22
Ergodic channel, 34
Error vector, 39
ETSI, 6
EXIT chart, 125
Explicit enumeration, 45
Extrinsic information, 63

F
Fixed-point, 129
turbo code, 128
Flexibility, 211
Floating point, 69
Forward recursion, 59
FPGA, 5
Frame error rate, 22

G
3GPP, 6
Gaussian distribution, 246
Gaussian variable, 246
Generator matrix, 40
Girth, 154
Golden reference model, 71
GOPS, 216
GPP, 4
GPU, 5
Gray mapping, 25
GSM, 6

H
Hamming code, 41
Hamming distance, 38
Hard-values, 26
Hardware
approximation
function, 113, 115
convolutional
codes, 215
LDPC, 215
turbo code, 215
High-level synthesis, 8
HSPA
TC performance, 238

I
I/O, 71
IC, 2
Implementation efficiency, 211
Implementation style, 71
Implicit enumeration, 45
Individual components, 67
Inner receiver, 10
Integer program, 44
Interface, 71
Interleaver, 87
block interleaver, 87
cyclic block interleaver, 87
LTE, 87
random, 87
UMTS, 87
Interleaver table, 87
Iteration control, 142
ITRS, 3
ITU, 6

J
Jacobian logarithm, 51

L
Latency, 71
LDPC, 147
encoder, 161
irregular, 148
multi-edge type, 158
quasi cyclic, 155
regular, 148
standards, 147
LDPC decoder, 171
parallel, 164
partly parallel, 165
quantization, 172
scheduling, 172
Linear block code, 39
Log-likelihood ratio, 24
Log-MAP, 51
Lookup table, 112
LP relaxation, 45
LTE
TC performance, 238

M
Max-Log-MAP, 51, 59
trellis, 58
Maximum likelihood criterion, 42
optimization problem, 44

Maximum likelihood decoding
repetition code, 46
single parity check code, 46
Memory, 72, 73
Memory access conflict, 93
Memory fragmentation, 84
Memory hierarchy, 84
Metrics, 211, 214
MIMO, 33, 185
APP, 189
data rate, 188
decision tree, 193
ML, 189
transmitter, 187
MISO, 33
ML decoding
trellis, 55
MMSE, 189
Modulator, 24
Monte Carlo, 23
Moore's law, 3
MOSFET, 2
Multi-edge type LDPC, 158

N
Next iteration initialization, 140

O
Outer receiver, 9

P
Parallel interleaving, 140
Parity bits, 41
Parity check matrix, 40, 160
hardware mapping, 169
Power, 76, 217
Probability, 243
distribution, 244
mass function, 245
Processing unit, 108
Puncturing, 118
PVT, 179

Q
Quality of Service, 22, 68
Quantization, 70
turbo code, 137

R
Radix-4, 140
Random variable, 243
Receive diversity, 33
Recursion unit, 109, 140
Reference model, 69
Repetition code, 46

S
SIMO, 33
Single parity check code, 46
SMAP, 131, 138
SNR, 22
Soft-input soft-output, 62
Soft-values, 26
SRAM, 73
area, 79
cycle time, 79
dual port, 74
power, 79
single port, 74
Static channel, 34
Sub-optimal algorithm, 70
Symbol-by-symbol MAP criterion, 42
Symbol-by-symbol MAP decoding, 49
single parity check code, 49
Syndrom, 40
Systematic bits, 41
Systematic code, 41

T
Tanner graph, 148
Technology, 76, 224
Throughput, 71
Transport transmission interval (TTI), 90
Trellis, 54
TSMC, 3
Turbo code, 117
decoding, 120
encoder, 117, 118
FER performance, 126
fixed-point, 128
HSPA performance, 238
LTE performance, 127, 238
standards, 118

U
Uplink, 7
UWB, 220

V
Viterbi algorithm, 48, 56
VLSI, 2

W
Windowing, 134, 139
Word depth, 76
Word width, 76

X
XMAP, 106, 138

Z
Zero forcing, 189