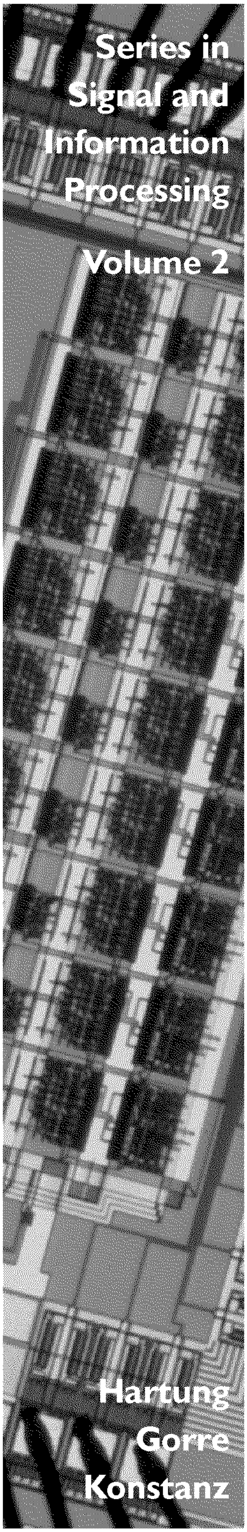


Diss. ETH No 13879

On the Design of Analog VLSI Iterative Decoders

A dissertation submitted to the
Swiss Federal Institute of Technology, Zürich
for the degree of
Doctor of Technical Sciences

presented by

Felix Lustenberger

Ing. en Microtechnique dipl. EPFL
born on May 31, 1969
citizen of Kriens (LU) and Honau (LU)

accepted on the recommendation of
Prof. Dr. George S. Moschytz, examiner
Prof. Dr. Hans-Andrea Loeliger, co-examiner
Prof. Dr. David A. Johns, co-examiner

November 2000

Diss. ETH No 13879

On the Design of Analog VLSI Iterative Decoders

A dissertation submitted to the
Swiss Federal Institute of Technology, Zürich
for the degree of
Doctor of Technical Sciences

presented by

Felix Lustenberger

Ing. en Microtechnique dipl. EPFL
born on May 31, 1969
citizen of Kriens (LU) and Honau (LU)

accepted on the recommendation of
Prof. Dr. George S. Moschytz, examiner
Prof. Dr. Hans-Andrea Loeliger, co-examiner
Prof. Dr. David A. Johns, co-examiner

Hartung-Gorre Verlag, Konstanz, November 2000

Für Rita, Simon und David

Acknowledgements

First of all I would like to thank my doctoral father Prof. Dr. George S. Moschytz for his confidence in me and my work by giving me a large scientific freedom and for always having the door of his office open in case of doubts and setbacks. I very much admire his human kindness to create a fruitful working environment within the Signal and Information Processing Laboratory.

I am deeply indebted to my mentor and friend Prof. Dr. Hans-Andrea Loeliger who set the initial spark to this fascinating research project. With his never ending enthusiasm he led me safely through all the periods of frustration and exhaustment which are common to ambitious research projects. I very much appreciated the long technical and sometimes also philosphical discussions that broadened my understanding of coding theory, data communications and signal processing.

Many thanks also to Prof. Dr. David A. Johns from the University of Toronto, Canada, for having accepted to serve as a competent co-examiner for the present doctoral dissertation.

My special thanks go to Markus Helfenstein and Felix Tarköy of our research team. Their maturity, both technically and personally, substantially added value to the results of this interdisciplinary research. It was and still is a great pleasure for me to share ideas and dreams together.

All the colleagues at the laboratory always helped to create an intellectually very stimulating and familiar working environment. I especially wish to thank my roommates and friends Hanspeter Schmid and Pascal Vontobel, but also Dieter Arnold, Marcel Joho, Daniel Lippuner, Heinz Mathis, and Stefan Moser for the time they invested into long and profound discussions on many different topics. I also enjoyed and learned a lot from the collaboration with Georg Fromherz and Ermanno Schinca whose diploma work I supervised.

I would also say ‘thank you’ to Max Dünki, who keeps our Sun workstation cluster constantly running and up-to-date. With great pleasure I also acknowledge the assistance with the design of printed circuit boards and measurement setup provided by the technical staff of the laboratory, in particular, Felix Frey, Patrick Schweizer, and Thomas Schärer.

Whenever there were problems that we were not able to solve at the laboratory, I always found kind persons whithin the department of electrical engineering willing to help me. The following persons (in alphabetical order) provided me with practical solutions to many problems: Christoph Balmer, Hubert Käslin, Ruedi Köppel, and Andreas Wieland from the Design Center; Armin Deiss, Norbert Felber, Clemens Hammerschmied, Hanspeter Mathys, Michael Oberle, Dirk Pfaff, and Robert Reutemann from the Integrated Systems Laboratory (IIS); Didier Cottet and Michael Scheffler from the Electronics Laboratory (IfE); Geert Bernaerts and Etienne Hirt from Art of Technology, a spin-off of IfE. Thank you very much to all of you for your help.

My deepest gratitude goes to my wonderful family. My wife and best friend Rita always supported me during the time of working on this dissertation, which is just simply as hard as doing the research itself. For months, my little sons Simon and David had to share their father on weekends only. I would also thank my parents who taught me in my younger years the curiosity and the passion to know how things work. All of them gave me the courage to finish the work.

This work was financially supported by the Swiss National Science Foundation under Grants 21-49619.96 and 20-55761.98, and by ETH Zurich under Grant 41-2639.5.

Abstract

The rapidly growing electronic networking of our society has created the need for a high-speed and low-power data communications infrastructure. Both voice and data communications have been made available for the mobile user. Additionally, more complex coding schemes and decoding algorithms have been introduced to protect the user data from corruption during the transmission over a communications channel. The aim of all these new coding and decoding approaches is to meet the theoretical channel capacity limit to make a better use of the signal power and channel bandwidth. The iterative probability-propagation-type algorithms that are used to decode state-of-the-art codes such as Turbo codes and low-density parity-check codes create the need for a tremendous computational power. Often, the computational complexity can not be implemented with a traditional digital design approach and a given power budget.

This thesis discusses the efficient implemention of high-performance decoding algorithms in analog VLSI technology. The building blocks are very simple analog translinear circuits that implement vector multipliers with basically only one transistor per element of the outer product of two discrete probability distributions. The presented analog probability propagation networks made of these building blocks are a direct image of the underlying sum-product algorithm. The design of these analog networks follows a heavily semiconductor-physics-centered bio-inspired design approach, that exploits, rather than fights against, the inherent nonlinear behaviour of the basic semiconductor devices. By using such a bio-inspired design approach, the performance of these networks in terms of speed or power-consumption or both is increased by at least a factor of 100 compared to digital implementations. Despite the use of very-low-precision circuit devices, a remarkable system-level accuracy can be achieved by such a large, highly-connected analog network.

The first part of the thesis discusses the background of channel coding and decoding and the theoretical foundations of factor graphs and the sum-product algorithm, which operates by message passing on such graphs. This part provides a brief introduction to the information-theoretic aspects of the interdisciplinary research effort.

The second part of the thesis is devoted to the actual transistor level implementation of the sum-product algorithm using very simple analog-VLSI computational building blocks. This part discusses the design-oriented aspects of the research, however, it relies heavily on the information-theoretic concepts introduced in the first part.

Finally, we present practical designs and design studies of several decoding networks. Algorithmic simulations, circuit simulations, and, where available, measurement results of the implemented decoding networks are presented. Two of the decoder examples were actually fabricated in a $0.8\,\mu\mathrm{m}$ BiCMOS process. Additionally, application-specific design problems are discussed.

The thesis is finished with a summary of the achieved results and a presentation of future research propositions in the field of analog decoding.

Keywords: Iterative decoding, low-density parity-check (LDPC) codes, repeat-accumulate (RA) codes, trellis codes, Turbo codes, maximum-*a posteriori* probability (MAP) decoder, maximum-likelihood (ML) sequence detection, sum-product algorithm, Viterbi algorithm, probability propagation, factor graphs, analog VLSI technology, bio-inspired networks.

Kurzfassung

Die schnell wachsende elektronische Vernetzung unserer Gesellschaft hat einen grossen Bedarf an schneller und leistungsarmer Datenkommunikationsinfrastruktur erzeugt. Sowohl Sprach- wie auch Datenkommunikationsmittel sind inzwischen für den mobilen Benutzer zugänglich. Zusätzlich werden laufend komplexere Kodierungsverfahren und Dekodieralgorithmen eingeführt, um die Benutzerdaten vor Übertragungsfehlern zu schützen. Das Ziel dieser neuen Kodier- und Dekodierverfahren ist das Erreichen der theoretischen Kanalkapazitätsgrenze, damit die vorhandene Signalleistung und Kanalbandbreite optimal ausgenutzt werden können. Die bei der Dekodierung der dem aktuellen Stand der Technik entsprechen Kodes (wie zum Beispiel Turbo Kodes und Kodes mit dünn besetzter Paritätsprüfmatrix) verwendeten iterativen Wahrscheinlichkeits-Fortpflanzungs-Algorithmen benötigen eine enorme Rechenleistung. Diese Rechenkomplexität kann bei vorgegebenem Leistungsbudget oft nicht mehr mit traditionellen digitalen Entwurfsansätzen erreicht werden.

Die vorliegende Dissertation beschäftigt sich mit der effizienten analogen VLSI-Implementation von leistungsstarken Dekodieralgorithmen. Die Baublöcke der vorgestellten Technik sind sehr einfache analoge translineare Schaltungen zur Implementierung von Vektormultiplizierern. Dabei wird praktisch nur ein Transistor zur Bildung eines Elementes des äusseren Produkts von zwei diskreten Wahrscheinlichkeitsverteilungen benötigt. Die aus den Baublöcken aufgebauten analogen Wahrscheinlichkeits-Fortpflanzungs-Netzwerke sind ein direktes Abbild des zugrundeliegenden Summe-Produkt-Algorithmus'. Der Entwurfsprozess dieser analogen Netzwerke verfolgt einen auf die Halbleiterphysik ausgerichteten und von der Biologie inspirierten Entwurfsansatz, wobei das grundlegend nichtlineare Verhalten von Halbleiterelementen ausgenutzt wird anstatt dagegen anzukämpfen. Indem dieser bio-inspirierte Entwurfsansatz verfolgt wird, kann das Leistungsverhalten in Bezug auf den Stromverbrauch oder die Geschwindigkeit oder beides, verglichen mit einer äquivalenten digitalen Lösung, um mindestens einen Faktor 100 erhöht werden. Obwohl nur Bauelemente mit sehr schlechten Präzisionseigenschaften verwendet werden, erreichen diese hochgradig verbundenen analogen Netzwerke eine erstaunliche Systemgenauigkeit.

Der erste Teil der Dissertation vermittelt Hintergrundinformationen zum Thema Kanalkodierung und -dekodierung und liefert die theoretischen Grundlagen über Faktorgraphen und den Summe-Produkt-Algorithmus, der gemäss dem sogenannten Nachrichten-Übertragungs Prinzip auf solchen Graphen angewandt wird. Dieser Teil gibt eine kurze Einführung in die informationstheoretischen Aspekte der interdisziplinären Forschungsanstrengungen.

Der zweite Teil der Arbeit ist der eigentlichen Implementierung auf Transistorebene des Summe-Produkt-Algorithmus' mittels sehr einfacher Rechenbaublöcke gewidmet. Dieser Teil diskutiert somit die entwurfsorientierten Aspekte der Arbeit. Er nimmt jedoch sehr stark Bezug auf die im ersten Teil vorgestellten informationstheoretischen Konzepte.

Schliesslich werden im dritten Teil praktische Ausführungen und Entwürfe von verschiedenen Dekodiernetzwerken besprochen. Es werden dabei algorithmische Simulationen, Schaltungssimulationen und, soweit vorhanden, Messresultate der von uns gebauten Dekodiernetzwerke vorgestellt. Zwei dieser Dekoderbeispiele wurden in einer 0.8μm BiCMOS-Technologie fabriziert. Zusätzlich werden auch anwendungsspezifische Entwurfsprobleme besprochen.

Die vorliegende Dissertation wird durch eine Zusammenfassung der erzielten Resultate und Vorschläge für weitergehende Forschungsprojekte im Bereich der analogen Dekodierung abgerundet.

Stichwörter: Iterative Dekodierung, Kodes mit dünn besetzter Paritätsprüfmatrix, Repetitions-Anhäufungs-Kodes, Trellis-Kodes, Turbo-Kodes, Maximum-*a-posteriori*-Wahrscheinlichkeits-Dekoder, Maximum-Likelihood (ML)-Sequenzdetektion, Summe-Produkt-Algorithmus, Viterbi-Algorithmus, Wahrscheinlichkeitsfortpflanzung, Faktorgraphen, analoge VLSI-Technik, bio-inspirierte Netzwerke.

Contents

Chapter 1

Introduction

Motivation 1.1

Coding: why, where, and how

In the last few years the demand for efficient and reliable digital data transmission and data storage has tremendously increased. This trend has been accelerated by the emergence of high-speed data networks even at the local-area scale. Already in 1948, Shannon [1] showed that it is possible, by proper *encoding* of the information source, to reduce errors induced by a noisy channel to any desired level, without sacrificing the rate of information transmission or storage of a given channel, as long as the rate is below the so-called channel capacity. The term encoding in this context means that we add redundant information to our data stream. This type of coding is generally called channel coding.[1] The redundancy introduced by channel coding will help the decoding block of a receiver with finding the best decision for the sent data sequence. Decoding can be regarded as a projection of the possibly infinite number of received messages of the channel-output vector space onto the channel-input vector space of the codewords. As a very simple example we could analyze the situation as shown in Fig. 1.1. Every mark is assumed to be a valid configuration or codeword of our code whereas the received data can be any point in the plane. The individual codewords are schematically separated by decoding-region border-lines. The decoder block in the receiver path tries to match the incoming data, which is corrupted by channel noise, symbol interference, or other destructive events, to the nearest valid configuration, i.e., it will create

[1]The two other main types of coding are source coding (or data compression), where we try to reduce the redundancy of the data source, and cryptography which modifies the characteristic of the data-source such that unwanted observers cannot see the information content.

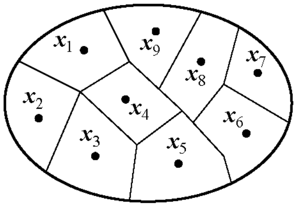


Figure 1.1 *A simple code viewed as discrete points in the plane of all messages possibly received by a data communications receiver.*

an estimate of the most probable codeword. This is a very simplistic picture of the purpose of coding, but it will help to see how coding transforms an arbitrary channel into an almost reliable bit pipe.

Complex coding gives more protection

In general, we can state that the more complex our coding scheme is constructed, the more protection we get from coding. On the other hand, decoding will become more complicated. The computational complexity of decoding for codes that try to reach the theoretical limits defined by Shannon [1] is growing more than linearly, i.e., quadratically or even exponentially. Today's state-of-the-art codes such as Turbo codes [2–4], low-density parity-check codes [5–8], and other similarly built codes need huge computational power to deliver real-time results.

Rapidly growing complexity seeks for new decoding techniques

Because of this rapidly growing computational effort, new decoding techniques and technologies are investigated. The problem can be tackled in different ways. The first and most obvious approach is to boost processing speed by using more sophisticated semi-conductor processes. Unfortunately, unless more parallelisms are introduced in the decoding system, the processing speed is just increasing linearly with the clock frequency. Alterations of the decoding system on the algorithmical level is a second and generally more successful approach to the complexity dilemma. But these alterations are in most cases made at the cost of precision, i.e., the decoding is no more ideal. This can be seen as a loss in the bit error-rate (BER). But for practical solutions, a trade-off can be found in many cases which satisfies more or less both parts, decoding speed and precision. The third approach, which gains more and more momentum, is the bio-inspired network-decoding approach.

Bio-inspired circuits may solve dilemma

Today's comprehension of the functioning of the human brain is that it consists of an agglomeration of neurons [9–11], each having a relatively poor precision, but highly interconnected. Comparable to this understanding, one can think of highly connected electrical networks consisting of very simple local processors which globally exchange imprecise information. Interestingly, both 'networks' can reach outstanding precision on the system level through a high degree of interconnectivity. In the case of the human brain, learning modifies and even densifies the connection pattern between the individual neurons. Thus, information storage seems to be a matter of a three-dimensional arrangement of individual cells far more complex than any known standard computer hardware. But nevertheless, electrical networks inspired by its biological counterparts and relatively simple compared to them promise achieving a very fast and robust systems behaviour. Thus, one of the sources that had the greatest influence on our motivation and inspiration is the bio-inspired background in general and Mead's outstanding work on neuromorphic systems in particular [11]. Mead's work showed clearly the direction to take if very efficient electronic solutions for processing analog signals are demanded. Beside a large academic interest, we find the bio-inspired design approach also in industrial products such as Logitech's trackball marble [12], OCR readers for banking applications [13], and combined angle/torque sensors for automotive applications [14].

1.2 Outline of this Thesis

The present thesis is divided into six chapters. This chapter gave some introductory comments on the general motivation of the implementation of analog decoders. Chapter 2 is devoted to a short review of the basic notions of coding theory, and the presentation of some inspiring sources with many citations of interesting literature. In Chapter 3, we present the algorithmic background and propose a new dissection of the general sum-product algorithm into generic trellis computation modules. In the fourth chapter, we fill the gap between the mathematical representation of the building blocks and their VLSI implementation. Thereby we introduce the generic transistor-level implementation of these modules, and we discuss many practical design aspects of large probability propagation networks composed of such modules. Then in Chapter 5, we present and discuss five designs of decoders for error-control codes. They represent the practical part of the thesis. The examples are completed and are discussed in various depth, mainly due to some time constraints of the whole research project. Beside an implementation using discrete BJT devices, we have designed two complete decoders in BiCMOS technology which have been tested partly. Additionally, we discuss a Viterbi-decoder design using our generic trellis calculation modules. Finally, we propose a plain CMOS implementation and discuss its properties by using high-level simulations. The content of the fifth, as well as of the fourth chapter are mainly results of original work.

We conclude the thesis by a summary of the results achieved during the whole research and give some comments on where future work may be directed and on where we see the future of analog probability calculation networks.

Chapter 2

Background Information

About Coding 2.1

Terms and definitions for the binary case

In this section, we will briefly introduce some terms and definitions of coding theory. The section primarily addresses circuit designers to help them understand what follows. We will restrict ourselves to the binary case, i.e., codes over the Galois field GF(2). This means that our information unit is the binary digit or bit. The extension to codes over GF(q) for $q > 2$ is generally possible and straightforward, but is omitted for the sake of brevity.

General Communication System 2.1.1

'Figure 1' of information theory

Text-books on coding such as [15, 16] often start with what is commonly known as 'Figure 1' in information theory [1]. Fig. 2.1 shows this system overview of a data communication (transmission or storage) system. An information source emits a sequence of binary digits (bits), called the uncoded sequence $\mathbf{u}$. This sequence is transformed into the coded sequence $\mathbf{x}$ by an encoder. We assume that during transmission over the channel (or the storage medium), the coded sequence $\mathbf{x}$ is corrupted by a noise vector $\mathbf{n}$. This assumption is correct if the noise is of additive nature and no inter-symbol interference is present, i.e., the channel is without a filter transfer function. Thus we will observe a noisy sequence $\mathbf{y}$ at the input of our decoder. The decoder then estimates the most probably sent data sequence $\hat{\mathbf{u}}$ using $\mathbf{y}$.

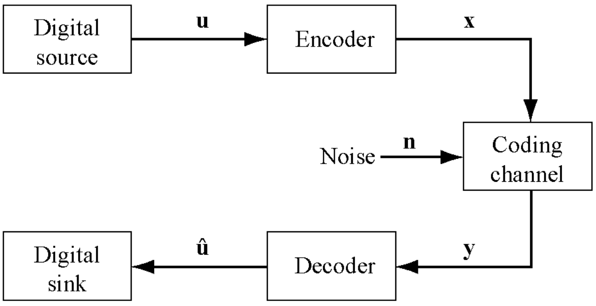


Figure 2.1 *Simplified model of an encoder/decoder system.*

2.1.2 Types of Codes

Different types of coding

In general, we distinguish between two main types of codes that are of common use today, *block codes* and *convolutional codes*. The output of a block encoder is strictly block oriented and is generated by combinatorial operations, whereas the convolutional encoders create data streams of possibly infinite length. Additionally, the output of a convolutional encoder is created by a finite state machine, i.e., the encoder incorporates memory that tracks the history of the incoming data bits. In the following we will briefly discuss the two cases and introduce some terms related to coding.

Block Codes

Definition of a block code

A binary block code is defined as an algebraic mapping from the vector space F^k over the Galois field $F = \mathrm{GF}(2)$ into the vector space F^n, i.e., a data sequence of length k is mapped onto a codeword of length n. If this mapping from one vector space to another is linear we speak of a *linear code*; otherwise the code is non-linear. We restrict ourselves to the most important concepts and thus omit intentionally the detailed presentation of the non-linear case.[1]

Notation: $[n,k]$ block code

The encoder for a *block code* cuts the incoming datastream into blocks of length k. In the binary case 2^k possible messages are encoded into n bit long codewords. Thus we speak of an

[1]One can show that capacity is actually achieveable with linear codes.

$[n,k]$ *block code*. A *linear block code* is entirely described by its generator matrix $\mathbf{G}$. A codeword is built using the relation

$$\mathbf{x} = \mathbf{u} \cdot \mathbf{G}, \tag{2.1}$$

where $\mathbf{x}$ and $\mathbf{u}$ are assumed to be row vectors. This assumption commonly used in coding theory is in contrast to the general notation in linear algebra. By using (2.1) we observe immediately that linear codes transform the all-zero input vector into the all-zero codeword. Equivalently, the codeword $\mathbf{x}$ always has to satisfy the relation

$$\mathbf{H} \cdot \mathbf{x}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}, \tag{2.2}$$

where $\mathbf{H}$ is a parity-check matrix. The rows of this matrix contain the information of which bits are checked by a given parity check, and the columns describe in which parity checks a given bit is involved. The parity-check matrix can be derived from the generator matrix $\mathbf{G}$.

Code rate of a block code

The *code rate* of a block code is defined as the proportion of the number of information-carrying bits compared to the total code-word length. If the generator matrix is a $k \times n$ matrix with full rank then the rate is given by

$$R = k/n. \tag{2.3}$$

Convolutional codes

Definition of convolutional codes

Like block codes, *convolutional codes* are defined as mapping from one vector space to another. But this time, the incoming and the outgoing data streams of an encoder can be of infinite length. An encoder for a convolutional code is shown in Fig. 2.2 as a *finite-state machine*, i.e., a sequential logic circuit, with a *memory order* of m. The generator matrix $\mathbf{G}$ of a convolutional encoder has a general form of

$$\mathbf{G}(D) = \begin{bmatrix} G_{11}(D) & G_{12}(D) & \cdots & G_{1n}(D) \\ G_{21}(D) & G_{22}(D) & \cdots & G_{2n}(D) \\ \vdots & \vdots & \ddots & \vdots \\ G_{k1}(D) & G_{k2}(D) & \cdots & G_{kn}(D) \end{bmatrix}, \tag{2.4}$$

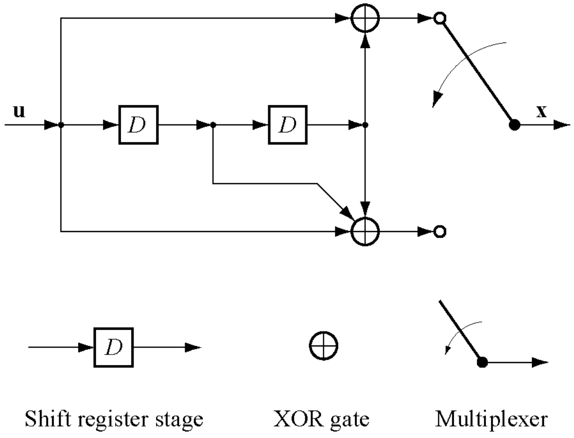


Figure 2.2 *Binary convolutional encoder with rate $R = 1/2$ and memory order $m = 2$.*

where each element $G_{ij}(D)$ represents a transfer function (Kronecker-delta response) of a linear discrete-time system (LDS) of order m:

$$G_{ij}(D) = \frac{a_m D^m + a_{m-1} D^{m-1} + \cdots + a_0}{b_m D^m + b_{m-1} D^{m-1} + \cdots + b_0}. \tag{2.5}$$

Code rate of a convolutional code

In the case of the encoder of Fig. 2.2, we observe two polynomials $G_{11}(D) = 1 + D^2$ and $G_{12}(D) = 1 + D + D^2$. The code rate of a convolutional code is defined by the input bits divided by the number of outcoming code bits or equivalently

$$R = k/n, \tag{2.6}$$

where k and n are the dimensions of $\mathbf{G}(D)$. In the example of Fig. 2.2, we have $R = 1/2$.

Increased complexity by higher-order memory

Typically, the code rate R is kept constant, whereas the memory order m is increased in order to combat channel noise. This means that the complexity of the code is increased to obtain more redundancy.

High code rates for better efficiency

In general, one wishes to keep the code rate close to unity, but still have strong error-protection capabilities. This creates constraints in the process of finding appropriate codes. Large code rates, i.e. larger than 0.9, are normally used for storage applications such as magnetic recording, where the information density on the storage medium is mainly limited by the used material and by the transmission speed. Note that the capacity (in bits per use) of a magnetic-recording channel and binary signaling in general is limited to unity. For data communication applications, the constraint on the code rate is somewhat less stringent, although one wishes to have good error-correcting capabilities without sending too much redundant information. But finally only the achievable throughput of the communication system is interesting for us. So often we encounter code rates of 1/2 and less for very noisy channels. Under these circumstances we may transmit the data symbols at a higher speed in order to achieve the desired communication rate. By doing so we intentionally allow the channel to make a certain amount of errors that can later be corrected by the decoder. Note also that the code rate can be greater than 1 if the transmitted code symbols are m-ary symbols and not binary any more. This is often the case in modem technology such as in the V.34, V.90 and xDSL standards [17–19].

Capacity of the channel

Definiton of the channel capacity

According to Shannon's 1948 pioneering paper [1], the capacity C of a given noisy channel is defined as the maximum possible transmission rate, i.e., the rate at which the source is properly matched to the channel. This abstract definition has very deep significance. It is always possible to send information at a rate lower than C through a channel with an error probability as small as desired by properly encoding the information source. This statement on controlled error probability is not true for rates above C. Shannons theorem on error-free transmission does not tell us how to make good codes, it only tells us the limit. Practical error control schemes have long been far away from this theoretical limit. Only the advent of the complex Turbo codes [2, 3] and their iterative decoding by belief propagation has brought us to some tenth of dBs above the Shannon limit. An even higher rate has been reached using very large low-density parity-check codes [20], which were invented by Gallager [5].

Systematic codes

Definition of a systematic code

Often, the uncoded data sequence is part of the codeword. If the uncoded information bits are transmitted together with the parity-check bits over the channel, the code is called *systematic*. But this must not be necessarily the case. The code can consist equally well of parity information only. In order to improve the code rate of a given code, sometimes a different approach is chosen: first the data sequence is encoded as usual, but then some bits are pruned and thus not transmitted over the channel. This encoding procedure can often be observed in Turbo coding.

Hamming distance

Definition of the Hamming distance

The distance between two codewords is the number of positions where they differ. The minimum distance d of a code is then defined as the minimum of all distances between any two codewords of the code. A code with minimum distance d is capable of correcting $\lfloor (d-1)/2 \rfloor$ errors, where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x. If d is even, the code can *simultaneously correct* $(d-2)/2$ errors and *detect* $d/2$ errors. This statement is strictly true for block codes. In the case of convolutional codes, the decoder may be confused if more than $(d-2)/2$ errors are present and may not recover anymore until the decoder is resynchronized with a known state.

The Hamming distance is called free distance for convolutional codes

The definition of the previous paragraph is only strictly valid for block codes. In the case of convolutional codes, we speak of the *free distance* of a code, but the meaning of this distance measure is basically the same [21].

2.1.3 Hamming Codes

Hamming codes are a whole class of linear block codes that can correct single errors. The error-correcting capability is given by its *Hamming distance* as defined before.

Definition of a Hamming code

According to [15], Hamming codes of length $n = 2^r - 1$ $(r \geq 2)$ are defined as having a parity-check matrix $\mathbf{H}$ whose columns consist of all non-zero binary vectors of length r, each used once. A Hamming code is thus an ($n = 2^r - 1,\ k = 2^r - 1 - r,\ d = 3$) block code. The Hamming code as it was first defined is the $[7,4,3]$ block code with the parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \qquad (2.7)$$

Equivalence of block codes

Any code consisting of rows that are created using linear combinations and column permutations of the original matrix $\mathbf{H}$ is said to be *equivalent*. There exist several extensions of the original Hamming code using redundant equations, for example the $[8,4,4]$ extended Hamming code. They generally perform better because of their larger minimum distance d.

Low-Density Parity-Check Codes 2.1.4

Defintion of an LDPC code

Low-density parity-check (LDPC) codes have a parity-check matrix $\mathbf{H}$ that is very sparse, i.e., there are only a few 1's in the matrix. According to Gallager's initial definition in 1963 [5], the parity-check matrix contains a small fixed number j of 1's in each column and another small fixed number k of 1's in each row.

LDPC codes may outperform Turbo codes

For good performance, large block lengths are required. MacKay [22] has shown recently that the number of 1's in the columns has not necessarily to be constant. Even better results are obtained by statistically varying the number of 1's within a column but still keeping the total number small. Recently, it has been shown by Richardson *et al.* [20] that very long low-density parity-check codes outperform comparably long Turbo codes by a distinct margin and come even closer to the theoretical capacity of a given channel.

Trellis Codes 2.1.5

In what follows, we will briefly present two different ways of temporally describing a convolutionally encoded sequence. The two graphical representations completely define the code. The code-tree representation graphically explodes for large

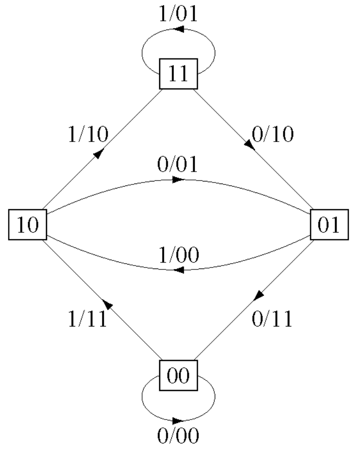


Figure 2.3 *The state-transition diagram for the convolutional encoder of Fig. 2.2.*

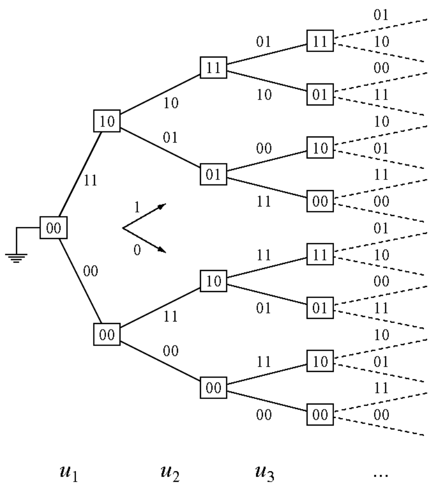
code sequences. Thus the more condensed trellis diagram is a solution to that representation problem.

Convolutional code represented by a code tree...

Assume the convolutional encoder of Fig. 2.2 and the corresponding state-transition diagram of Fig. 2.3. The initial content of the encoder memory is assumed to be 00. We may now draw a tree, as in Fig. 2.4, with branches labeled according to the outcoming bits of this encoder for any incoming data sequence $\mathbf{u}$. The boxed nodes of the tree denote the content of the encoder memory and will be called the *state of the encoder*. Following the upgoing branch at a given node means that we have encoded a binary 1, and a 0 otherwise. Going through the entire tree we can on the one hand read off the information sequence and on the other also collect the encoded sequence.

... or a trellis diagram

Because the size of the tree representation is rapidly outgrowing standard papersizes for larger code lengths, we might look for a more compact image of the code that is more quickly cognizable. The *trellis* representation is the solution to this problem. Since the output at time $t+1$ is simply defined by the state (i.e. the encoder memory) of time t and the new input digit, we can collapse all nodes showing the same state memory content at a given time instant. For our simple example of Fig. 2.4 this means that we have only $2^m = 2^2 = 4$ different states to depict.



The code tree for the convolutional encoder of Fig. 2.2. **Figure 2.4**

As shown in Fig. 2.5, the name choice seems to be obvious: the look of the graph is the same as the trellises we find in our gardens to fasten tall flowers and espalier trees to the wall. After having left the initial transitional trellis sections, the trellis of the code is simply built by concatenating identical trellis sections. Compared to a block code, the convolutional code has possibly infinitely long codewords. But often the code is manually terminated to the zero state by adding an appropriate number of zeros at the input of the encoder after a certain number of bits. By doing so, we may transform a convolutional code into a block code.

Description of a trellis section

A single section of the complete trellis diagram is caracterized by the left states S_t, the right states S_{t+1} and the branches connecting the left states and the right states with a characteristic pattern (Fig. 2.6). A branch between a left state and a right state indicates an allowed state transition of the encoding state-machine of Fig. 2.3. The labels on the branches directly indicate the incoming data and the encoder output.

General trellis codes

Trellis codes represent a huge class of codes that can be defined by trellis diagrams as defined above. The encoding of any trellis code can be done by a finite-state machine. The general trellis code may consist of many different trellis sections [21].

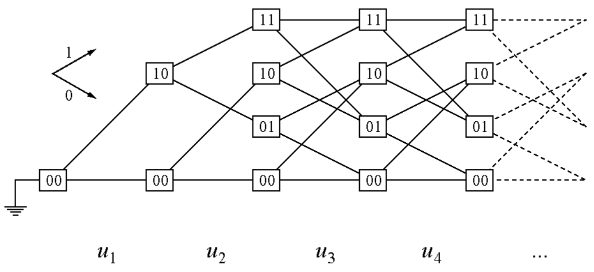


Figure 2.5 *Trellis representation of the code tree of Fig. 2.4.*

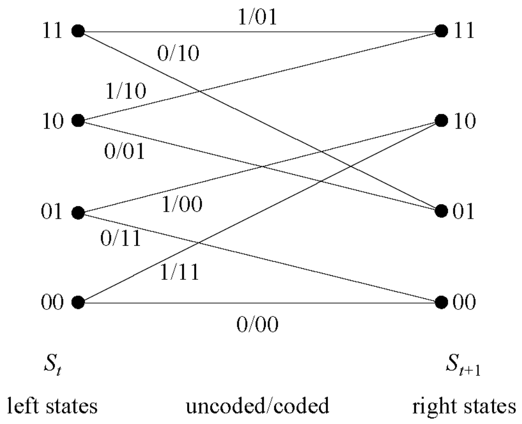
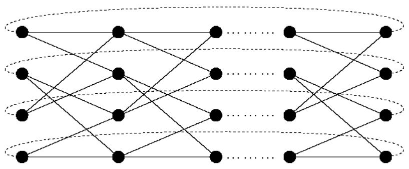


Figure 2.6 *One trellis section of the code of Fig. 2.2.*


Formation of a tail-biting trellis. **Figure 2.7**

Thus, the finite-state machine may be very complex and consist of interconnected simpler finite-state machines. In contrast to the general trellis codes, convolutional codes represent only a sub-class of trellis codes. The convolutional codes are linear codes since the output sequence is found by convolving the input sequence with the Kronecker-Delta response of the encoder. Convolutional codes have therefore a limited number of states and state transitions which are the same for all time instances. Hence, the finite state-machine is generally simple.

Tail-Biting Trellis Codes

Definition of a tail-biting trellis

A sub-class of the trellis codes are the so-called *tail-biting trellis codes*. This type of code is defined by the concatenation of a certain number of individual trellis sections. But instead of terminating the overall trellis at the beginning and at the end, the tail-biting trellis is formed by connecting the outgoing states of the last trellis section to the incoming states of the first trellis section. Fig. 2.7 shows such a tail-biting trellis which forms a closed ring structure. A valid codeword is then defined by a path starting in any state at a certain point, i.e., not necessarily the zero state, and terminating in the same state after one turn. Encoding by a convolutional encoder is somewhat more difficult than for the non-tail-biting case, since an additional condition for the closed path has to be met. The benefit of this closed structure is that no termination information, which can cause much overhead for convolutional codes of small blocklengths, has to be added.

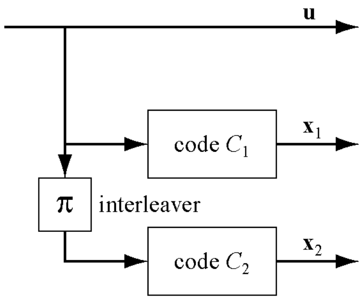
2.1.6 Turbo Codes

Parallelly concatenated Turbo codes

A recently discovered new class of concatenated codes, the so-called Turbo codes, has turned the view of coding theory upside down. In fact, until 1993, all types of codes known up to then were separated from Shannon's limit by well over 1 dB. Even the most advanced coding schemes, as for example the serial concatenation of constituent codes that has been used for state-of-the-art satellite communications, struggled with this imagined boundary. In 1993, Berrou *et al.* [2] presented their first article on Turbo codes. The imagined boundary was overstepped by the introduction of this new coding scheme that consists of two parallelly concatenated convolutional codes C_1 and C_2 connected by a bit-interleaving structure or permutation π, as schematically presented in Fig. 2.8. The excellent performance of Turbo codes is rooted in both the construction of the code and the corresponding iterative decoding techniques.

Functioning of Turbo codes

In fact, the high complexity of Turbo codes is distributed both over the constitutent convolutional codes and the interleaver. As we will see in Section 3.2.2, the interleaver can be seen as a complex connection pattern between the two parallelly concatenated convolutional codes. If we cut this connection pattern vertically at a given position and count the number of lines crossed by this section, we directly get the number of states of an equivalent convolutional code. As an example, we could assume that the section crosses m pattern connections of the interleaver. Under these conditions, an equivalent convolutional code would have at least 2^m states, since we did not take into account the number of states of the constituent convolutional codes so far. Hence, very complex concatenated codes can be constructed even if relatively low-complexity convolutional codes are used for their constituent codes. Furthermore, the interleaver allows, by spacial (or temporal, depending on the point of view) decorrelation of adjacent information pieces, the control of hard-to-correct errors. However note that interleaving structures for decorrelating error bursts are also used for other data transmission schemes, but there they are a pre-step before decoding, and hence, they are not involved in the decoding itself. This decorrelation is most important in the case of burst errors which are, for example, present in the case of mobile communication very close to a base station.



Principal ingredients of a Turbo code. **Figure 2.8**

A note on decoding of Turbo codes

Convolutional codes with the same complexity as Turbo codes are very difficult to decode by standard means (e.g., the Viterbi algorithm [23,24]), since the number of states grows exponentially with the memory order m. Using iterative decoding techniques significantly lowers the computational complexity. This is mainly due to the interleaver which creates most of the complexity in Turbo codes, and is a simple permutation operation.

Turbo codes show steep waterfall curves

The characteristic of Turbo codes is a very steep curve of bit-error rate (BER) versus signal-to-noise ratio (SNR). Such characteristic curves are also known as water-fall curves. In the case of parallel concatenation of constituent codes, the BER versus SNR curve flattens at higher SNR values to an error floor. This is due to the small minimum distance of parallely concatenated Turbo codes. Serially concatenated codes, in contrast to parallelly concatenated codes, have a higher free distance, thus the error-floor phenomenon is not present in this case. A comparison of BER curves of several typical parallelly and serially concatenated codes is shown in Fig. 2.9 (see also [21,25]).

2.1.7 Channel Models

Up to here, we have treated the coding channel of Fig. 2.1 as a black box. Actually it consists of a modulator, the physical channel, and a demodulator as shown in Fig. 2.10. In the context of this thesis we introduce two simple channel models of interest, the *additive white gaussian noise channel* (AWGN channel) and the *binary symmetric channel* (BSC). Although

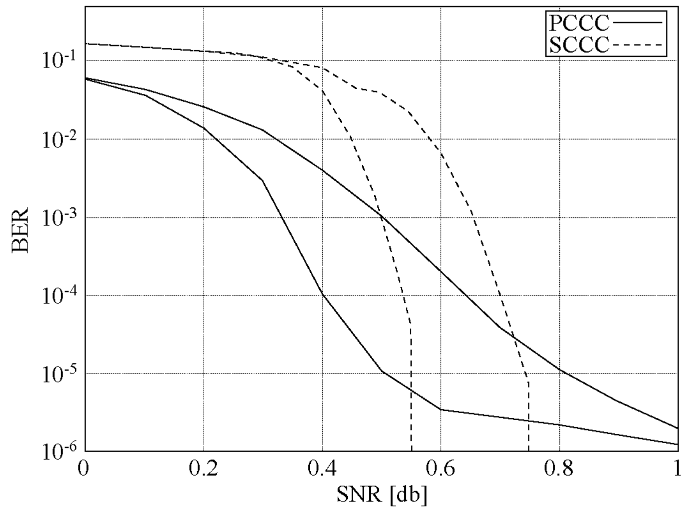


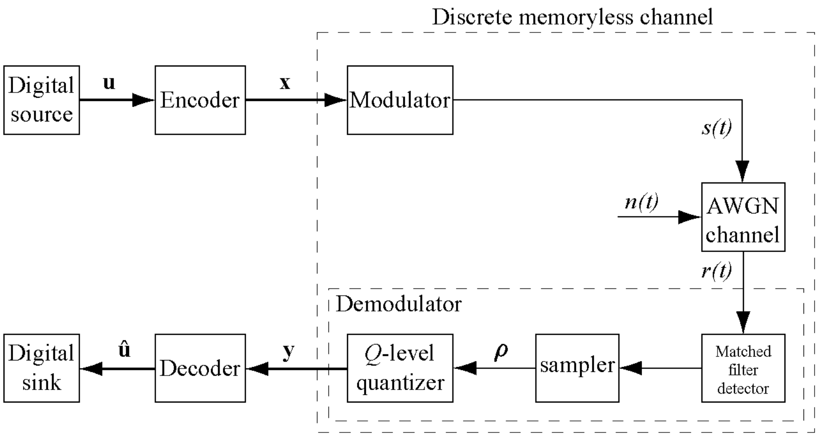
Figure 2.9 *A comparison of serially (SCCC) and parallely (PCCC) concatenated Turbo codes. Parallel concatenated Turbo codes show their characteristic error-floor behaviour [25].*

neither represents the actual situation of most of today's mainstream applications with any accuracy, they are well-suited for our presentation of analog decoders. Complex situations such as a mobile radio channel incorporate, in addition to the AWGN element, also problems like inter-symbol interference (ISI) due to multipath transmission, and fading [26]. We will intentionally omit these complex channel situations from the presentation to keep it simple.

AWGN channel

BPSK modulator

The modulator of Fig. 2.10 adapts the discrete-time, binary sequence to the continuous-time, analog, physical channel. Generally, this is done by so-called pulse-shaping filters. In the binary case, the output of such a modulator is built out of two signals $s_0(t)$ and $s_1(t)$ for an encoded 0 and 1, respectively. In terms of simple detectability, a good choice of these signals for



A more detailed view of Fig. 2.1 for a coded system on an additive white Gaussian noise (AWGN) channel. **Figure 2.10**

a wideband channel is

$$\begin{aligned} s_0(t) &= \sqrt{\frac{2E_c}{T_s}}\sin\left(2\pi f_0 t + \frac{\pi}{2}\right), \quad 0 \le t < T_s \\ s_1(t) &= \sqrt{\frac{2E_c}{T_s}}\sin\left(2\pi f_0 t - \frac{\pi}{2}\right), \quad 0 \le t < T_s \end{aligned} \qquad (2.8)$$

where T_s is the duration of one symbol (or the sampling period), f_0 is a multiple of $1/T_s$, and E_c is the energy of each symbol. This is called *binary-phase-shift-keyed* (BPSK) modulation. The transmitted signal is a sine-wave pulse whose phase is either $+\pi/2$ or $-\pi/2$ depending on the encoder output. Since we have $s_1(t) = -s_0(t)$ we speak of antipodal signaling. The two signal forms allow us to send one channel bit per symbol and time T_s.

Continuous AWGN definition

If we assume that the physical channel is memoryless, i.e., the channel output depends only on the presently transmitted symbol, a common form of disturbance of the channel can be AWGN. If we assume the transmitted signal as $s(t)$, the received signal is

$$r(t) = s(t) + n(t), \quad 0 \le t < T_s \qquad (2.9)$$

where $n(t)$ denotes a sample function of the additive white Gaussian noise process with *power spectral density* (PSD) $\Phi_{nn}(f) = N_0/2$.

BPSK demodulator

The demodulator has to produce an output corresponding to the signal received in each time interval T_s. This output may be a real number or one symbol from a discrete set of preselected symbols, depending on the demodulator design. An optimum demodulator always includes a matched filter or a correlation detector followed by a sampling process. For the BPSK-case with coherent detection, the sampled output of the demodulator is the real number

$$\rho = \int_0^{T_s} r(t) \sqrt{\frac{2}{T_s}} \sin\left(2\pi f_0 t + \frac{\pi}{2}\right) \mathrm{d}t. \tag{2.10}$$

Demodulator for M-ary signals

For the M-ary case, the demodulator decomposes the received signal and the noise into N-dimensional vectors, where $N \leq M$. This means that the signal and the noise are expanded into a series of linearly weighted orthonormal basis functions $\{f_n(t)\}$ as is shown, for example, in [26]. It is assumed that the N basis functions $\{f_n(t)\}$ span the signal space, so that each of the possible transmitted signal waveforms can be represented as a weighted linear combination of $\{f_n(t)\}$.

Discrete AWGN definition

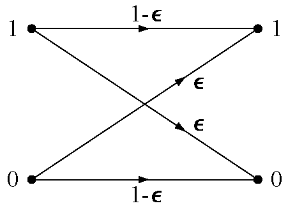
In contrast to the continous-time AWGN channel definition of (2.9), we can define an equivalent discrete version of the AWGN channel whose samples at time instance i are characterized by

$$\tilde{y}_i = \tilde{x}_i + n_i, \tag{2.11}$$

where n_i is a white Gaussian random process, and $\tilde{x}_i$ was obtained by mapping the binary bits of the codeword to an antipodal signal with amplitude $\sqrt{E_c}$: $x_i \mapsto \tilde{x}_i = \pm\sqrt{E_c}$. The Gaussian random process is white in the sense that each sample is independent of any other sample. The probability density function of each sample of such a process is defined by

$$f_n(x) = \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{x^2}{2\sigma_n^2}}, \tag{2.12}$$

where σ_n^2 is the variance of the zero-mean white Gaussian noise $n(t)$. This variance σ_n^2 is related to the the one-sided PSD N_0



The transition diagram of a binary symmetric channel (BSC) with cross-over probability ϵ. **Figure 2.11**

by

$$\sigma_n^2 = \frac{N_0}{2}. \tag{2.13}$$

Analog-input and discrete-input soft decoders

In fact, we can think of the output $\tilde{y}_i$ of the discrete AWGN channel as of an unquantized demodulator output ρ of the continous time AWGN channel. By doing so we can treat both of the channel models equivalently. Hence, the sequence of unquantized demodulator outputs can be passed on directly to an *analog decoder*. Today, a much more common approach to decoding is to quantize the continuous detector output ρ into one of a finite number Q of discrete symbols. In this case, the *digital decoder* has discrete inputs.

Binary Symmetric Channel

DMC and BSC definitions

If we assume a memoryless physical channel enclosed by an M-ary modulator and a Q-ary output demodulator, a *discrete memoryless channel* (DMC) can be modelled. The most important case in the context of this thesis is the *binary symmetric channel* (BSC) with $M = Q = 2$. This configuration can be represented by a channel diagram and is completely described by the *transition probability* ϵ. The probabilities on the branches of Fig. 2.11 represent the conditional probabilities $p(y_i|x_j)$ with $i, j \in \{0, 1\}$.

DMC and hard decoders

A decoder following a DMC with $M = Q$ is generally called hard-decision decoder or hard decoder. Through the quantisation of the channel output which can be relatively coarse, these decoders perform generally worse than those in conjunction with soft-output channels such as the AWGN channel. Note that in the case of a hard decoder, the input can be still a real-valued number but the possible number of symbols or values is limited.

BPSK transition probability expressed by the complementary error function Q

The transition probability ϵ can be calculated from the knowledge of the signals used, the probability distribution of the noise, and the output quantization threshold of the demodulator. In the case of BPSK modulation on an AWGN channel with optimum coherent detection and binary output quantization, the transition probability ϵ is just the bit error probability for a signal sequence with equally likely symbols given by

$$\epsilon = Q\left(\sqrt{\frac{2E_c}{N_0}}\right), \tag{2.14}$$

where $Q(x) \triangleq (1/\sqrt{2\pi}) \int_x^{\infty} e^{-y^2/2} \mathrm{d}y$ is the *Q function* of Gaussian statistics [26].

2.1.8 Types of Errors

Random-error channels

On memoryless channels, the noise affects each transmitted symbol independently. As an example, consider the BSC with a transition diagram as shown in Fig. 2.11. Each transmitted bit has a probability ϵ of being received incorrectly and a probability $1 - \epsilon$ of being received correctly, independently of other transmitted bits. Hence transmission errors occur at random in the received data sequence. Therefore memoryless channels are called *random-error channels*. Typical examples of such channels are the deep-space channel and many satellite channels, as well as most line-of-sight transmission systems [16].

Burst-error channels

On the other hand, on channels with memory, noise is not independent from transmission to transmission. A very simple example is a model with two states: a ‘good’ state in which transmission errors occur infrequently and a ‘bad’ state in which transmission errors are highly probable. Both of them may be modeled according to Fig. 2.11 but with different ϵ’s. The channel is in the good state most of the time, but occasionally shifts to the bad state due to a change in the transmission characteristic of the channel induced, for example, by ‘deep fading’ caused by multipath transmission. As a consequence, transmission errors occur in clusters or bursts because of the high transition probability in the bad state. Such channels are thus called *burst-error channels*. Typical examples include radio channels, where the error bursts are caused by signal fading due to multipath transmission, wire and cable transmission that is subject to impulse switching noise and crosstalk, and magnetic recording, which is subject to tape dropouts due to surface defects and dust particels [16]. For this type of errors, codes with error-decorrelation capabilities, such as e.g. Turbo codes, are especially suitable. The interleaver separates the error burst which then can be corrected on a local scale. Large LDPC codes have comparable capabilities (see also Section 2.1.4).

Compound channels

Finally, combinations of both channels with random errors and burst errors can be found. We call these channels *compound channels*. For each of the above cases, codes especially adapted to their environment may be constructed.

2.2 Analog Viterbi Decoding

2.2.1 Computational Considerations on the VLSI Implementation of Viterbi Decoders

Happy match between algebraic coding theory and digital circuit primitives

The common basic digital circuits with binary memory cells and logic gates are ideally suited for finite-field arithmetic which is the mathematical basis for algebraic coding and decoding theory. This happy match has been exploited for a long time to build efficient VLSI implementations of decoders for such codes. Codes constructed according to algebraic coding theory are best used in applications that have a sufficient margin to the theoretical performance limit. Typical examples of such codes are BCH codes and Reed-Solomon codes [15, 27], which provide strong error protection against low noise levels.

Probabilistic coding techniques are less ideal for a VLSI implementation

In contrast to the algebraic approach, *probabilistic* techniques like the maximum-likelihood (ML) sequence detection using the Viterbi algorithm [23, 24] and decoding techniques based on probability calculus, such as the sum-product algorithm that we use in the context of this thesis, are best suited for applications that have to operate near the theoretical performance limit. However, the match of these techniques to digital VLSI is less than perfect. In fact, the implementation of a high-speed Viterbi decoder takes considerably more chip area than, say, a

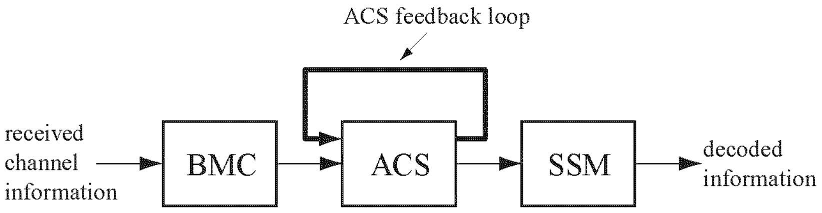


Figure 2.12 *Simplified block diagram of a Viterbi decoder with its main constituents branch-metric computation (BMC) unit, add-compare-select (ACS) unit and storage-survivor-memory (SSM).*

BCH decoder achieving the same bit rate. This is mainly due to the binary number system of today's computers which creates a considerable overhead in the implementation of floating-point arithmetic units and thus needs a significant amount of chip area. Approximations to the floating-point number representation using equivalent fixed-point units can be made, but the binary system still introduces a large amount of redundancy in terms of the data representation at the cost of higher power consumption and larger chip area.

Partly analog Viterbi decoders are a solution to the high-speed and low-power dilemma

Because of the ever growing transmission-speed and power-efficiency requirements for both fixed and mobile communication devices, some researchers have recently become interested in analog decoding techniques. Many analog or hybrid implementations of the Viterbi decoding algorithm for trellis codes have been proposed [28–40]. All of them simply replace the most critical parts of the Viterbi decoder of Fig. 2.12, generally the add-compare select (ACS) unit and its feed-back loop, by analog circuit implementations. But still, the whole decoder remains a sequential machine that performs one trellis computation after the other. Therefore, the time needed for one trellis computation limits the speed of the overall system.

2.2.2 Circuit Implementation of Analog and Mixed-Signal Viterbi Decoders

Distinction between voltage-mode and current-mode implementations

Regarding the different circuit implementations, we can distinguish between *voltage-mode* and *current-mode* implementations. These two terms are generally used to classify whether the information-carrying signals are mainly of current nature or of voltage nature. This distinction is in practice not so obvious. For example the input information of a simple current mirror can be seen as the current passing the diode-connected input transistor as well as the gate-source voltage of the same transistor driving the second transistor. A more appropriate means to identify voltage- and current-mode circuits is the criteria whether currents are driving mostly high-impedance nodes (voltage-mode) or low-impedance nodes (current-mode) at the input of a building block [41, 42].

Distinction between continuous-time and discrete-time circuits

A second classification criteria of the different analog and mixed-signal implementations of Viterbi decoders are their mode of operation in time. We can mainly distinguish between continuous time circuits and switched or discrete time circuits. For the overall decoding function to be implemented it does not matter at all which of the two operation modes is used. The main classes of the discrete-time circuits are switched-capacitor (SC) circuits [43, 44] and switched-current (SI) circuits [45]. In that sense this classification criterion is orthogonal to the distinction between voltage-mode and current-mode circuits.

Discussion of analog and mixed-signal Viterbi decoders

From the point of view of the impedance criteria, the implementations described in [28–32] are clearly continuous-time voltage-mode circuits using SC cells to store the analog state-metrics and feed them back to the input of the ACS unit. They all use opamp-based adders and comparators to implement the ACS unit. With the evolution in time, they have consecutively reached higher operational speeds starting from some 10 Mbit/s to over 200 Mbit/s. The solution described in [38] uses SC techniques for both, metric calculation and storage and is therefore also a pure voltage-mode circuit implementation. A processing speed of 500 kbit/s at a power consumption of less than 8 mW has been demonstrated by this technique. The approach in [33, 34] is one of mixed current- and voltage-mode. The metric calculations are accomplished partly in the current domain [33], whereas the comparison and the storage of the state metrics (SC sample-and-hold (S/H) circuits) are in the voltage-mode. The realization of the ACS function is implemented by a new diode network using threshold-programmable diode devices. Hence this part of the Viterbi decoder is a continuous-time circuit. Very high operational speed of over 300 Mbit/s has been reported in [32]. Pure current mode implementations are reported by Demosthenous and Taylor in [35–37]. The addition and comparison of two current signals are simple tasks in continuous-time current-mode circuits. The storage of current signals can equally well be accomplished by SI memory cells compared to SC techniques. Decoding speeds of $> 100\,\mathrm{Mbit/s}$ are expected using the fully-current-mode approach. But although the calculations in the current domain are very promising regarding their simplicity, the fully-current-mode approach is still in its feasability-study state, whereas voltage-mode circuits are already considered for use in practical applications.

2.3 Network Decoding

In order to meet the ever growing demand in decoding speed, several methodologies for creating network decoders have been proposed so far. Most of them are based on sequential machines or are some other kind of discrete-time signal processors. But there exist also continuous-time processors which, as we will see later on in this thesis, come most closely to our approach of probability-based analog, continuous-time networks. In this section we revise the most important examples of the network decoding approach.

2.3.1 Non-Algorithmic Diode Decoding

Shortest-path problem

Minty [46] described an elegant solution method to the shortest path problem: assume a net of flexible, variable length strings. In that scale model of the network you wish to find the shortest path. If you now pick up the source node and the destination node and pull the nodes apart until the net is tightened, you directly find the solution along the thightened path as in Fig. 2.13. Note that Minty's solution applies to non-directed graph models only. It is thus not directly applicable to trellis decoding. The decoding of a trellis code is equivalent to the shortest path problem in a *directed* graph [24, 46].

Diode decoder

An analog circuit solution to the shortest-path problem in directed graph models has been found independently by Davis and much later by Loeliger [48, 49]. It consists of an analog network using series-connected diodes. According to the individual path section lengths as in Fig. 2.14 a number of series-connected diodes are placed. The current I will then flow along

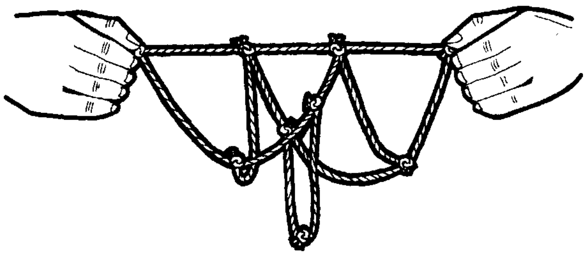


Figure 2.13 *The shortest path problem solved using a rope net [47] (©1999 IEEE).*

the path with the least number of series-connected, forward-biased diodes. Note however that the sum of the diode threshold voltages fundamentally limits practical applications. Very high supply voltages will be needed for larger diode networks, which makes this elegant solution useless for VLSI implementations.

Diode decoder with variable threshold voltage

A similar method to Davis' diode decoder has recently been presented by Bu *et al.* [47]. The basic network elements are variable threshold-voltage diode-devices connected to an analog network. For each branch of the network, exactly one device is introduced for each variable parameter. Instead of assembling more or fewer diodes in series for one branch, the threshold voltage can be tuned continuously within a certain range. Fundamentally, this solution suffers from the same limitations in terms of supply voltage as the solution of [49]. The idea of variable turn-on voltages however offers a possibility to build soft-input decoders. This idea can also be found in the analog Viterbi decoder implementation of Shakiba *et al.* [33, 34].

2.3.2 Neural Network and Fuzzy Logic Decoding

Neural networks and decoding

Classifying signals is a task to which neural networks are often applied. This approach can be found in various domains of signal and information processing as for example in optical character recognition (OCR) [50–53], handwriting recognition [54–56], voice recogniton [57, 58] and general pattern recognition and classification [59, 60]. The mapping of

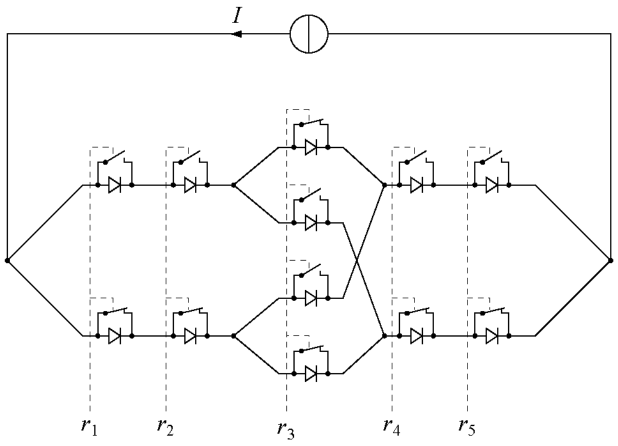


Figure 2.14 *Hard-decision decoder for a very simple trellis code. The received bits r_i control directly the switches as indicated by the dashed lines.*

the high-dimensional received input-signal space to the lower-dimensional space of valid codewords can also be seen as a 'simple' classification problem. Hence, it is not astonishing that several attempts to solve the decoding problem with neural networks have been proposed [39, 40, 61–63]. Wang and Wicker [39] and Verdier *et al.* [40] even proposed an analog implementation of a neural network for decoding purposes.

Fuzzy sets for classifiers

Related to the neural network decoding approach is the fuzzy logic decoding idea. In 1965, Zadeh proposed the *fuzzy set* concept [64] as a means of handling unreliable information. Based on these mathematical foundations, Wu *et al.* [65] introduced a hybrid fuzzy-neural-network decoder. The proposed fuzzy neural classification network basically consists of a three-layer neural network. To enhance the associative capability of the network, fuzzy membership functions were defined for each hidden node.

Analog Network Decoding

With the advent of computationally demanding iterative decoding techniques, several researchers started to look for non-traditional representation and implementation methods of algorithmic decoding networks. Wiberg *et al.* were the first to speculate on analog implementations of the so-called sum-product algorithm [66]. Inspired by the analog diode network decoders presented in Section 2.3.1 and Wiberg's work on iterative graph decoding [67], a new analog implementation approach for the *maximum-a posteriori* (MAP) decoding has recently been presented independently by Hagenauer *et al.* [68–70] and by us [71–79]. This work is also the basis for the present thesis, which deals mainly with design aspects and implementation issues of analog VLSI iterative decoders based on the sum-product algorithm. For a long time, Hagenauer *et al.* did not consider the actual transistor implementation of their non-linear networks at all. Only very recently they became interested in chip implementations of their networks [70]. They finally came up with the same generic transistor modules as we had before [71], with only a slight difference in the way of connecting individual circuit blocks.

2.4 Bio-Inspired Networks

Adaptive analog systems are far more efficient

Mead championed a completely new analog VLSI design style [11]. The *neuromorphic* approach for analog signal processing circuits mimics in many ways the function of the biological nervous system. This design style is characterized by exploiting, rather than fighting, the fundamental nonlinearities of transistor physics. Precision is achieved on the system level despite low-precision components. Mead suggests in [80] that *"adaptive analog systems are 100 times more efficient in their use of silicon, and they use 10'000 times less power than comparable digital systems."* Many practical examples such as building blocks for neural networks [81–86], artificial cochleas [87–91], silicon retinas [92–94], and motion detectors [12,95] have been successfully fabricated. They show indeed astonishing system-level performance compared to traditional analog and digital systems.

neuromorphic vs. bio-inspired

In the context of decoding of digital codes, 'neuromorphic circuits' does not seem to be the correct term. In fact, we do not want to copy the function of the nervous system. The aim is rather to have an electronic system that has advantages comparable to its biological counterpart. Thus we speak of *bio-*

inspired circuits instead of neuromorphic circuits to make this difference clear. Key-features of bio-inspired circuits are low-power and high-speed operation on system level. They benefit from collective computation, its precision is gained on the system level despite the use of low-precision components, and they are small in size. A further key-feature of the bio-inspired design approach is that one exploits, rather than fights against, the inherent nonlinearities of the basic semiconductor devices. This means that in contrast to a conventional analog circuit-design approach, the bio-inspired approach uses the devices 'as is' and does not try to implement 'linear' transfer characteristics out of linearized non-linearities by relying on the small-signal concept.

Chapter 3

The Probability-Propagation Algorithm

Problem Statement 3.1

In this section, we will review the basic decision theory and its application to the decoding problem. Several standard decoding criteria such as the maximum-likelihood (ML) rule and the maximum-*a posteriori* (MAP) rule are discussed for both bit- and block-wise decoding.

Basic Decision Theory 3.1.1

Decoding is a decision process

Basically, decoding is a decision-making process. Based on the observed data vector, the decoder tries to figure out which actual information bit or information vector has been generated by the information source.

Recalling our data transmission system of Fig. 2.1, we assume in the following that the data source is an independently and identically distributed (i.i.d.) source, so that we can write

$$P_{\mathbf{U}}(\mathbf{u}) = \prod_{i=0}^{k-1} P_U(u_i) \tag{3.1}$$

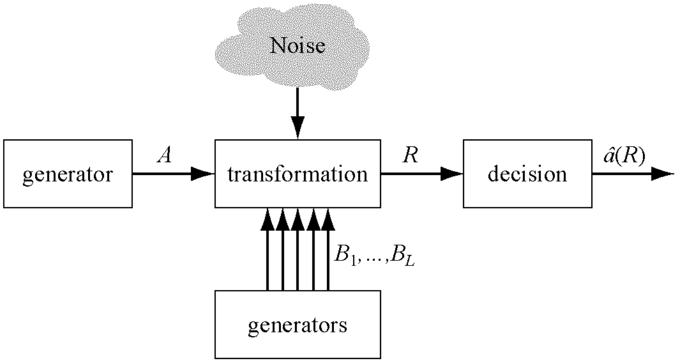


Figure 3.1 *A general model of the decision problem. Using observation R one wants to decide on A without having direct access to it.*

with $P_U(u_i)$ having the same distribution for all i.[1] The data transmission is assumed to be over a time-invariant, memoryless and feedback-less channel. Under these conditions we are allowed to write the probability of the received values conditioned on the sent values as a product of conditional probabilities:

$$P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=0}^{k-1} P_{Y|X}(y_i|x_i). \tag{3.2}$$

Decision model

A general model of a decision[2] problem is shown in Fig. 3.1, with A being a random variable or random vector with finite alphabet $\mathcal{A}$, $B_1 \in \mathcal{B}_1, \ldots, B_L \in \mathcal{B}_L$ some additional random variables we are not directly interested in, and observation R a continuous or discrete random variable or random vector. For reasons of convenience, in the further derivations, we assume R to be a finite discrete random variable with alphabet $\mathcal{R}$.

Quality measure for decisions

A good quality measure of a decision is the probability of making a correct decision. This probability $P_{\text{correct}} \triangleq P[\hat{A}(R) = A]$

[1]The subscript U of the probability function P denotes the investigated random variable, whereas the argument u is a concrete realization of this random variable U. To have a more condensed writing, one often writes simply $P(u)$. A more detailed description of the random variable concept and stochastic processes can be found in [96].

[2]It is important to note the difference between *decision* and *estimation*. While in an estimation problem a value close (according to a certain distance measure) to the actual occurred value is headed for, in a decision

can be calculated in a general way as

$$\begin{aligned} P_{\text{correct}} &= \sum_{r \in \mathcal{R}} P(\hat{A}(r) = A | R = r) P_R(r) \\ &= \sum_{r \in \mathcal{R}} P_{A|R}(\hat{a}(r)|r) P_R(r). \end{aligned} \tag{3.3}$$

The aim of a decision-making process is then to maximize this probability. The two most popular rules, the MAP and the ML decision rules, are discussed in the following subsections.

MAP Decision Rule 3.1.2

By definition, probabilities are non-negative numbers in the range $[0..1]$, and the decision $\hat{a}(r)$ of A can be defined freely for each outcome r of R. Hence P_{correct} is maximized by choosing $\hat{a}(r)$ for each r such that $P_{A|R}(\hat{a}|r)$ is maximized. This follows directly from (3.3), and thus we define

$$\hat{a}_{\text{MAP}}(r) \triangleq \arg\max_{a \in \mathcal{A}} P_{A|R}(a|r) \tag{3.4}$$

$$= \arg\max_{a \in \mathcal{A}} \sum_{\mathbf{b} \in \mathcal{B}_1 \times \ldots \times \mathcal{B}_L} P_{A\mathbf{B}|R}(a, \mathbf{b}|r), \tag{3.5}$$

where the function $\arg\max_x f(x)$ returns the particular x that maximizes $f(x)$.[3] This decision rule is called *maximum-a posteriori* (MAP) decision rule to express that it maximizes the *a posteriori* probability $P(a|r)$ for r.

problem one looks for the exact value. Therefore, an estimation can be more or less good, whereas the decision is either right or wrong.

[3]In general, arg max is a function that returns a set of all maxima. Of course, this set has more than one element if the maximum is achieved for different values; but here we assume that arg max returns only one value.

3.1.3 ML Decision Rule

In the previous subsection we have seen that the maximization of P_{correct} is independent of $P_R(r)$, and so we can write (3.4) as

$$\begin{aligned}\hat{a}(r) &= \arg\max_{a\in\mathcal{A}} P_{A|R}(a|r) = \arg\max_{a\in\mathcal{A}} \frac{P_{AR}(a,r)}{P_R(r)}\\ &= \arg\max_{a\in\mathcal{A}} P_{AR}(a,r) \qquad (3.6)\\ &= \arg\max_{a\in\mathcal{A}} P_{R|A}(r|a)P_A(a).\end{aligned}$$

However, $P_A(a)$ is often unknown, so one assumes that A is uniformely distributed, i.e., $P_A(a)$ is constant for all $a \in \mathcal{A}$. With this assumption we can postulate a new decision rule:

$$\hat{a}_{\text{ML}}(r) \triangleq \arg\max_{a\in\mathcal{A}} P_{R|A}(r|a). \qquad (3.7)$$

This decision rule is known as the *maximum-likelihood* (ML) decision rule. We immediately verify the equivalence between the ML-case on the left-hand side of (3.8) and the MAP-case on the right-hand side of the equation of (3.8), when A is uniformely distributed:

$$\arg\max_{a\in\mathcal{A}} P_{R|A}(r|a) = \arg\max_{a\in\mathcal{A}} P_{R|A}(r|a)P_A(a), \qquad (3.8)$$

again using the fact that the maximization is independent of $P_R(r)$.

Equivalence and optimality of MAP and ML

We have observed that the MAP decision rule and the ML decision rule are equivalent if the condition that A is uniformely distributed is met. In the general case, however, the ML decision does not necessarily maximize P_{correct}, since the distribution of A can rarely be guaranteed to be uniform. For practical applications, where $P_A(a)$ is often unknown, the ML decision rule is used regularly despite its non-optimum character. This is not a limiting drawback as the aim of possible source encoding is to modify the probability distribution of A to have as uniform a distribution as possible.

3.1.4 Decoding Rules

Block- and bit-wise decoding

In order to match the terminology of Fig. 2.1 and Fig. 3.1, we have first to find correspondences. Two different cases have to be distinguished in the context of decoding: block- and bit-wise decoding.

Block-Wise Decoding

For block-wise decoding, the translation is as follows:

$A \triangleq \mathbf{U}$	unknown information vector
$R \triangleq \mathbf{Y}$	received data vector
$\hat{A} \triangleq \hat{\mathbf{U}}$	reconstructed information vector

MAP block decoding

The random variables B_i are not used in this case. Starting with these definitions and using a simplified notation, the MAP decison rule is then given by

$$\hat{\mathbf{u}}_{\mathrm{MAP}}(\mathbf{y}) = \arg\max_{\mathbf{u}} P(\mathbf{u}|\mathbf{y}) \tag{3.9}$$

$$= \arg\max_{\mathbf{u}} P(\mathbf{u},\mathbf{y}). \tag{3.10}$$

MAP block decoding maximizes $P_{\mathrm{correct}} = P[\hat{\mathbf{U}} = \mathbf{U}]$. This is equivalent to the minimization of the block error probability $P_{\mathrm{Blockerr}} = P[\hat{\mathbf{U}} \neq \mathbf{U}]$. This kind of block decoding is also known as *sequence estimation*, although the term estimation is somewhat misleading in the context of decoding. More appropriate is the term *sequence detection*.

ML block decoding

Equivalently, the ML decision rule is defined as

$$\hat{\mathbf{u}}_{\mathrm{ML}}(\mathbf{y}) = \arg\max_{\mathbf{u}} P(\mathbf{y}|\mathbf{u}) \tag{3.11}$$

if $P_{\mathbf{U}}(\mathbf{u})$ is assumed to be uniformly distributed.

An example for such an ML decoder is the well-known Viterbi sequence detector used in harddisc drive applications (see e.g. [29]). Note that a MAP-version of the Viterbi algorithm is possible as well, as we will see in Section 5.4.1.

Decoding type	Decision rule
(block-) MAP	$\hat{\mathbf{u}}(\mathbf{y}) = \arg\max_{\mathbf{u}} P(\mathbf{u}\vert\mathbf{y})$
(block-) ML	$\hat{\mathbf{u}}(\mathbf{y}) = \arg\max_{\mathbf{u}} P(\mathbf{y}\vert\mathbf{u})$
symbol-MAP	$\hat{u}_k(\mathbf{y}) = \arg\max_{u_k} P(u_k\vert\mathbf{y}) \quad \forall k$
symbol-ML	$\hat{u}_k(\mathbf{y}) = \arg\max_{u_k} P(\mathbf{y}\vert u_k) \quad \forall k$

Table 3.1 *Decision rules for the main decoding types.*

Bit-Wise Decoding

Bit-wise decoding uses a somewhat different assignment of terminology:

$A \triangleq U_k$	unknown information bit
$R \triangleq \mathbf{Y}$	received data vector
$\hat{A} \triangleq \hat{U}_k$	reconstructed information bit
$B_i \triangleq U_i$	$\forall i \neq k$

MAP bit-wise decoding

for each $k = 0, \ldots, K-1$. With these definitions, the MAP rule for bit-wise decoding is given by

$$\hat{u}_k(\mathbf{y}) = \arg\max_{u_k} P(u_k|\mathbf{y}) \tag{3.12}$$

$$= \arg\max_{u_k} \sum_{\tilde{\mathbf{u}} \in \mathcal{U}^k : \tilde{u}_k = u_k} P(\tilde{\mathbf{u}}|\mathbf{y}) \tag{3.13}$$

$$= \arg\max_{u_k} \sum_{\tilde{\mathbf{u}} \in \mathcal{U}^k : \tilde{u}_k = u_k} P(\tilde{\mathbf{u}}, \mathbf{y}) \tag{3.14}$$

for all $k = 0, \ldots, K-1$. The MAP decision rule for bit-wise decoding maximizes $P_{\text{correct}} = P[\hat{U}_k = U_k]$. At the same time it minimizes the symbol error probability $P_{\text{Symbolerr}} = P[\hat{U}_k \neq U_k]$. Likewise we define the bit-wise ML decoding rule. Table 3.1 summarizes all four cases.

3.2 Factor Graphs

After having laid out the decision-theoretic background of decoding, we will briefly have a look at an important graphical model which is used afterwards to describe both code and decoding system.

3.2.1 Definition of Factor Graphs

Factor graphs represent the factorization of a global function

According to the definition in [97], a *factor graph* is a *bipartite graph* that expresses how a "global" function of many variables factors into a product of "local" functions. This generalization of Tanner graphs [98] by adding hidden state variables has been introduced in [66] and in Wiberg's doctoral thesis [67]. Factor graphs subsume many other graphical models such as *Markov random fields*, and *Bayesian networks*. To see the relationship between these graphical models and the factor graphs, the reader may consult [97] and the references therein.

A very simple factor-graph example

To introduce the factor-graph concept, let us start with a very simple example [97]. Take a real-valued function $g(x_1,x_2,x_3,x_4,x_5)$ of five variables that can be written as the product

$$\begin{aligned} &g(x_1,x_2,x_3,x_4,x_5) \\ &\quad = f_{\mathrm{A}}(x_1)f_{\mathrm{B}}(x_2)f_{\mathrm{C}}(x_1,x_2,x_3)f_{\mathrm{D}}(x_3,x_4)f_{\mathrm{E}}(x_3,x_5) \qquad (3.15)\end{aligned}$$

of five functions f_{A}, f_{B}, f_{C}, f_{D}, and f_{E}.

Graphical ingredients of factor graph

The factor graph corresponding to (3.15) is shown in Fig. 3.2. We can identify a circle for each variable x_i representing *variable nodes* and a filled rectangle for each factor f representing *function nodes*, respectively. The variable nodes for x_i are connected to the function node for f by means of edges if and only if x_i is an argument of f. A third type of node, not actually present in Fig. 3.2, is the class of auxiliary variables or *state nodes* which are drawn as double circles (see e.g. Fig. 3.5). They form a subset of the variable nodes and are often used to shape the factor graph. They are not observable from the outside of the system.

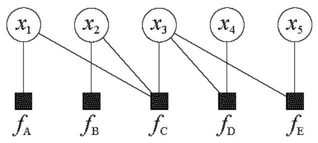


Figure 3.2 *A factor graph expressing that a global function $g(x_1,x_2,x_3,x_4,x_5)$ factors as the product of the local functions $f_A(x_1)$, $f_B(x_2)$, $f_C(x_1,x_2,x_3)$, $f_D(x_3,x_4)$, and $f_E(x_3,x_5)$.*

Forney's Normal Graphs

Definition of Forney's normal graphs

The original definition of factor graphs allows only the connection of nodes of different type [67]. As we will see in Chapter 5, it might be interesting to allow direct connections between function nodes. By doing this, the factor graphs and the sum-product algorithm that runs on these factor graphs can directly be transformed into an analog transistor-level description. In [99, 100], Forney takes the original definition of factor graphs and applies some modifications to allow direct connections of function nodes. He calls these new graphs *normal graphs*. He uses the convention that the degree of a variable node has to be less or equal to 2, i.e., variable nodes have at most two connections. By doing this, he can separate external variables (I/O nodes observeable from outside) very easily from internal variables or state variables. State-variable nodes have degree two whereas external variable nodes have only degree one. As we will see in the introduction to the sum-product algorithm, a variable node with two connecting edges has no processing task to fulfill since it passes the messages right away to the next function node. Since the internal variables have no explicit function anymore, they are not needed anymore and can be safely omitted. Thus internal variables, i.e., what we called states in the original factor-graph definiton, appear as the edges only between function nodes. Forney even modifies the original graphical representation by introducing so-called *stub nodes* for the I/O-nodes. In the context of this thesis, we do not apply the whole notational rigorosity and just use the node-splitting property in factor graphs for our implementation purposes.

Node splitting

The transformation of an original factor graph into the new Forney-style factor graph can be carried out easily by *node splitting*. Variable nodes with a degree higher than 2 can be

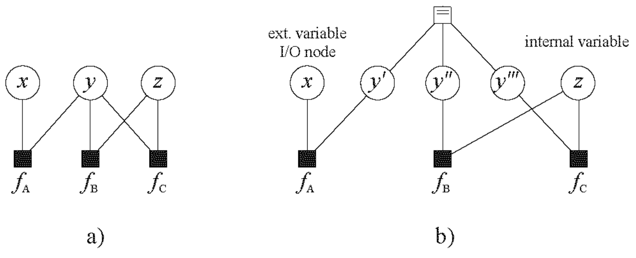


Figure 3.3 *a) original factor graph, b) node-splitting procedure to keep the degree of variable nodes below or equal to 2.*

replaced by the special function 'equal' and variable nodes of degree 2 as in Fig. 3.3. The 'equal' function node assures that y', y'' and y''' are kept at the same value. For a formal definition of the 'equal' function see Section 3.2.2.

Function Summaries

Marginal and function summaries

Recalling the simple example of Fig. 3.2, we suppose that we are interested in determining the influence on one specific variable by the rest of the global function. To find the solution to this problem, we need to compute a function *summary* or *marginal*. According to the slightly unconventional summation notation of [97], the 'summary for x_2' of a function h of three variables x_1, x_2, and x_3 is denoted by a 'not-sum' of the form

$$\sum_{\sim\{x_2\}} h(x_1,x_2,x_3) \triangleq \sum_{x_1}\sum_{x_3} h(x_1,x_2,x_3). \tag{3.16}$$

Therefore by using the notation of (3.16) we have the general ith marginal function associated with $g(x_1,\ldots,x_n)$ denoted by

$$g_i(x_i) \triangleq \sum_{\sim\{x_i\}} g(x_1,\ldots,x_n), \tag{3.17}$$

which is the summary for x_i of g. To get the marginal function, we thus sum over all possible configurations of g other than x_i. The need for marginal functions will be clearer in the following subsection, where a specific decoding example will be described.

3.2.2 Examples of Factor Graphs

Factor graphs appear in various fields

The application range of factor graphs is very broad. Factor graphs can be applied to both set-theoretic and probabilistic modeling. Examples for set-theoretic modeling include code description and state-space models, whereas typical examples for probabilistic modeling include Markov chains, hidden Markov models. Furthermore, factor graphs can even be used to describe fast transforms, which has been demonstrated by Kschischang *et al.* in [97] for the case of the Fast Fourier Transform (FFT). We will restrict ourselves in this subsection to the presentation of very simple, but important examples in the context of coding and decoding.

Linear Block Codes

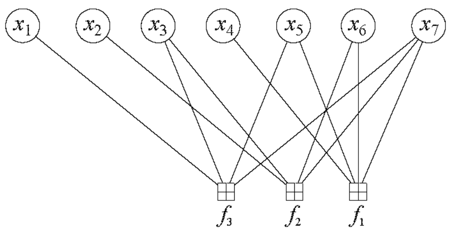
Iverson's notation

Let us begin with the description of linear codes by factor graphs. For every code, one or even more than one factor graph representation can be found. In the case of linear codes, it is convenient to start with the parity-check matrix as for example in (2.7). The Hamming code C is defined over GF(2) and each binary 7-tuple $\mathbf{x} \triangleq (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ has to satisfy $\mathbf{H}\mathbf{x}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$. Each row of the parity-check matrix gives us an equation that has to be satisfied by $\mathbf{x}$, and simultaneously, *all* the equations have to be satisfied to form a valid codeword. Now, "Iverson's convention" [101] is very useful to assist behavioural modeling. If P is a predicate, i.e., a Boolean proposition, then $[P]$ is the binary function indicating truth of P:

$$[P] \triangleq \begin{cases} 1 & \text{if } P; \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

Code membership function

For each equation of the code C, a binary indicator function can be defined which describes the satisfaction of the check equation. The product of these functions indicates then the membership in the code. Therefore, the three rows of the considered code C can be used to derive the code membership indicator



The factor graph of the binary Hamming code defined by the parity-check matrix (2.7). **Figure 3.4**

function $g(x_1,x_2,\ldots,x_7)$ as the product of three local functions:

$$\begin{aligned} g(x_1,x_2,\ldots,x_7) &= [(x_1,x_2,\ldots,x_7) \in C] \qquad (3.19) \\ &= [(x_4 \oplus x_5 \oplus x_6 \oplus x_7) = 0] \\ &\quad \cdot [(x_2 \oplus x_3 \oplus x_6 \oplus x_7) = 0] \\ &\quad \cdot [(x_1 \oplus x_3 \oplus x_5 \oplus x_7) = 0], \qquad (3.20) \end{aligned}$$

where $\oplus$ denotes the sum operator (or XOR function) in GF(2). Using (3.20) we can draw the corresponding factor graph as shown in Fig. 3.4. Therein, instead of the black square used in general factor graphs, we have inserted the special symbol $\boxplus$ to indicate the actual parity-check function. We will freely use symbols for function nodes, depending on the type of the local function. However, variable nodes will always be drawn as circles; sometimes though double circles will be used for auxiliary variables (states) as in Fig. 3.5.

Convolutional Codes

Rectilinear factor graphs for convolutional codes

We can draw a factor graph for a convolutional code in the same style as in Fig. 3.4. The membership functions are somewhat different this time: as we have seen in Fig. 2.5, a valid codeword of a convolutional code can be read off as a path in the corresponding trellis diagram. At each time step, a new trellis section is added. Thus not suprisingly, the factor graph of a convolutional code is rectilinear, i.e., a straight line, as is shown in Fig. 3.5. Each function node is characterized by a binary indicator function which can be drawn as a trellis diagram where edges exist only for valid state transitions. For a non-terminated

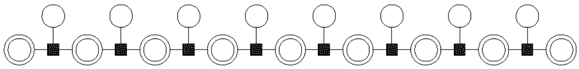


Figure 3.5 *The factor graph representation of a terminated trellis code.*

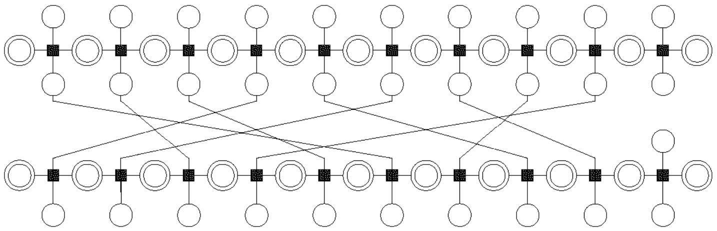


Figure 3.6 *The factor graph of a very short turbo code.*

convolutional code, the length of the factor graph is, like the trellis diagram, possibly infinite. The factor graph of a tail-biting code is formed by identifying the first and the last state node of the factor graph of an ordinary convolutional code and thereby forming a ring structure.

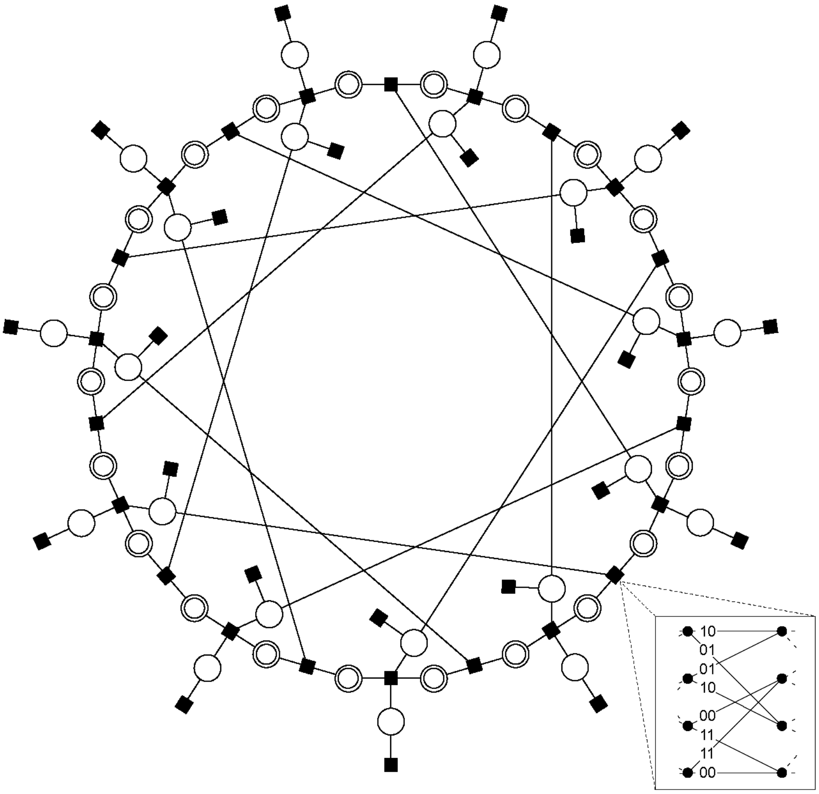
Turbo Codes

Interleaver of Turbo codes is visible in factor graph

Taking again the turbo encoder of Fig. 2.8, we can easily draw the factor graph using the knowledge of the form of convolutional codes. The concatenation of the two constituent codes C_1 and C_2 of a turbo code as shown in Fig. 2.8 is realized by the interleaver. This permutation π can be identified as the connection pattern in the middle row of Fig. 3.6. Obviously, to get a good code according to Shannon [1], the codes have to be much longer than those of Fig. 3.6.

Turbo-style codes have a similar connection pattern

The connection pattern is the main feature of the factor graph of a turbo code. Apart from the original turbo codes, a whole class of similar codes has been identified. The so-called turbo-style codes also have a highly connected pattern as their key features. As a beautiful example, we redraw the factor graph of the [22,11,7] subcode of a binary Golay code in Fig. 3.7 (see [26, 67]). The membership indicator function of the function nodes on the main ring structure is again characterized by a trellis diagram as shown in the inset of Fig. 3.7. Note that the



The factor graph of a turbo-style code. **Figure 3.7**

aesthetically very pleasing appearance of this factor graph is mainly due to the introduction of additional state variable-nodes which let us factor the overall function in the desired way.

Probability distributions and decoding

Probability distributions are another important class of functions that can be represented by factor graphs. Since conditional and unconditional independence of random variables is expressed in terms of a factorization of their joint probability mass function or joint probability density function, factor graphs for probability distributions appear in many different

Factorizations of joint probability mass functions appear in many situations

situations. As we saw in Section 3.1.1 discussing the basic decision theory, decoding is one of the many applications where exactly this kind of functions arises.

MAP decoding is a situation that is often modeled

A situation that is often modeled in coding theory is as follows: select a codeword $(x_1,\ldots,x_n)$ with uniform probability from a code C of length n which is then transmitted over a discrete memoryless channel without feedback that has a corresponding output $(y_1,\ldots,y_n)$. Since we assumed a memoryless channel, by definition the conditional probability mass function or conditional probability density function evaluated at a particular channel output is given by the product form

$$p(y_1,\ldots,y_n|x_1,\ldots,x_n) = \prod_{i=1}^{n} p_{Y|X}(y_i|x_i). \tag{3.21}$$

The *a priori* probability of selecting a particular codeword is constant. Thus the *a priori* joint probability mass function of the codeword is proportional to the code set membership function. Using (3.21) we can therefore write the joint probability mass function of $\{x_1,\ldots,x_n,y_1,\ldots,y_n\}$ as

$$\begin{aligned} &p(x_1,\ldots,x_n,y_1,\ldots,y_n) \\ &\qquad = \gamma \cdot [(x_1,\ldots,x_n) \in C] \cdot \prod_{i=1}^{n} p_{Y|X}(y_i|x_i), \end{aligned} \tag{3.22}$$

where γ is a constant, positive scale factor. The code membership indicator function $[(x_1,\ldots,x_n) \in C]$ itself may factor into a product of local indicator functions as we have seen in (3.20). Reusing the Hamming code example, we can write the joint probability mass function according to

$$\begin{aligned} &p(x_1,\ldots,x_7,y_1,\ldots,y_7) \\ &= \gamma \cdot [(x_4 \oplus x_5 \oplus x_6 \oplus x_7) = 0] \cdot [(x_2 \oplus x_3 \oplus x_6 \oplus x_7) = 0] \\ &\quad \cdot [(x_1 \oplus x_3 \oplus x_5 \oplus x_7) = 0] \cdot \prod_{i=1}^{7} p_{Y|X}(y_i|x_i), \end{aligned} \tag{3.23}$$

which can be easily described as the factor graph shown in Fig. 3.8.

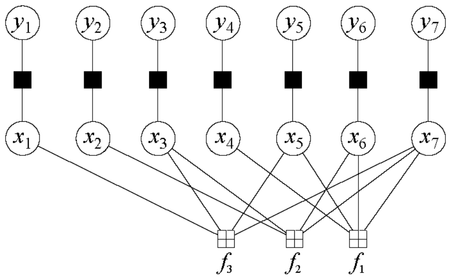


Figure 3.8 *Factor graph for the joint probability density function of channel input and output for the Hamming code of Fig. 3.4.*

Getting a factor graph of the decoder by augmenting the factor graph of the code

Compared to the factor graph of the code (see Fig. 3.4), the factor graph of the joint probability mass function of codeword symbols and channel output symbols is obtained simply by augmenting the factor graph of the code. This is done by adding a channel function and the corresponding channel-input variable for each I/O variable node of the factor graph as demonstrated in Fig. 3.8. This is a very interesting observation, since we realize that the factor graph of the code and the factor graph of a possible decoding scheme are tightly related. In general, one possible decoder can directly be derived knowing the code structure.

Logic functions

Logic circuit diagrams are very popular factor graphs

Surprisingly, even if you did not hear of factor graphs explicitely before, you may be already quite familiar with certain types of them. The local functions of Fig. 3.9 are drawn as logic gates and remind us of the definition of the corresponding binary indicator function. For example the OR gate with inputs u_1 and u_2 and output x_1 represent the binary indicator function $f(u_1,u_2,x_1) = [x_1 = (u_1 \mathrm{OR}\, u_2)]$. The global function of the factor graph representation of the logic circuit of Fig. 3.9 can be written as

$$\begin{aligned} &g(u_1,u_2,u_3,u_4,x_1,x_2,y_1) \\ &= [x_1 = (u_1 \mathrm{OR}\, u_2)][x_2 = (u_3 \mathrm{XOR}\, u_4)][y_1 = (x_1 \mathrm{NAND}\, x_2)]. \end{aligned} \tag{3.24}$$

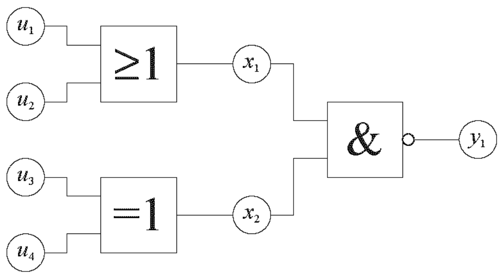


Figure 3.9 *A logic circuit is also a factor graph.*

The global function g takes the value of 1 if and only if all its arguments form a valid configuration which is consistent with the logic circuit of Fig. 3.9. In general, every block diagram may be viewed and drawn as a factor graph.

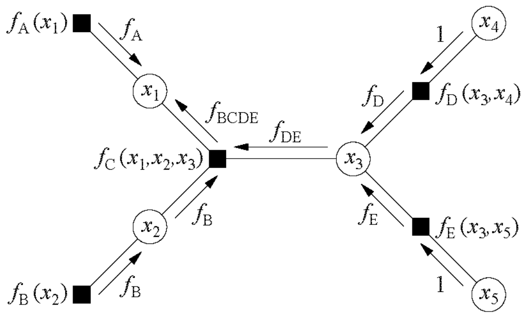
3.3 The Sum-Product Algorithm

Generic sum-product algortihm calculates function summaries in a distributed manner

The sum-product algorithm is a generic algorithm that operates on a factor graph via a sequence of local computations at every factor-graph node [97]. The computation rules consist only of multiplications and additions, hence the name 'sum-product algorithm'. The local results are passed as *messages* along the edges of the factor graph. The algorithm can be used to compute the exact function summary, as defined by (3.17), in a factor graph that forms a tree, i.e., has no loops. But the sum-product algorithm can also be applied to factor graphs with cycles where it results in an iterative algorithm without a natural termination. This makes the function summary non-exact. But decoding of turbo codes or low-density parity-check codes are some of the most exciting applications that reflect precisely this situation with a factor graph having cycles. And with some precautions, the algorithm performs very well.

Exercise example of calculations...

To formally start the mathematical presentation of the sum-product algorithm, we would like to make a short example [102]. Let us consider the specific case with the real-valued global function defined in (3.15) that may represent a conditional joint probability mass function of a collection of discrete



Gathering separate product terms in the factor graph to compute the marginal $g_1(x_1)$. **Figure 3.10**

random variables, given some observation y. We are then interested in the function summary

$$p(x_1|y) = \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} g(x_1,x_2,x_3,x_4,x_5) = g_1(x_1). \quad (3.25)$$

Using the factorization given by (3.15), we derive

$$\begin{aligned} p(x_1|y) &= \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_A(x_1)f_B(x_2)f_C(x_1,x_2,x_3)f_D(x_3,x_4)f_E(x_3,x_5) \\ &= f_A(x_1)\sum_{x_2} f_B(x_2)\underbrace{\sum_{x_3} f_C(x_1,x_2,x_3)\underbrace{\underbrace{\sum_{x_4} f_D(x_3,x_4)}_{f_D}\underbrace{\sum_{x_5} f_E(x_3,x_5)}_{f_E}}_{f_{DE}}}_{f_{BCDE}} \quad (3.26)\end{aligned}$$

We observe immediately that $g_1(x_1)$ can be calculated by only knowing f_A and f_{BCDE}. The latter can be computed by just knowing f_B, f_C, and f_{DE}. Finally, f_{DE} can be calculated by just knowing f_D and f_E. The products can be assembled in the factor graph as shown in Fig. 3.10.

The meaning of messages and message passing

With each node in the factor graph we can now imagine an associated processor which is capable of doing local products and local function summaries. They may communicate together by sending and receiving messages from neighbouring nodes. The messages are whole distributions, i.e., the outcome of the function nodes, which are passed from one factor graph node to another connected by an edge. In general, they represent discrete probability mass functions, but also continuous probability distributions are included in the framework. Through the message passing behaviour, all information needed to calculate $g_1(x_1)$ becomes available at x_1. Hence, the information is distributed fully bi-directional on all branches of the network if we calculate the function summary for all variables.

3.3.1 The Sum-Product Update Rules

The sum-product algorithm uses very simple update rules

The simple computational update rule of the sum-product algorithm can be described, in all generality, as follows [97]:

> The message sent from a node v on an edge e is the product of the local function at v (or the unit function if v is a variable node) with all messages received at v on edges *other* than e, summarized for the variable associated with e.

Thus after calculating the product of all incoming messages including the local function, a summary function with respect to the considered node or the variable to which the resulting message is sent, has to be applied.

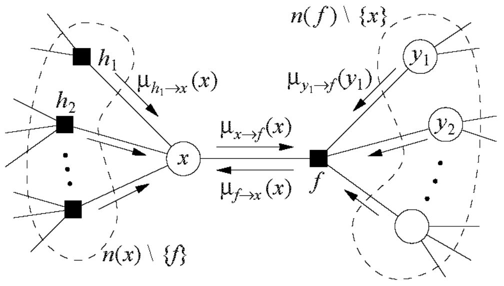
Let us denote $\mu_{v \to w}$ as the message sent from node v to node w. Then, as illustrated in Fig. 3.11, two different computations can be expressed for the update between a variable node and a function node and vice-versa:

variable-to-function update:

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x) \tag{3.27}$$

function-to-variable update:

$$\mu_{f \to x}(x) = \sum_{\sim\{x\}} \left(f(X_{n(f)}) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y) \right) \tag{3.28}$$



The sum-product algorithm update rules illustrated in a fragment of a factor graph. **Figure 3.11**

The set $n(v)$ denotes the neighbours of node v, i.e., $n(f) = \{x, y_1, y_2, \ldots\}$ and $n(x) = \{f, h_1, h_2, \ldots\}$. The particularly simple form of (3.27) is due to the fact that there is no local function to include, and the summary for x of a product of functions of x is simply the product itself. Equation (3.28) on the other hand generally involves complicated function multiplications and the summary operator application afterwards.

Simplified update rules for leaf nodes

Special cases arise when a variable node has only degree 2, i.e., it has only two neighbours. Then the message is just passed on. Leaf nodes, i.e, nodes with only one neighbour send the following messages:

$$\mu_{x \to f}(x) = 1 \tag{3.29}$$

and

$$\mu_{f \to x}(x) = f(x) \tag{3.30}$$

respectively, where, by a slightly abused notation, 1 denotes the unit function.

Message Passing Schedules 3.3.2

How should we schedule the updates?

So far, we have discussed the update rules of the sum-product algorithm in detail. So we know exactly how to calculate the messages. But then the question of how to initiate the updates and how to sequence the updates arises immediately. In fact, finding the optimal update schedule with respect to the least number of calculations is a non-trivial problem.

Initialization of the algorithm by unit messages

It is not clear how message passing is initiated, since a message generally depends on messages that have been sent before. Initially we thus suppose that a unit message is present on every edge on any given vertex. This means that every node has sent a unit function to all of its neighbours. With this convention, every node is in a position to send a message at any time starting from its equilibrium state.

Assume a synchronized message passing scheme

A second assumption is that of the synchronized message passing schedule, i.e., we assume a discrete time signal processing automaton synchronized with a global clock. Although this is not necessarily the case in practical implemenations (as we will see with the case of asynchronous analog VLSI networks described later on in this thesis), it is a fairly reasonable assumption that simplifies the understanding of the problem considerably. Thus only one message may be passed on any given edge in any given direction during one clock cycle, and this message replaces any previous message passed on that edge in any direction. We say that a vertex v has a message *pending* at an edge e if the message that v can send on e is potentially different from the previous message sent on e. For example, variable nodes initially have no messages pending, since they would initially only send a unit message, and this is exactly what they are supposed to send. On the other hand, function nodes will create pending messages at the beginning. In general, whenever a message arrives at a vertex v, it will create a pending message at every edge other than the one where the message has arrived. Thus, a message that arrives at a leaf node will not cause any other pending message, since there are no edges other than e. Thus leaf nodes *absorb* pending messages, whereas non-leaf nodes *distribute* pending messages.

Abundantly many different update schedules

Although we make the assumption of a synchronized automaton, there is a huge number of different message passing schemes possible. Fortunately, this is not a limiting problem. We only need to ensure a possible schedule to be *nowhere idle* to terminate the sum-product algorithm of a cycle-free factor graph. We call a schedule that sends at least one pending message at each clock tick *nowhere idle*. If the cycle-free factor graph has a finite number of nodes, i.e., it is a finite tree, the calculations of the sum-product algorithm are terminated in a finite number of steps. This is easily unterstood if one remembers that leaf nodes absorb pending messages and that in a finite tree every path eventually reaches a leaf. Conversely, factor

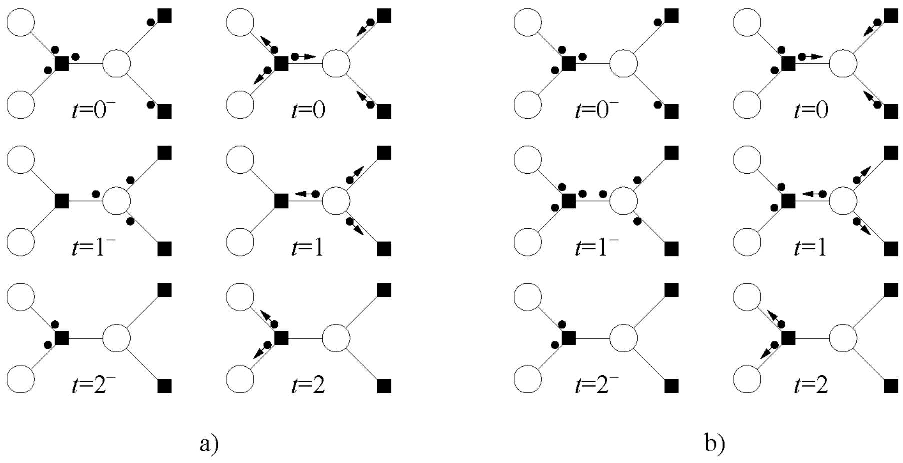


Figure 3.12 *Two message passing schedules for a simple factor graph: a) the flooding schedule and b) a two-way schedule.*

graphs with cycles never have a nowhere-idle message-passing schedule. The termination of the calculations is done in these cases arbitrarily by truncation or by a suitable stopping rule. The number of iterations to be performed to get satisfying results is generally determined by simulation.

The flooding schedule and the two-way schedule

Two examples of possible message passing schedules are shown in Fig. 3.12 where the flow of pending messages is visualized. A pending message is shown as a dot near the given edge, whereas the transmission of a message is indicated by an attached arrow to the dot. Time-flow is also indicated, with messages sent at non-negative integer times t. Figure 3.12a) shows the so-called *flooding schedule* in which all pending messages are sent during each clock cycle, starting with the pending messages at function nodes. Having in mind the original factor-graph formulation, one could equivalently say that first all variable nodes are updated and then a time step later all function nodes are updated and so on. As there are no direct connections between two variable nodes of an original factor graph, the order of calculation of the different variable nodes is not important. The same applies also to the function nodes. This alternative formulation of the flooding scheme is not correct if Forney's normal graphs are used. Though not optimal in the sense of the least number of computations, it is the most simple message passing scheme to implement in software since no special attention has to be paid to the order of the update. As a second example, Fig. 3.12b) shows a *two-way schedule*. Under this schedule we will have exactly one pending message that passes in each direction over a given edge for all time instances.

The forward-backward update schedule for trellises

A third update schedule that is particularly suitable for rectilinear factor graphs, e.g. the factor graph of a trellis, is the forward-backward scheme. As the name already states, one first needs to calculate in the forward direction of the rectilinear factor graph and then in the backward direction. A variable node and a function node is updated alternately until the end of the factor graph is reached. Then the whole process is reversed for the backward direction. Note that this is independent of the forward propagation and can be done at the same time. The forward-backward update schedule is a special case of the two-way schedule and uses the least possible number of calculations to build the function summary in a rectilinear factor graph. In fact, the forward-backward update schedule is essentially equivalent to the calculations sequence of the BCJR algorithm or forward-backward algorithm [103].

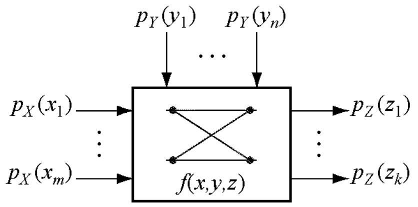
The fully parallel update schedule for asynchronous networks

Having in mind the application of the sum-product algorithm to the decoding of error-correcting codes using large asynchronous analog networks, none of the previously mentioned message passing schedules is appropriate. In fact our previous assumption of synchronicity is not met and the update schedule can not be controlled directly. If the calculation speed of all network nodes is equal, the scheduling scheme can be seen as a fully bi-directional asynchronous flooding scheme. Messages are transmitted at any time in every direction.

3.4 Probability Calculus Modules

Dissect the sum-product algorithm into building blocks

In the previous sections of this chapter we have introduced the basics of factor graphs and the sum-product algorithm which can be run on these graphs. We have also seen in the factor graph examples in Section 3.2.2 that the messages passed from one node to another often have the meaning of probabilities or probability density functions. To construct probability propagation networks, we consider in the following building blocks as



The building block of probability propagation networks. **Figure 3.13**

in Fig. 3.13. The building blocks compute a discrete probability mass function p_Z from the discrete probability mass functions p_X and p_Y as follows: let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ be finite sets. Let p_X and p_Y be the input probability mass functions defined on the sets (alphabets) $\mathcal{X}$ and $\mathcal{Y}$, respectively. Let p_Z be the output probability mass function on $\mathcal{Z}$ defined by

$$p_Z(z) = \gamma \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p_X(x) p_Y(y) f(x,y,z), \qquad \forall z \in \mathcal{Z}, \quad (3.31)$$

where f is a function from $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ into $\{0,1\}$ and where γ is an appropriate scale factor that does not depend on z. The scale factor γ is mathematically required to yield a probability distribution $p_Z(z)$ at the output whose sum is $\sum_i p_Z(z_i) = 1$. Equation (3.31) can be identified as the function-to-variable update of the sum-product algorithm as defined by (3.28). This processing step is used to build summary functions or marginals as defined by (3.17). Note that compared to the notation of (3.28), the different terms of (3.31) are slightly rearranged.

Indicator function f represented by trellis diagrams

The $\{0,1\}$-valued functions f can be illustrated by trellis modules as in Fig. 2.6, the inset in Fig. 3.7 and Fig. 3.14. Such a trellis module is a bipartite graph with labeled edges as introduced in Section 2.1.5. The set of left-hand vertices is $\mathcal{X}$, the set of right-hand vertices is $\mathcal{Z}$, and an edge between $x \in \mathcal{X}$ and $z \in \mathcal{Z}$ with label $y \in \mathcal{Y}$ exists if and only if $f(x,y,z) = 1$. Conversely, the trellis module uniquely defines f. In the context of coding theory, the binary indicator functions f are known as local indicator functions of the factorized global code membership indicator functions as we have seen in the factor graph examples in Section 3.2.2.

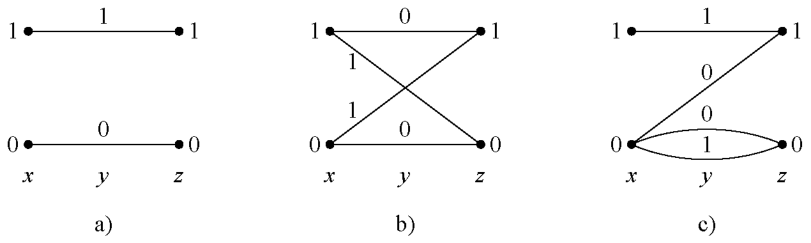


Figure 3.14 *Trellis representations of a) the equal gate, b) soft-XOR gate, and c) the backward reasoning on a soft-AND gate.*

3.4.1 Soft-Logic Gates

Equal Gate

In Section 3.2.1, we encountered the 'equal' function to allow node splitting. We will now have a closer look at this special function. The function $f(x,y,z)$ for this particular case is equal to 1 if and only if $x = y = z$ and $f(x,y,z) = 0$ otherwise. The corresponding trellis module is shown in Fig. 3.14(a). The indicator function f can be substituted in (3.31) to calculate the probability formulation of the output distribution which is given by

$$\begin{bmatrix} p_Z(0) \\ p_Z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_X(0)p_Y(0) \\ p_X(1)p_Y(1) \end{bmatrix}, \qquad (3.32)$$

where γ is a scale factor to satisfy $p_Z(0) + p_Z(1) = 1$. Thus the computations of the *equal gate* reduce to the component-wise product of p_X and p_Y. Such computations appear whenever independent information of some random variables is combined.

Soft-XOR Gate

Another common function f is defined as $f(x,y,z) = 1$ if and only if $z = x \oplus y$ with $\oplus$ denoting the standard modulo-2 addition and $f(x,y,z) = 0$ otherwise. The corresponding trellis diagram is depicted in Fig. 3.14b). With this function f the module of Fig. 3.13 becomes a *soft-XOR* gate: if p_X and p_Y are the distributions of two independent binary random variables X and Y, respectively, then the distribution of $X \oplus Y$ is p_Z given by

$$\begin{bmatrix} p_Z(0) \\ p_Z(1) \end{bmatrix} = \begin{bmatrix} p_X(0)p_Y(0) + p_X(1)p_Y(1) \\ p_X(0)p_Y(1) + p_X(1)p_Y(0) \end{bmatrix}. \qquad (3.33)$$

Soft-AND Gate in Backward Direction

Backward-reasoning on soft-logic gates is also possible

As a generalization of the soft-XOR gate, "soft" versions of all standard logic gates can be constructed by a suitable choice of f. But not only the standard forward-functioning direction of logic gates can be considered. Recall that soft-logic versions of factor graph nodes work fully bi-directionally. Thus, backward reasoning, i.e, from one of the inputs and the output to the other input, is possible, too. For example assume the logic AND function with inputs y and z and output x in the backward direction from x and y to z which is shown by the trellis diagram of Fig. 3.14(c). If we substitute the corresponding characteristic indicator function f into (3.31) we get

$$\begin{bmatrix} p_Z(0) \\ p_Z(1) \end{bmatrix} = \gamma \begin{bmatrix} p_X(0)p_Y(0) + p_X(0)p_Y(1) \\ p_X(0)p_Y(0) + p_X(1)p_Y(1) \end{bmatrix}, \qquad (3.34)$$

where again γ is needed to ensure a probability distribution p_Z at the output. This module is very unlikely to occur in coding, but would naturally appear in many applications of Bayesian networks.

3.4.2 Building Blocks with Multiple Inputs

Dissection property using Forney's normal graphs

The building block of Fig. 3.13 takes two input probability distributions and calculates one output probability distribution. In general however, factor graph nodes have a degree of more than three, which implies that building blocks with more than two inputs are needed. This is not an actual problem, since every building block with more than two input distributions can be dissected into a cascade of two-input building blocks as shown in Fig. 3.15b). This approach is also consistent with Forney's definition of normal graphs [99] where internal degree-2 variable nodes do not modify the messages from the incoming branch to the outgoing branch and can therefore be omitted from the drawing. Alternatively to the partitioning of

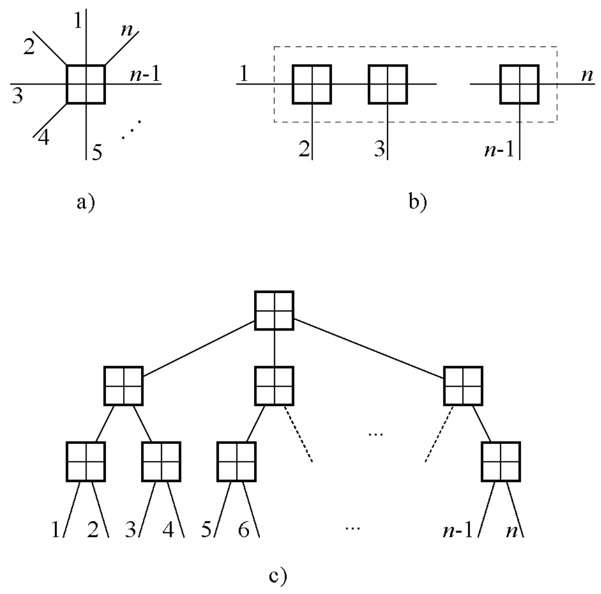


Figure 3.15 *A factor graph representation of the dissection property of building blocks with n-input probability distributions to make them compatible to 2-input building blocks of Fig. 3.13. The figure shows a) an n-input block, b) a cascade structure, and c) a tree structure. State nodes are intentionally omitted (see [99]).*

Fig. 3.15b), we might draw a tree structure which is an interesting option from the point of view of the signal delays. Because every processing node adds some delay due to parasitic effects in the transistor circuits we want to implement, the dissection solution of Fig. 3.15b) adds much more delay from port 1 to port n than to a port in the middle of the cascade, than the structure of Fig. 3.15c). By applying a tree structure the delay from one port to any other port is partly equalized.

Fully bi-directional building blocks can be constructed

Fully bidirectional factor graph calculation nodes for probability propagation networks can be constructed by using building blocks as in Fig. 3.13. To optimize the number of operations, intermediate results from certain building blocks may be reused. As an example we look at a soft-XOR gate of degree 4, i.e., a soft-XOR gate with four neighbours as shown in Fig. 3.16b).

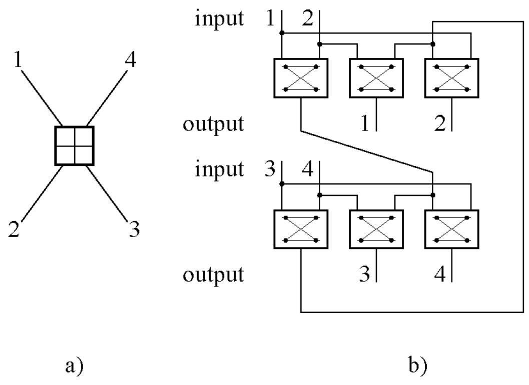


Figure 3.16 *A fully bidirectional factor graph node of a soft-XOR gate implemented using 2-input building blocks: a) factor graph node and b) calculations implemented using 2-input building blocks. The inputs and outputs denote the incoming and outgoing messages (probability distributions), respectively.*

The actual transistor level implementation of the building blocks in Fig. 3.13 is described in the following chapter. There we explore how a schematic of an actual probability gate can directly be generated by simply knowing its corresponding indicator function f.

Chapter 4

Circuit Implementation

In Chapter 3 we learnt how to build a factor graph description of a given code, and how to augment it into the factor-graph description of a decoder, on which the sum-product algorithm can be applied to find a symbol-wise MAP solution of the decoding problem. Now we will present a generic solution for the transistor-level implementation of these building blocks. Furthermore, practical design aspects and design issues are discussed.

Basic Circuit 4.1

Main operations of the building blocks: multiplication and summation

The underlying equations of the general building block of Fig. 3.13 can be separated into two main computational parts: first, component-wise products of the incoming discrete probability distributions have to be built. Second, product terms that belong to the valid configurations, i.e., fulfill the binary indicator function f, are summed for the appropriate terms of the discrete output probability distribution.

Signal Summation 4.1.1

Summation is easy in current domain

Summing signals is easily accomplished in the current domain, i.e., when signals are represented by currents. This is due to Kirchhoff's current summation law, which states that the sum of all currents along the incoming branches to a given node is equal to the sum of all currents of the outgoing branches. If only one outgoing branch exists, it automatically carries the sum current of all incoming branches. This means that current summation is simply done by connecting wires. As in the trellis diagram, the selective sum needed for the implementation of

(3.31) can be built connecting the appropriate wires (terms) of the pair-wise products. Doing the same operation in the voltage domain would definitively need more circuitry. Most of the traditional voltage-adder circuits, such as for example the opamp based voltage adders (see e.g. [104]), are anyway based on the principle of current addition, but additionally they need domain transformation from voltage to current and back. Furthermore, also the discrete-time switched capacitor (SC) adders are not direct voltage adders. The voltage signals applied to SC circuits are inherently transformed to a charge representation on the capacitors. Hence, the addition is based on a charge transfer from one capacitor to another, and finally the charges are inherently transformed back to a voltage. A comprehensive overview of the general SC techniques can be found in [43, 44]. Note that the SC-based circuits are not anymore continous-time, asynchronous circuits and thus not suitable for our intended asynchronous networks. But still, this principle is extensively used in SC filter circuits.

4.1.2 Basic Translinear Network Theory

The term 'translinear' was coined by Barrie Gilbert in 1975

Building the outer products of the incoming discrete probability distributions is a more complex task than doing the summation of currents. But luckily, Gilbert coined the term *translinear* network in 1975 [105]. This thinking style is also current-based and thus fits well to our extremely efficient current addition primitives described in the previous subsection. The translinear networks (TN) are based on an astonishingly simple theory [106, 107]. The heart of translinear networks are bipolar junction transistors (BJT). These transistors exhibit an exponential characteristic of the collector current in forward active mode according to

$$I_C = A_E J_S(T) e^{V_{BE}/nU_T} = I_S(T) e^{V_{BE}/nU_T}, \qquad (4.1)$$

where A_E is the emitter area, J_S is the saturation current density, and I_S is the saturation current. The absolute temperature is T and U_T denotes the thermal voltage kT/q. The factor n is the 'emission coefficient', an indicator of imperfect emission of electrons, generally close to unity. We use the standard notation for the transistor terminals (see e.g. [108, 109]). A similar characteristic can be found for the drain current of a weakly inverted

MOS transistor with

$$I_D = \frac{W}{L} J_0(T) e^{(V_G - nV_S)/nU_T} = I_0(T) e^{(V_G - nV_S)/nU_T}, \tag{4.2}$$

where W and L are the width and the length of the transistor, respectively, J_0 is a specific current density comparable to the one in the BJT case, and I_0 is the corresponding specific current. The slope factor n falls in the range of $[1\ldots 2]$, generally close to 1.5. Note that the mechanisms leading to n in the case of a bipolar transistor and a MOS transistor are not the same, although the effects thereof are comparable.

Kirchhoff's voltage law applied to a special arrangement of transistors

With these definitions in mind, we assume the circuit configuration of Fig. 4.1. Furthermore, we assume that all transistors have identical geometric dimensions. According to the translinear theory, we can form a closed loop of an even number N of base-emitter junctions (gate-source steps), $N/2$ in each direction (clock-wise (CW) and counter-clock-wise (CCW)) and arbitrarily ordered. According to Kirchhoff's voltage law, we write

$$\sum_{\text{CW}} V_{\text{BE}_i} = \sum_{\text{CCW}} V_{\text{BE}_i}. \tag{4.3}$$

A voltage sum corresponds to current multiplications

Inserting (4.1) into (4.3) leads to the conclusion that the voltage sum over all V_{BE_i} corresponds to the product of the collector currents I_{C_i} normalized to I_{S_i}:

$$\prod_{\text{CW}} \frac{I_{\text{C}_i}}{I_{\text{S}_i}} = \prod_{\text{CCW}} \frac{I_{\text{C}_i}}{I_{\text{S}_i}}. \tag{4.4}$$

The translinear principle is also valid in MOS technology

This result is totally independent of the temparature T and (at least on the level of principle) also independent of the current gain β of the transistors. Several distinct loops may share some of their base-emitter junctions. The results, as presented for the bipolar case, are also fully valid for weakly inverted MOS transistors [110]. The generalized translinear principle for quadratically behaving MOS devices leads to somewhat different, but also very useful expressions [111–113]. The MOS translinear principle expresses that the sum of the square-rooted currents from the clock-wise oriented transistors equals that of the opposite direction.

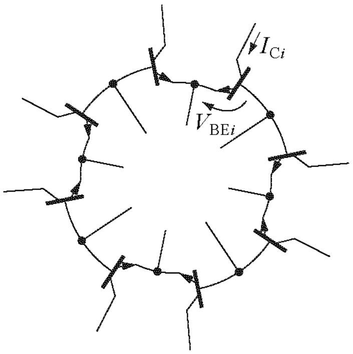


Figure 4.1 *A simple translinear loop.*

4.1.3 Core Circuit for Matrix Multiplications

A transistor matrix inspired by the translinear principle

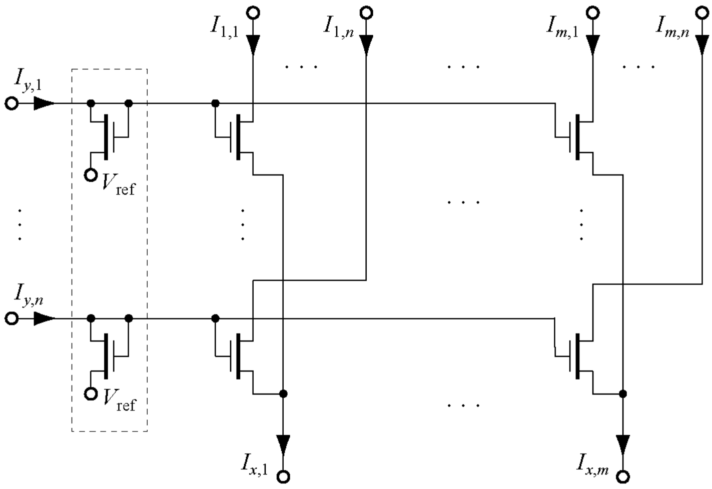
Using (4.4), it is relatively easy to construct a circuit that creates pair-wise products of two incoming probability distributions. The fundamental circuit that underlies the realization of all the building blocks is shown in Fig. 4.2. Its inputs are the currents $I_{x,i}$, $i = 1,2,\ldots,m$ and the currents $I_{y,j}$, $j = 1,2,\ldots,n$. Its outputs are the currents $I_{i,j}$. All transistors in Fig. 4.2 are assumed to be ideal voltage-controlled current sources according to (4.1) and (4.2) for BJT and weakly inverted MOS transistors, respectively. Our terminology and notation correspond in the following to the weakly inverted MOS case. As we will see later in this section, the function of the circuit is then given by

$$I_{i,j} = I_z(I_{x,i}/I_x)(I_{y,j}/I_y) \tag{4.5}$$

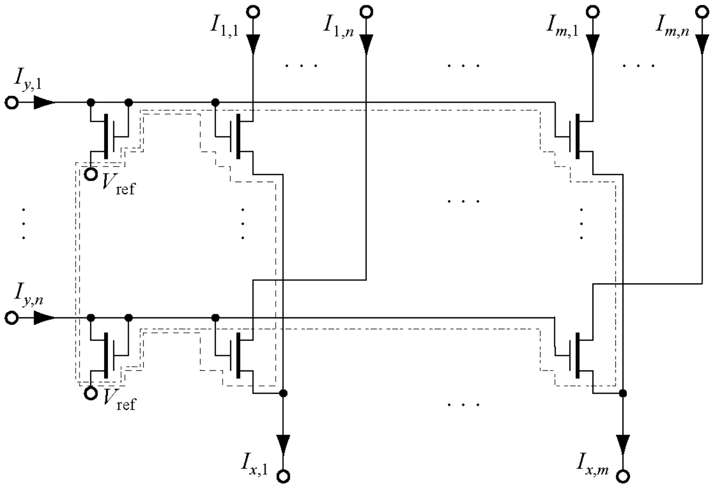
with $I_x \triangleq \sum_{i=1}^{m} I_{x,i}$, $I_y \triangleq \sum_{j=1}^{n} I_{y,j}$, and $I_z \triangleq \sum_{i=1}^{m}\sum_{j=1}^{n} I_{i,j} = I_x$. The circuit thus computes the scaled pairwise product of the two probability mass functions $\tilde{p}_X(i) \triangleq I_{x,i}/I_x$, $i = 1,\ldots,m$, and $\tilde{p}_Y(j) \triangleq I_{y,j}/I_y$, $j = 1,\ldots,n$.

Matrix multiplication for two discrete probability distributions

The application of the circuit of Fig. 4.2 to the computation of (3.31) is now straightforward. Let $\mathcal{X} = \{x_1,\ldots,x_m\}$ and $\mathcal{Y} = \{y_1,\ldots,y_n\}$. The input terminals of the circuit are fed with the currents $I_{x,i} \triangleq I_x p_X(x_i)$ and $I_{y,j} \triangleq I_y p_Y(y_j)$, respectively, where the sum currents I_x and I_y can be chosen freely in the range where (4.1) and (4.2) hold for all transistors in the



Fundamental circuit using two input distributions, each represented as a current vector. **Figure 4.2**



Fundamental circuit with $(n-1)m$ independent translinear loops. Two of the loops are shown by the dashed polygons. **Figure 4.3** circuit. The output currents then equal $I_{i,j} = I_z p_X(x_i) p_Y(y_j)$, $i = 1,\ldots,m$, $j = 1,\ldots,n$.

The computation of (3.31) is completed by summing the currents $I_{i,j}$ for each $z \in \mathcal{Z}$ for which $f(x_i, y_j, z) = 1$. If a term $p_X(x_i) p_Y(y_j)$ is used more than once, the corresponding current $I_{i,j}$ must first be copied a corresponding number of times.

Translinear Network Interpretation

The circuit may be analyzed by first drawing all $(n-1)m$ independent translinear loops in the circuit of Fig. 4.3 and writing down the corresponding loop equations. Second, the constraints on the three sum-currents of the individual distributions can also be stated immediately. Finally, the system of $(n-1)m + 3$ equations can be brought to the form of (4.5).

Standard Large-Signal Analysis

Large-signal proof of the basic function

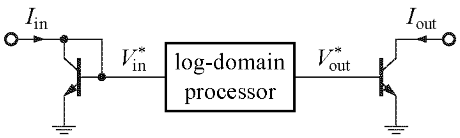
The result of (4.5) may also be verified by standard large-signal analysis as given in the following proof using the first order approximations of the drain (or collector) current given by (4.2) and (4.1), respectively.

Proof of (4.5): Let $V_{x,i}$ and $V_{y,j}$ denote the potentials at the input terminals for $I_{x,i}$ and $I_{y,j}$, respectively. On the one hand, we have

$$\frac{I_{i,j}}{I_{x,i}} = I_{i,j} \Big/ \sum_{\ell=1}^{n} I_{i,\ell} \tag{4.6a}$$

$$= I_0\, e^{\frac{V_{y,j} - n V_{x,i}}{n U_\mathrm{T}}} \Big/ \sum_{\ell=1}^{n} I_0\, e^{\frac{V_{y,\ell} - n V_{x,i}}{n U_\mathrm{T}}} \tag{4.6b}$$

$$= e^{\frac{V_{y,j}}{n U_\mathrm{T}}} \Big/ \sum_{\ell=1}^{n} e^{\frac{V_{y,\ell}}{n U_\mathrm{T}}}. \tag{4.6c}$$



The principle of a log-domain signal processor. **Figure 4.4**

On the other hand, we have

$$\frac{I_{y,j}}{I_y} = I_{y,j} / \sum_{\ell=1}^{n} I_{y,\ell} \tag{4.7a}$$

$$= I_0 e^{\frac{V_{y,j} - nV_{\text{ref}}}{nU_{\text{T}}}} / \sum_{\ell=1}^{n} I_0 e^{\frac{V_{y,\ell} - nV_{\text{ref}}}{nU_{\text{T}}}} \tag{4.7b}$$

$$= e^{\frac{V_{y,j}}{nU_{\text{T}}}} / \sum_{\ell=1}^{n} e^{\frac{V_{y,\ell}}{nU_{\text{T}}}} . \tag{4.7c}$$

Combining (4.6c) and (4.7c) finally yields (4.5). □

Log-Domain Signal Processing Interpretation

A third interpretation of the circuit functioning of Fig. 4.2 can be given by the log-domain signal processing concept. In this technique, the input signals are compressed by the logarithm function, then the actual processing task is fulfilled in the compressed signal domain and finally the signal is again expanded by the inverse function (exponentiation) as shown in Fig. 4.4. This concept was first introduced by Adams in 1979 [114] to achieve electronic gain and cut-off frequency control in filters. It was afterwards extensively developed by different researchers [115–118] in its main application field of filtering. Log-domain signal processing is also very attractive for low-power and low-voltage applications [119, 120]. The main aspects of log-domain signal processors (or generally speaking: companding signal processors) are low-voltage operation, enhanced dynamic range and high-frequency operation. Furthermore, filters exhibit a wide frequency tuning range. The issues of companding signal-processors such as signal-dependent noise, distortions due to device mismatch, intermodulation production by interference and increased bandwidth requirements are mainly a problem in linear circuits. Non-linear processors such as decoders are much more robust against these problems.

4.1.4 Log-Likelihood Interpretation of Input and Output Distributions

Log-likelihood ratios appear between the gate voltages of the diode-connected transistors

The circuit of Fig. 4.2 can be operated differently, if we omit the input diode-connected logarithm transistors on the left-most column as in Fig. 4.5. Instead of using the input currents $I_{y,i} \triangleq I_y p_Y(y_i)$ we apply the voltages equivalent to the different gate voltages $V_{y,i} \triangleq nU_\mathrm{T} \ln[p_Y(y_i)] + const$. If we consider the voltage difference

$$V_{y,1} - V_{y,n} = nU_\mathrm{T}(\ln[p_Y(y_1)] - \ln[p_Y(y_n)]) = nU_\mathrm{T} \ln \frac{p_Y(y_1)}{p_Y(y_n)} \tag{4.8}$$

we immediately recognize the log-likelihood ratio representation of inputs 1 and n. Through the thermal voltage U_T, this log-likelihood-ratio representation by $(n-1)$ voltage differences is temperature dependent. The representation is equivalent to the probability representation by n currents if we know exactly the absolute temperature T (see Section 4.4.4 for a detailed analysis of the temperature dependence). In principle, we can thus freely choose the input representation of p_Y. By the same means of a battery of diode-connected transistors, the input and output probability distribution p_X and p_Z, respectively, can be transformed into a log-likelihood-ratio representation.

4.2 Soft-Logic Gates and Trellis Modules

Main pitfalls appear already in very simple modules

We will now recall the examples of Section 3.4.1 and construct the corresponding circuit diagrams. In many ways, they exhibit the main problems that may arise during the construction of such trellis modules. We will discuss these issues during the presentation of the individual examples.

Butterfly trellis is visible in the circuit diagram

Soft-XOR gate. Let us start with the most simple module, the soft-XOR gate. It can be drawn directly using its butterfly trellis section that has been derived from its binary indicator function f. This circuit module is shown in Fig. 4.6. Like all modules

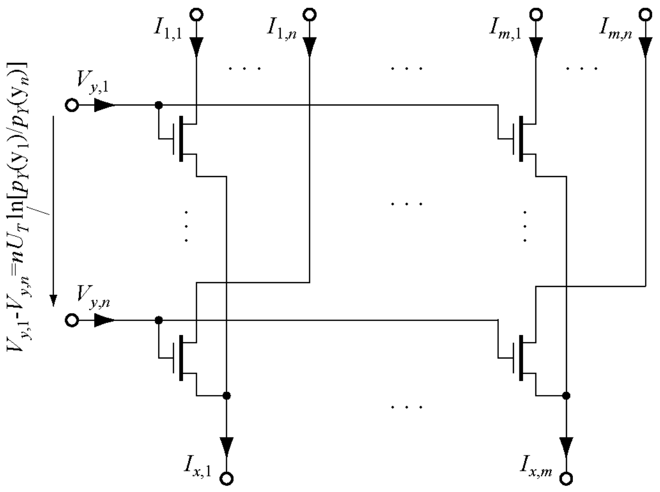


Figure 4.5 *The generic multiplication matrix with voltage inputs for the p_Y distribution representing log-likelihood ratios.*

with binary input distributions, it consists of 6 core transistors forming the multiplication matrix and its characteristic connection pattern of the trellis. In fact the trellis pattern will be directly visible on silicon if the devices are properly arranged. This fact may be helpful in order to create automatic tools for generating such building blocks in a chip-design environment. All product terms are used to build the output probability distribution p_Z. The output terms are mirrored by the current mirrors on top of the kernel circuit. The input currents $I_{x,i}$ are also passed through an input current mirror. By doing this, the module gets freely cascadable by simply connecting the output current vector I_z to one of the input current vectors I_x and I_y, respectively, of the next circuit section. This method marks the most simple way of interconnection of several building blocks. Note also that all the current mirrors may equally well operate in the strong inversion region of MOS transistors, thereby having a standard quadratic behaviour.

Soft-XOR gate is a version of the Gilbert multiplier

By defining a proper data representation (different from our probability mass functions), the soft-XOR circuit can be identified as a version of the so-called Gilbert multiplier [121] for two real-valued differential inputs. But in Gilbert-multipliers,

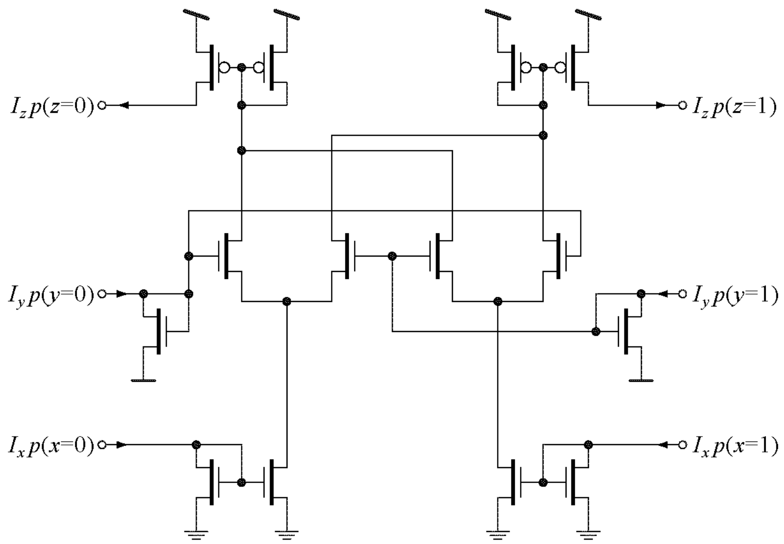


Figure 4.6 *Transistor-level implementation of the soft-XOR building block.*

the inherent scaling of the output signal by the sum of the current inputs $I_{x,i}$ is generally not wanted. Furthermore, the output values are limited to this current sum which limits the application of the Gilbert multiplier for general-purpose real-value multiplications even more.

Other multiplier implementations exist

Beside the Gilbert multiplier [121], many other forms of solid-state multipliers have been developed since. Some of them rely on cross-coupled transistor pairs both in MOS and bipolar technology [122–124], but also different approaches exist such as the quadratic-translinear principle [125], the quarter-square principle [126, 127] and floating-gate MOSFET techniques [128, 129]. But none of them reaches the outstanding simplicity of doing a multiplication matrix with only one transistor per multiplication. Thus, it is hard to see such bulky multiplier circuits in very large analog probability propagation networks where thousands if not even millions of such multipliers are needed.

Dummy paths need a proper termination

Equal gate. The second example is identical to the previous example, except for the two special signal paths that attract our attention. The two outputs of the transistors in the middle of

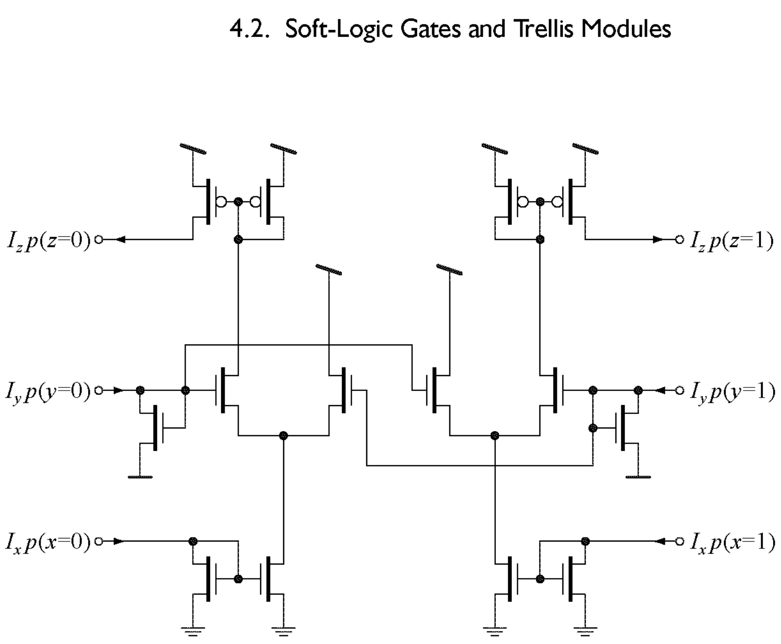


Figure 4.7 *Transistor-level implementation of the equal gate building block.*

the circuit diagram of Fig. 4.7 are not used for the computation of the output distribution of the equal gate. Thus they represent dummy paths. They are terminated in order to ensure the correct operation of the two transistors of the kernel circuit. In practical circuits, one tries to keep the drain voltage levels of the output transistors of the computation kernel at the same level. This reduces the effect of the finite output resistance of these transistors which is largely drain-voltage dependent. One possible solution could be to pass the dummy currents through a diode-connected p-type transistor as in the input section of a current mirror. Through its heavy compression, the voltage swing is kept minimal at this point. A more effective, but also costly solution in terms of circuit overhead would be the addition of cascode stages for each output transistor of the kernel.

Asymmetric trellis diagrams have to be expanded for a clean circuit implementation

Let us come back to the dummy product terms. The corresponding transistors cannot simply be omitted because for a correct operation, the multiplication matrix has always to enclose all $n \cdot m$ transistors. By omitting certain transistors in the multiplication matrix, the input distributions would be changed by building them only on a subset of the terms. The observation that all product terms have to be present in the multiplication

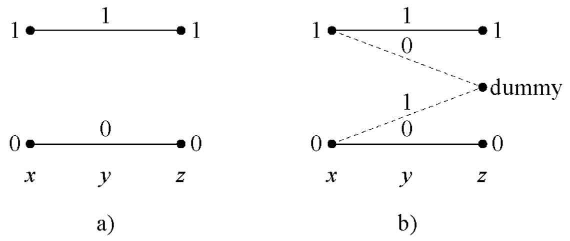


Figure 4.8 *Trellis expansion operation for a correct circuit implementation. Fig. a) shows the original trellis of the equal gate and b) the expanded trellis of the same gate, ready for transistor implementation.*

matrix can also be illustrated on the trellis diagram level of the binary indicator function f. At every left state of the incoming distribution p_Z, branches for every incoming symbol of y have to be drawn. Valid configurations, i.e., branches actually drawn in the final trellis diagram, lead to the right states, whereas non-valid configurations lead to some dummy right state. Exactly these branches leading to the dummy state represent the dummy paths that are afterwards present in the circuits. As a general statement we can say that every state of the trellis diagram has to carry the same number and the same type of the outgoing branches to enable the correct circuit implementation. This expansion, sometimes necessary for a circuit implementation, is shown schematically in Fig. 4.8 for the equal gate.

Backward reasoning through logic gates

AND gate backwards. In our third example, the backward reasoning of a soft-AND gate, we encounter again a speciality. As already shown in the trellis diagram of Fig. 3.14c), the product term $p_X(0)p_Y(0)$ is used twice. Compared to a voltage duplication, current duplicates are not for free. Thus instead of first adding product terms according to the binary indicator function f, we first have to copy the currents of the individual product terms by means of a current mirror and build the sum only afterwards. The product term $p_X(1)p_Y(0)$ is not used at all in this case. This leads to the circuit implementation of Fig. 4.9.

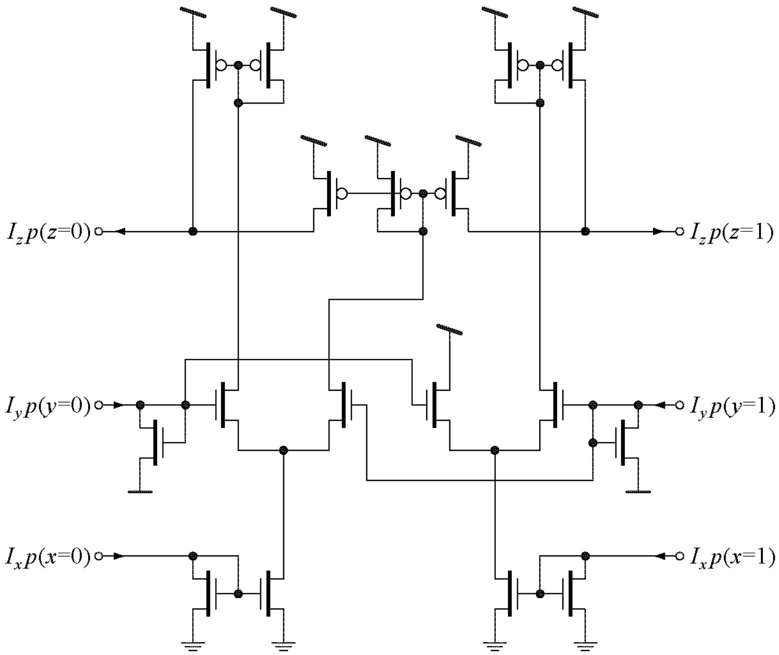


Figure 4.9 *Transistor-level implementation of the backward reasoning of an AND gate building block, i.e., from one input and the output back to the other input of the AND gate.*

Any trellis diagram can be implemented

General Trellis Diagrams. The implementation of a general trellis diagram is straightforward if one thinks of the previous three examples. All the ingredients necessary for the circuit implementation are presented therein. For a trellis module with n and m elements of the incoming probability distribution, an $n \times m$ transistor matrix is drawn. Additionally, for the distribution p_Y, n diode-connected transistors are drawn and connected to the matrix according to Fig. 4.2 for current inputs. If the log-likelihood representation is chosen, these transistors are omitted. The inputs for p_X are completed by current mirrors and the outcoming product terms of the transistor matrix are then connected according to the trellis diagram. Unused product terms are connected to a dummy node. Finally, for each term of the output distribution p_Z, a current mirror can be added to allow cascading the module directly with other modules.

4.3 Connecting Building Blocks

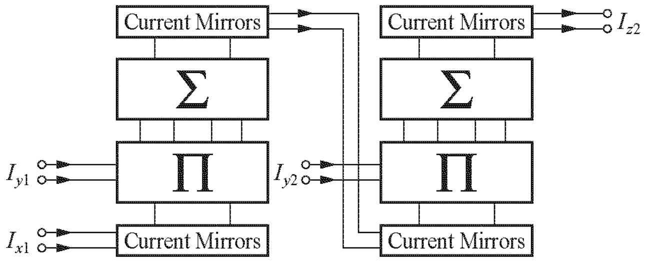
Large networks need the connection of many building blocks

In order to build large probability propagation networks, many building blocks have to be cascaded. Since the input and output signals can be represented by current vectors or voltage vectors, many different possibilities for cascading single stages may be considered. The easiest solution has already been shown in the examples of Section 4.2. Solutions different from the one previously shown are presented in the following. Another issue may be the current loss by unused product terms of the multiplication matrix. If many stages are cascaded, the currents may tend to zero and vanish in the noise floor of the electronic circuit. Therefore, the current levels have to be brought to a reasonable level from time to time. Solutions to that problem are presented in the Section 4.3.3

4.3.1 Current- or Voltage-Mode Connections?

Ongoing debate voltage mode vs. current mode

The debate between voltage-mode adherents and current-mode supporters is going on for already a few years. Both of them think that their thinking-style is the most appropriate for high-performance integrated-circuit design. But in our opinion, it does not matter whether one wishes to implement a voltage



Interconnection of several modules by simple current mirrors. **Figure 4.10**

mode circuit or a current-mode circuit. It is much more important whether the circuit fits in the overall system or not. It is even shown in [42] that no fundamental difference exists between the two domains. Thus we will look at the question of choosing voltage-mode or current-mode interconnects only with respect to the best fit in the complete system.

Connecting building blocks is most simple with current mirrors

As we have already seen in Section 4.2, the most simple solution for cascading our basic building blocks is to pull out the currents at the output by simple current mirrors and feed them directly to the input of the following building block. This introduces the least circuit overhead in terms of transistor count. Extra transistors can be added to the current mirrors to duplicate the currents several times if they are needed. By doing this, no superfluous domain-changes have to be made. One module just smoothly connects to another as is shown in Fig. 4.10.

The building blocks may be analyzed with non-traditional approaches

In the eyes of traditional IC-designers, this whole circuit is a real nightmare. Generally, they argue that there exist paths that may have probablity values of zero, thus conduct almost no current, and therefore are infinitely slow. Their solution would be to add bias current-sources to turn the whole circuit into a full class-A circuit. If we keep in mind the application field of our circuits, this would add a non-tolerable overhead to large networks of our building blocks. Fortunately, such a traditional design approach is not needed in our case. We can even argue intuitively that the large currents, i.e., the large probabilities, determine the transient behaviour of our circuits mostly, as is the case with digital simulations. Small currents (or probabilities) are not negligible, but far less important in the case of decoding.

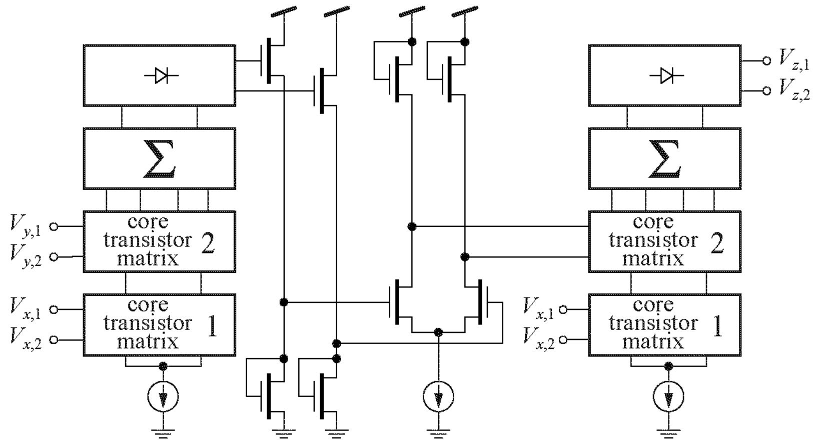


Figure 4.11 *Interconnection of several modules by level-shifter circuits as described by Moerz et al. [70].*

Voltage-mode connections are also possible

One can think of voltage-mode circuits as interconnects between voltage output modules and voltage input modules. In this case we save one connection because of the definition of the log-likelihood ratios of (4.8). The actual voltage-mode connection is established by level-shifters. The most simple implementation of such a level-shifter would be a source-follower circuit (see e.g. [130]). Unfortunately, the outputs of the source followers cannot be connected directly to the inputs of the following stage. Additional domain transformations have to be added by means of long-tailed pairs operating in the linear region and a linear back-transformation to voltage mode. In contrast to the actual computation circuit, the interconnection circuits are then fully class-A. Moerz *et al.* [70] have chosen this approach in their chip implementation as shown in Fig. 4.11. The diode-connected transistors in the interconnect circuit are designed to work in the linear region and act as resistors. Note also that the circuit has to be designed with transistors larger than minimum size to obtain the desired performance. The interconnect circuit works well for small alphabet sizes of the input distributions but is a potential source of severe signal errors due to the many domain changes. A careful design will eliminate these problems, but at the cost of a larger circuit area. Additionally, the voltage-mode connections suffer from inher-

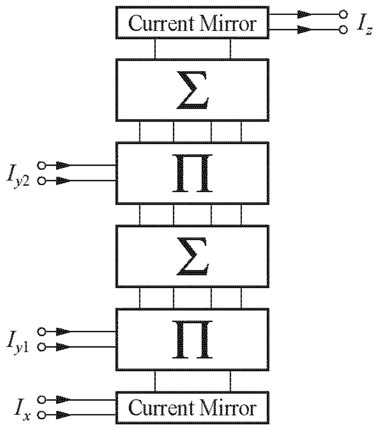


Figure 4.12 *Stacking several core circuits to get more than two input distributions for one building block.*

ent temperature tracking problems as we will discuss in Section 4.4.4.

4.3.2 Stacking and Folding Building Blocks

Stacking circuits for multiple-input building blocks

Beside simply cascading building blocks by current mirrors, there exist two other schemes which may save transistors under certain circumstances. As a first possibility, circuit modules may be stacked. This avoids the use of additional current mirrors. The number of stacked modules is thereby strictly limited by the available headroom left by the supply voltage. Generally, the stacking technique is not possible for low-voltage applications, i.e., below 5 V, using state-of-the-art silicon technologies. Apart from the area savings, the main advantage of stacking is a reduced power consumption. The current mirror sequence for a whole discrete probability distribution represents a dummy path in the sense that the vertical current through that path does not contribute to the actual computation. By omitting as many of these paths as possible, the total power consumption may be drastically reduced.

Folding reduces the current-mirror count

A second solution to the unwanted vertical current dissipation problem is creating folded building blocks as shown in

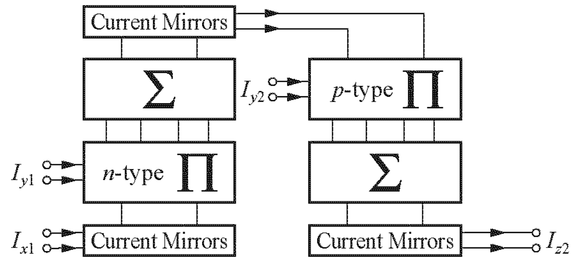


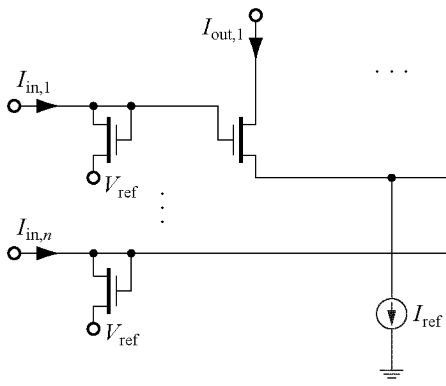
Figure 4.13 *Folding n-type and p-type building blocks saves half of the current mirrors.*

Fig. 4.13. Hereby, the outputs of a first, normal n-type module, i.e., a module with n-type transistors, are passed by current mirrors to a second, p-type module. A p-type module is just built by exchanging all n-type transistors by p-type transistors. From its appearance point of view, it may also be called the 'head-over' version of the building block. By cascading folded versions of the building block, half of the current mirrors are saved. Additionally, no superfluous vertical currents are flowing. Unfortunately, this interconnecting technique is not suitable for BiCMOS technology, since generally no good-quality vertical BJTs are available. A second drawback of this folding technique is its heavy use of p-type MOS transistors which are by a factor of about 3 larger than their n-type counterparts (due to slower mobility of holes, etc.). This makes p-type circuits considerably larger than n-type circuits.

4.3.3 Scaling Probabilities

Underflows in data representation may occur in the sum-product algorithm

The second issue in interconnecting many building blocks, besides choosing the right topology, is induced by the current loss of unused product terms. The problem is inherent to the sum-product algorithm, since the multiplication of a huge number of positive real values below unity tends towards zero. Thus one has to pay attention to underflows in data representation even in digital floating-point implementations. This issue is generally resolved by scaling up the terms of the discrete probability distributions from time to time to values acceptable in the data representation.



A vector normalizer circuit according to Gilbert [131]. **Figure 4.14**

Signals may fade into the noise floor

In analog implementations of large probability propagation networks, the fading problem of the probability values is even more pronounced. The noise floor of the electronic circuits puts a lower limit on the minimal currents representing the signals in analog electronic systems. As a rule of thumb, current-signal levels should not fall below a few 10^{-13} A. This limit is about in the same region where the exponential law of (4.1) and (4.2) is not valid anymore. So the probability mass functions represented by current vectors have to be scaled up electronically. In [131] Gilbert has presented an array-normalizer circuit that implements exactly the needed scaling function. The circuit of Fig. 4.14 can easily be explained by using the translinear principle [107].

Very efficient scaling circuit

The scaling circuit of Fig. 4.14 is actually a degenerate version of the fundamental circuit of Fig. 4.2 with $m = 1$ and $I_{x,1}$ fixed to some constant current. Its function is given by

$$I_{\text{out},j} = I_{\text{ref}} I_{\text{in},j} / \sum_{\ell=1}^{n} I_{\text{in},\ell}. \tag{4.9}$$

Adding scaling circuits at the output eliminates current fading

The scaling operation can be integrated into the building blocks. Whether it is at the output of the current module (Fig. 4.15a) or at the input of the next stage (Fig. 4.15b) does not matter at first glance. Scaling at the output has the advantage that small currents are brought back to a nominal level immediately after the block that caused the current loss and therefore speeds up the

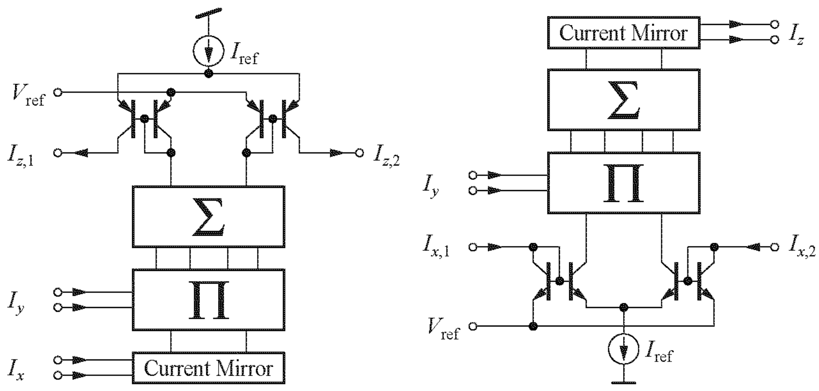
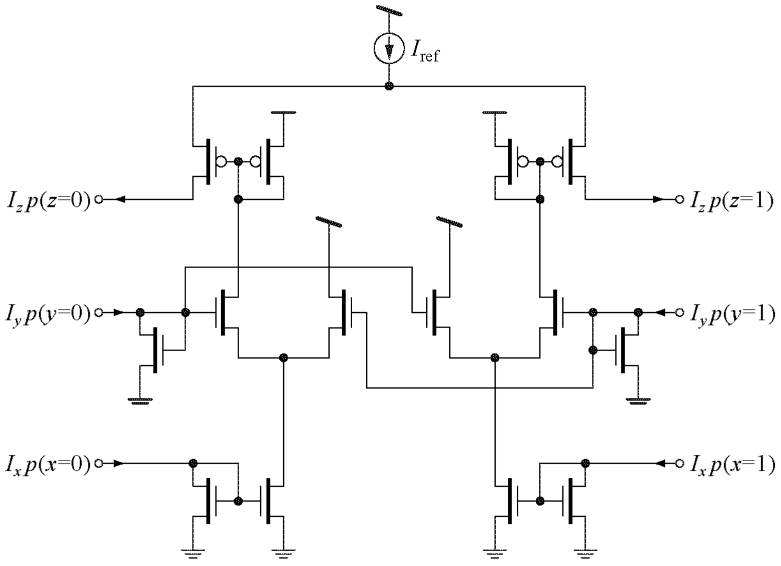


Figure 4.15 *Scaling circuits with a) scaling at the output and b) scaling at the input.*

overall network. Unfortunately, BiCMOS technologies generally provide fast vertical NPN transisitors and only relatively slow lateral PNP transistors, which would have to be used for the scaling circuits located at the output of a module. Thus, for practical reasons, scaling in high-speed circuits using BiCMOS technology will mainly be accomplished in the input stage of the following module as in Fig. 4.15b). An example of a module with scaling at the output is shown in Fig. 4.16. For this we have retaken the equal gate of Fig. 4.7 and added a scaling circuit using weakly inverted pMOS transistors.

By adding current mirrors at the outputs, the scaling circuit can even be expanded into a building block of its own. This solution is better suited for low-voltage implementations, since only three transistors are stacked in any given module. But this is at the cost of increased power consumption, since additional modules are introduced. In many applications, scaling after every module is not necessary. It suffices to scale only after every third or fourth module, or even later. But this is highly application specific and has to be investigated for every considered case or code.



The equal gate circuit of Fig. 4.7 with scaled-up outputs. **Figure 4.16**

Implementation Issues 4.4

Several problems arise in an actual implementation of a probability-propagation network

Both simulations and physical implementations of our decoding networks have shown a high immunity against non-ideal circuit behaviour. The decoder of Fig. 5.3, which we will discuss in Chapter 5, has been implemented using discrete BJT transistors out of the box, i.e., without any preliminary matching selection. Nevertheless the overall precision for this decoding network is within 5% of the theoretical values. This result should give a first impression of the robustness of the new technique. In the following subsection, we will give deeper insight into several non-ideal effects that may occur during physical implementation of the proposed analog probability propagation networks, such as device matching, temperature matching, and finite input resistance and output conductance of the transistors. Appropriate countermeasures against these effects are given if available.

4.4.1 Device Matching Considerations

In bio-inspired circuits, precision is gained on system level

Mismatch of transistors usually affects the functionality of analog circuits more than digital ones. Correct operation can often only be guaranteed by choosing large device sizes, which slows down the operation speed of the circuit. Fortunately, systems following the *bio-inpired* design style [11] seem to possess one big advantage over conventional analog systems: precision is gained on the system level by a parallelization of many computational units which are not inherently precise by themselves. In such systems, small device sizes do not degrade the overall precision significantly, which makes high-speed operation possible.

Quantification of the current errors of a single bipolar transistor

In a first attempt, we try to quantify the correspondence between errors in the probability representation (current domain) and the errors in the log-likelihood representation (voltage domain). To do this we recall the collector current equation (4.1) of a single bipolar junction transistor. A relative collector current error ε is introduced. Through some mathematical operations, this error on the left-hand side of (4.10) is then propagated until its influence on the input base-emitter voltage is visible:

$$\begin{aligned} I_C(1+\varepsilon) &= \left(I_0 e^{\frac{V_{BE}}{U_T}}\right)\cdot(1+\varepsilon) && (4.10)\\ &= I_0\, e^{\frac{V_{BE}}{U_T}} \cdot e^{\ln(1+\varepsilon)} \\ &= I_0\, e^{\frac{V_{BE}+U_T\ln(1+\varepsilon)}{U_T}} && (4.11)\\ &\approx I_0\, e^{\frac{V_{BE}+U_T\varepsilon}{U_T}}, && (4.12)\end{aligned}$$

where (4.12) has been derived from (4.11) by using the Taylor approximation $\ln(1+\varepsilon) \approx \varepsilon$ for small ε. This approximation actually over-estimates the influence of relative current errors on the the base-emitter voltage for larger ε. We observe in (4.12) that relative errors ε in the collector currents can be equivalently expressed as absolute errors on V_{BE}. If we assume a voltage swing of $\Delta V_{BE} = 300$ mV corresponding to a usable current of about 5 decades (actually $\Delta V_{BE} = 60$ mV/dec in the case of BJTs), we can put the absolute error on V_{BE} in relation to the total swing. For example, given a relative collector current error $\varepsilon = 10\%$, which results in a voltage error δV_{BE} of about 2.59 mV, this ratio is then $300/2.59 = 116$. This corresponds to an equivalent resolution of V_{BE} of about 7 bits. Table 4.1 summarizes the relationship between the dynamic range of the circuit and the achievable resolution at the voltage level (log-likelihood ratios) for a given allowed collector current error ε. Exactly the same results are obtained if a MOS transistor in weak inversion instead of a BJT is considered. The resolution may be reduced by higher circuit temperatures, since U_{T} is directly proportional to the absolute temperature. From information-theoretic considerations we know that an equivalent internal resolution of the log-likelihood ratios of about 4 to 6 bits is often sufficient for Turbo decoding applications [132]. For the external channel information, even a resolution of 3 to 4 bits are generally sufficient for almost negligible degradations in the BER characteristic [133, 134]. Thus, matching problems resulting in current errors should not fatally corrupt the overall circuit behaviour, even if small devices are used in the circuits.

ε	δV_{BE}	resolution @ DR = 4 dec $\Delta V_{\text{BE}} = 240\,\text{mV}$	resolution @ DR = 5 dec $\Delta V_{\text{BE}} = 300\,\text{mV}$
1%	0.258 mV	9.9 bits	10.2 bits
5%	1.26 mV	7.6 bits	7.9 bits
10%	2.47 mV	6.6 bits	6.9 bits
25%	5.78 mV	5.4 bits	5.7 bits
50%	10.5 mV	4.5 bits	4.8 bits

Table 4.1 *Relation between dynamic range and resolution of a single bipolar transistor or a single MOS transistor in weak inversion.*

Analytical mismatch analysis of a complete decoder is intractable

Since the analytical mismatch analysis for a complete decoding system is intractable at the present time, extensive Monte-Carlo analyses of several output parameters have been carried out for various decoder implementations. An example of such a simulation record is given in Fig. 4.17 and Fig. 4.18 for the tail-biting trellis decoder described in Section 5.2. Bit 1 and bit 3 have been toggled during their transmission and thus need to be corrected. Therefore they have been chosen in this particular codeword configuration. The decoding delays, i.e., the time needed by the decoder to change the output bits to correctly sliced output state, are denoted DD1 for bit 1 and DD3 for bit 3. The statistical simulations show that the behaviour of the networks is indeed inherently robust. All simulations have shown that the

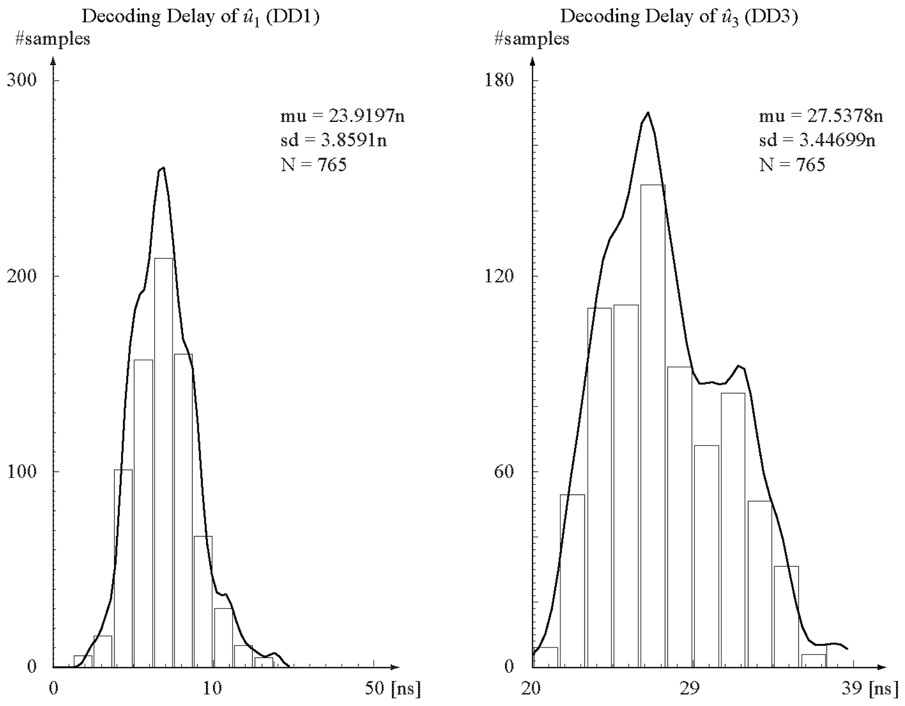
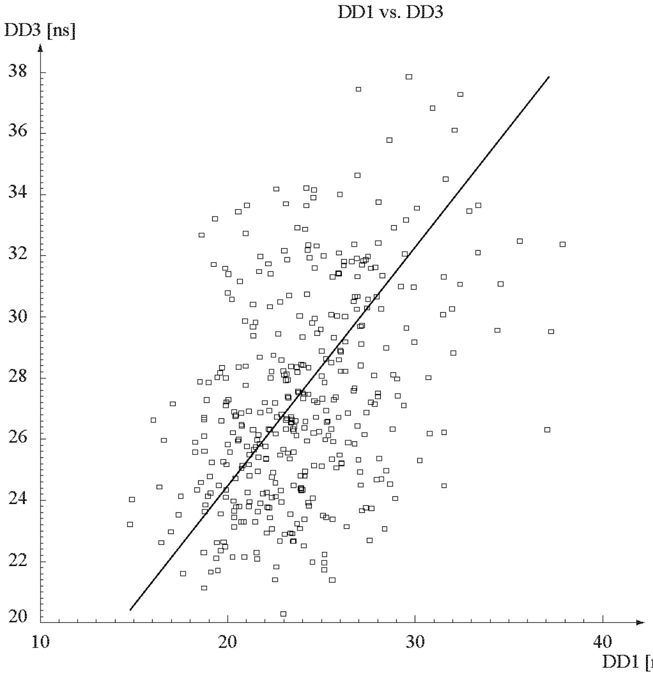


Figure 4.17 *Typical statistical distribution of the decoding delay of bit 1 (DD1) and bit 3 (DD3) affected by device mismatch.*

decoder converges to the correct ouput state. The robustness of the networks can even be increased by a proper description on the system level, e.g. by the choice of an adequate equation set which describes the code (see Section 5.5.2).

Monte-Carlo simulations show only a slight degradation in decoding speed

The only effects of the transistor mismatch observed in the simulation results is a *variation of the decoding time*. As shown in Fig. 4.17, the additive white Gaussian noise which models the variations in the device geometries and parameters in the Monte-Carlo simulation is clearly visible in the distribution of the decoding delays. This effect can be modelled to a first approximation by additional Gaussian noise added during the transmission of the symbols over the channel. For a given SNR in the received symbols, i.e., the *a priori* input probabilities, the decoder takes some amount of time to converge to the correct



The correlation between the decoding delay of bit 1 (DD1) and bit 3 (DD3) affected by device mismatch. **Figure 4.18**

output state. The smaller the SNR is, the longer it takes the decoder to correct the error bits. Taking the mismatch effects into account, the SNR is slightly modified according to the simple model. Since the changes of the device parameters are stochastic, the decoding time gets shorter or longer, depending on the applied codeword (Fig. 4.17). As is shown in Fig. 4.18, the correlation between the decoding delay variations is not very strong (correlation *r* below 0.5). Thus it may be assumed that the transistor variations are independent.

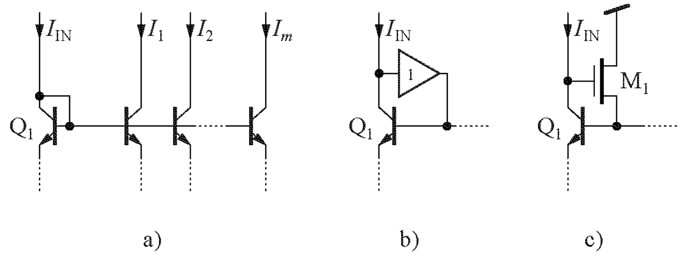


Figure 4.19 *Base current compensation for the fundemental multiplier matrix circuit: a) initial input connection, b) with an ideal voltage unity gain buffer, c) simple transistor implementation with one MOS transistor (source follower circuit).*

4.4.2 Finite Current Gain

The sum of all base currents may exceed input current in BiCMOS

A second implementation issue, which can seriously affect the performance of the proposed circuits, is the finite current gain of bipolar transistors. Compared to the almost infinite input resistance of MOS transistors, BJTs in a BiCMOS technology exhibit a current gain β of a few tens or hundreds and thus a much smaller input resistance than the MOS counterparts. In a BiCMOS implementation, each y-input to the core circuit has to drive $m+1$ bases of the $n \times m$ transistor array. If we consider trellis sections with two or four states (less than ten states in general), this would not affect the overall performance too much. But with $\beta = 100$ and a trellis section of a thousand states, which is quite usual in channel equalization applications, we would need 10 times the input current to only drive the bases of the transistors. This is clearly not possible without additional circuitry. Note that even in the 16-state case we encounter a systematic loss of about 16% in precision compared to the ideal case. Therefore we need a mechanism to compensate this loss. The basic diode-connected BJT Q_1 of Fig. 4.19 a) has to be replaced with a buffered version as shown in Fig. 4.19 b). The simplest circuit to implement the unity gain voltage buffer is the source follower circuit in Fig. 4.19 c). The design of the voltage buffer is a trade-off between speed and voltage overhead introduced by the additional gate-source voltage of transistor M_1.

Finite Output Resistance 4.4.3

Finite output resistances may even compensate other problems

The impact of the finite output resistance of both MOS transistors and BJTs is far lower than expected. As has been observed in transient simulations of a complete decoder, the finite output resistance even helps to partially compensate the finite current gain of the BJTs. Once again, the highly connected structure of the analog decoding networks helps to recover from implementation non-idealities. In CMOS versions of the circuits, however, the finite output resistance overestimates the desired output probabilities, i.e., it compresses the probability ratio and thus reduces the log-likelihood ratios. In case of very clear differences, this is not a problem anyway, since decisions are made easily under these circumstances. On the other hand, if the probabilities are almost equal, then the output probability ratios are far less disturbed by finite output resistances, since the drain-source voltages are at almost the same levels. In summary, the finite output resistance of both BJT and MOS transistors is not a big issue. On the contrary, it may even help to compensate other effects in the circuits of large networks.

Use cascode circuits for higher-precision outputs

If high-precision output values are required, standard techniques for improving the output resistance such as cascode structures can be applied [135–137]. But this extra transistor circuitry has to be replicated many times in order to improve the whole network, and the increase in chip area may be unacceptable. It may be more economical in terms of chip area to spend only a few extra transistors for compensating the finite current gain as seen in the previous subsection.

Thermal Effects 4.4.4

An important performance issue of analog circuits is the thermal behaviour of different devices. The controlling base-emitter voltage of BJTs, for example, is known to show a -2mV/K temperature gradient (see e.g. [138]). Similar problems arise in weakly inverted MOS transistors where a temperature dependence of the drain current with respect to the source voltage can be observed [109]. This may cause severe problems if thermal matching cannot be guaranteed. We will discuss two different cases, namely the effects of temperature on probability-based and on log-likelihood-ratio-based analog networks.

Temperature Dependence in Probability-Based Networks

Three common thermal situations

In the following, three thermal situations will be discussed: a) a uniform temperature distribution over the whole chip, b) a temperature gradient over the chip but a uniform temperature distribution within one building block, and c) a temperature gradient within one of the building blocks. They may affect the temperature behaviour of probability based networks, i.e., the basic information exchanged between the basic building blocks are current vectors representing discrete probability distributions.

a) A uniform temperature distribution affects all devices in the same manner. Since the information travelling through the decoding network has the form of *current ratios*, the temperature, which mainly affects the thermal voltage $U_\mathrm{T} = kT/q$, has no effect. As an example, consider the current ratio of two perfectly matched bipolar transistors in a current mirror. No influence of temperature on the current ratio can be observed in this case. Reasoning with translinear circuit principles [105], we see immediately that this simple example completely describes the temperature tracking behaviour within one circuit module of our considered networks.
b) If the temperature within one module is approximately constant, but the entire chip is affected by a slight temperature gradient, the solution is identical to the one in a), by following the same argumentation.
c) A temperature gradient within a circuit module is the most difficult case to handle. Depending on the nature of the temperature gradient, and depending on the layout chosen for the implementation, different values for the temperature tracking error are obtained. To estimate the maximum temperature difference within one module, we can make the following calculation. According to the thermal conductivity law, the temperature difference inside a slice of semiconductor with area A and thickness s is given by

$$\Delta T = \frac{Ps}{\lambda A}, \tag{4.13}$$

with the dissipated power P and the thermal conductivity coefficent λ.

Applying (4.13) to the implemented decoder modules with $A = 100 \cdot 200\ \mu\mathrm{m}^2$, $s = 100\ \mu\mathrm{m}$, $P = 3$ mW, and

$\lambda = 1.5$ W/(m·K) for a silicon substrate, the maximum temperature difference will be $\Delta T = 0.1$ K. For such a small value, the error introduced by thermal mismatch is negligible compared to the device matching error:

$$\varepsilon_{\text{rel}}(\Delta T) = -1 + \exp\left(\frac{V_{\text{BE}}}{\frac{kT_{\text{nom}}}{q}}\right)^{-\frac{\Delta T}{T_{\text{nom}}+\Delta T}} \tag{4.14}$$

At room temperature (300 K) and $V_{\text{BE}} = 0.7$ V, the error is approx. 1%. Although this type of error is not strictly random, it can well be approximated by additional noise, similarly as in the device-matching case.

It is most likely that we will encounter a mix of cases b) and c) in our circuit chips, because the body of the package induces a uniform temperature over the entire integrated circuit. Under these assumptions, thermal effects will not have a noticeable impact on the function of the probability-propagation networks.

Temperature Dependence in Log-Likelihood-Ratio Based Networks

The log-likelihood ratio representation appears as temperature-dependent voltages...

The second case, where the information exchanged between the different circuit modules is represented by voltages corresponding to log-likelihoods (or voltage differences for the log-likelihood ratios as introduced in Section 4.1.4), is more problematic. As we have noticed in analyzing (4.2), the drain current of a weakly inverted MOS transistor is temperature dependent through the thermal voltage U_{T} and the slope factor n. For the following temperature analysis, we can rewrite (4.2) as

$$I_{\text{D}} = I_0(T)e^{(V_{\text{G}}-nV_{\text{S}})/U_{\text{T}}} = I_0(T)e^{(\alpha V_{\text{G}}-\beta V_{\text{S}})/U_{\text{T}}}, \tag{4.15}$$

where $\alpha(T)$ and $\beta(T)$ are arbitrary parameters, introduced for the sake of simplicity in the following mathematical analysis. If we use this notation of the drain currents, we can describe the voltages $V_{y,j}$, no matter whether implicitly present at inputs of the circuit of Fig. 4.2 or applied to the inputs of the circuit of

Fig. 4.5, by

$$V_{y,j} = \frac{1}{\alpha}\left(U_\mathrm{T}\ln\frac{I_{y,j}}{I_0} + \beta V_\mathrm{ref}\right) \tag{4.16}$$

$$= \frac{1}{\alpha}\left(U_\mathrm{T}\ln p_Y(y_j) + U_\mathrm{T}\ln\frac{I_y}{I_0} + \beta V_\mathrm{ref}\right) \tag{4.17}$$

$$= \frac{1}{\alpha}U_\mathrm{T}\ln p_Y(y_j) + V_\mathrm{os}(T), \tag{4.18}$$

where V_ref is the source reference potential of the diode-connected transistors at the input of the transistor matrix and $V_\mathrm{os}(T)$ is a temperature-dependent offset voltage which can be freely chosen (corresponding to the free choice of the sum current I_y in (4.18)).

... hence do not consider voltage-mode-connections for large probability-propagation networks

Now assume a voltage mode connection situation, i.e., the probabilities are transformed to log-likelihoods by means of a logarithm circuit at the output of the circuit modules, and further assume two different temperatures T_1 and T_2 at the output and the input of the two modules to connect. Then the temperature dependent offset term cancels smoothly if we use a differential signal representation, i.e., assuming log-likelihood ratios, But unfortunately we observe that the temperature dependent term U_T/α introduces a non-recoverable error in the core transistor matrix, although the differential data representation would allow error-free signal transmission in theory. The voltage error proportional to the absolute temperature T is even amplified on the drain-current level by the inherent exponential VI-characteristic of the transistor. Hence, the main conclusion from the above reasoning is that it is not advisable to use voltage-mode connections for global connections on a large analog network, where the temperature cannot be guaranteed the same on the whole circuit chip. However, for very small networks, such as for example the one presented by Moerz *et al.* [70], the temperature effects seem to be negligible. The same temperature problem also affects the BJT case, where we observe the famous -2 mV/K temperature gradient on the base-emitter voltage of the transistor.

Other Implementation Issues 4.4.5

Topology-Induced Problems

Biasing large analog networks requires careful design and layout

Besides issues directly related to individual circuit devices, we also encounter *topology-induced problems*. By this we mean that, for example, biasing a large probability propagation network may cause severe problems since no local matching can be guaranteed for a distributed biasing networks (e.g. distributed current mirrors for the current sources needed in each cell). Since the geometrical dimensions may get very large, additional effects such as non-zero resistance of long metal wires show up. This may affect signal tracks as well as power-supply lines. It must be kept in mind during the design phase that a distributed bias-network implemented with BJTs draws a considerable amount of base current which causes large voltage drops on long metal tracks. In the extreme, these voltage drops may prevent the whole network from working correctly. Comparable problems arise in digital circuits for the clock distribution. There the solution is to balance the load in different branches of a clock distribution tree instead of having one large single track. Adapted to our networks, this would mean using local repeaters for the biasing circuits. Errors introduced by these circuits are not critical, since all calculations rely on relative signal strength.

Construction of Large Analog Networks

Large analog probability propagation networks are built by construction

A second, more general implementation issue is how to construct large analog computational networks. Up to a few hundred transistors, an analog system may be drawn very easily if a hierarchic design approach is chosen. But imagine a large factor graph of several hundreds or thousands of individual nodes. How do you want to make sure that, after a long day of drawing interconnection lines between the individual building blocks, you do not make drawing mistakes? It is certainly a good idea not to rely on your own drawing capability if a schematic can be generated by a computer program. In the context of coding, the structure of a block code is for example described by its parity-check matrix **H**. This matrix may serve as basis for many different design steps: the code's performance may be evaluated by using the matrix in a simulation program, but it may also serve as the basis for our schematic generation program. This approach has been demonstrated in the decoder example of Section 5.3. In general, it will not be possible to design a large analog network first-time-right without computer aided design. By this we subsume not only computer-aided drawing (CAD), but also computer-aided engineering (CAE), which includes much more than only the sketch of schematics.

Testability

Design analog networks for testablity

Another big issue of such large networks is *testability*. How can we guarantee that a circuit leaving the waferfab works as expected? Rudimentary tests such as checking the supply current or verifiying individual test blocks are generally not sufficent to guarantee the overall functionality. Testing large digital circuits is much easier than doing the same thing for analog networks. Boundary scan and JTAG test access ports are commonly used today for looking inside the working digital circuit. They mostly rely on digital registers that can be adressed and read out serially on certain circuit chip pins. Testing large analog systems is much more difficult. The measuring circuitry should not modifiy (by creating additional loads on the interesting nodes) the overall behaviour. Additionally, the resolution of measured values should be better than the resolution of the actual circuit under test. This means that the circuits for measuring have to be more precise, and are thus in general also more complex and space consuming, than the circuit to verify. Even if a measurement circuit can be shared among many circuit nodes to test, it may add a considerable overhead to the overall network. So it would be desired that the circuit's functionality could be guaranteed by design. One approach to this may lie in an information-theoretic approach that tries to quantify the impact of individual error sources to an overall probability propagation network. Unfortunately, we did not have the time yet to investigate such an approach, but it will be subject to future research.

Chapter 5

Decoder Examples

Five decoder examples are discussed on various levels of completion

In this chapter, we describe several decoder designs. The decoder examples are discussed on various levels of completion. First we will discuss an implementation of a very simple trellis code using discrete bipolar transistors. The result of this effort is a demonstration unit giving static output results. The second example consists of the complete VLSI implementation of a short tail-biting trellis code. For this example we are able to present dynamic measurement results. The third example has actually not been tested, since bad bonding contacts made by a subcontractor prevented us from measuring the chip. The two examples at the end represent design studies that we will use as the basis for further projects at our lab. Note that large schematics are placed in the appendix at the end of this chapter to simplify the reading of the text.

Decoder for a Simple Trellis Code 5.1

Code Description 5.1.1

Simple binary [5,2,3] block code described as trellis diagram...

As a first complete decoder example, we examine a decoder for a binary [5,2,3] block code, i.e., 2 databits u_i are encoded into codewords $\mathbf{x}$ of length 5 and with Hamming distances between different codewords being at least 3. The code consists only of the four codewords $[\underline{0},0,\underline{0},0,0]$, $[\underline{0},0,\underline{1},1,1]$, $[\underline{1},1,\underline{0},1,1]$, $[\underline{1},1,\underline{1},0,0]$. The first and third bit (underlined), x_1 and x_3 respectively, are considered as information bits. The considered [5,2,3] block code is *systematic*, since uncoded information bits are also present in the codeword. The code can be represented by a 5-section trellis diagram as given in Fig. 5.1. Hence, a valid codeword is indicated by one of the four paths through that trellis diagram.

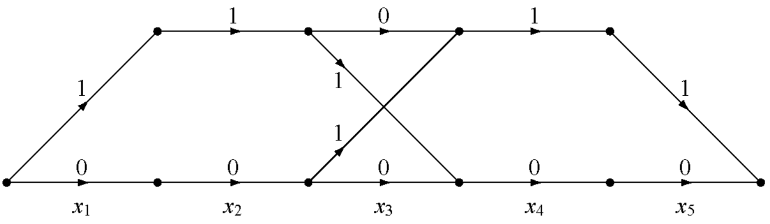


Figure 5.1 *A simple trellis code consisting only of 4 codewords.*

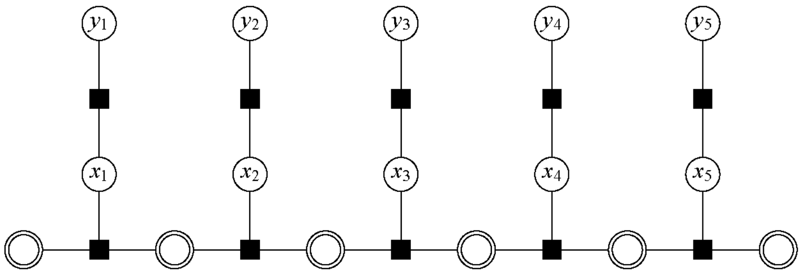


Figure 5.2 *Factor-graph representation of the binary [5,2,3] trellis code.*

...and as factor graph

Corresponding to the trellis diagram of Fig. 5.1, we can directly draw the augmented factor graph as shown in Fig. 5.2. It directly describes the topology of the decoder. Each function node in the lower part of the factor graph (black rectangle) corresponds to one trellis section of the code. State variables are drawn as double circles, whereas the observable variables are shown as single circles.

Block diagram of the analog decoder circuit

The block diagram of the decoder network is shown in Fig. 5.3 with trellis modules as in Fig. 5.4. It is a direct implementation of the forward-backward algorithm [103] which is a special adaptation of the general sum-product algorithm to codes described by trellis diagrams. In principle we may draw a corresponding building block for each function node in the block diagram. Since the building blocks are only uni-directional, we draw them separately for both directions: the upper row of the decoding network implements the forward part and the middle row the backward part of the forward-backward algorithm. The soft-output decoder network of Fig. 5.3 computes the *a posteriori* probabilities of the information bits only. Hard-decisions of the information bits can easily be formed from

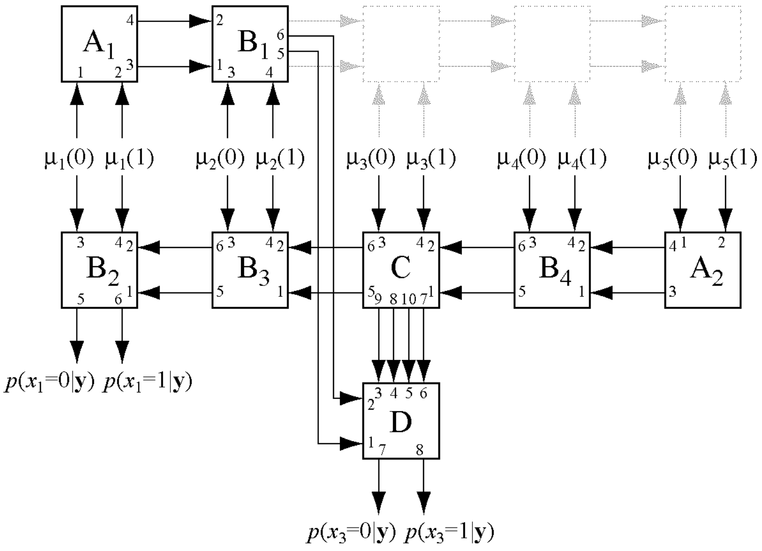


Figure 5.3 *A decoder for the code of Fig. 5.1. The trellis implementation of modules A-D are given in Fig. 5.4.*

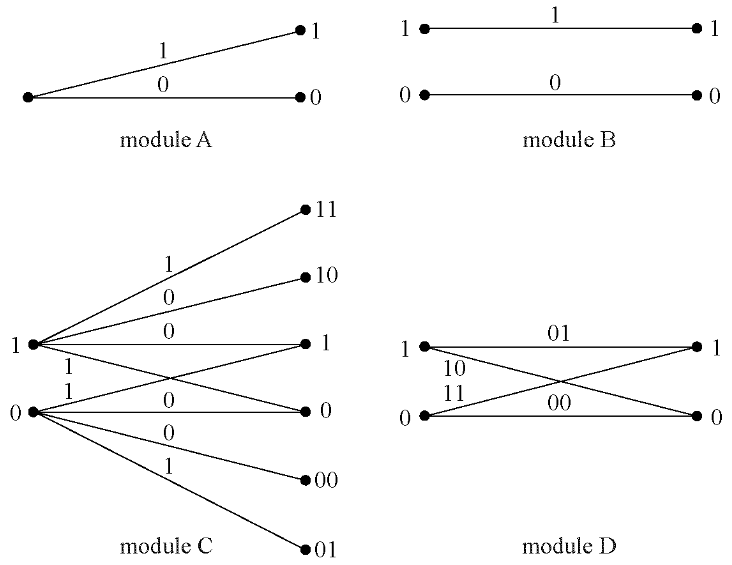


Figure 5.4 *Trellis modules for the decoder network of Fig. 5.3.*

these *a posteriori* probabilities by using a bit slicer. The dashed boxes in Fig. 5.3 correspond to those computations in the general forward-backward algorithm which are not used in this example, since they do not contribute to the final result.

Messages that are passed in the decoder are discussed individually

In the following paragraphs, all messages passed from block to block in the decoder (or from node to node in the factor graph) will be deduced step by step to point out the underlying algorithm. We assume that a codeword is transmitted over a BSC with transition probability $p(y|x)$. Let $\mathbf{y} = [y_1, \ldots, y_5]$ be the received channel output. Furthermore, we assume that the *a priori* probability is uniform over the codewords. Introducing the abbreviation $\mu_i(b) \triangleq p(y_i|x_i{=}b)$, the *a posteriori* probability of a codeword $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$ can be written as

$$p(\mathbf{x}|\mathbf{y}) = \gamma\, \mu_1(x_1)\, \mu_2(x_2)\, \mu_3(x_3)\, \mu_4(x_4)\, \mu_5(x_5), \qquad (5.1)$$

where γ is a scale factor that does not depend on the codeword. In our example, the *a posteriori* probabilities of the information bits are thus given by

$$\begin{aligned} p(x_1{=}0\,|\,\mathbf{y}) &= \gamma\cdot\big(\mu_1(0)\,\mu_2(0)\,\mu_3(0)\,\mu_4(0)\,\mu_5(0) \\ &\qquad + \mu_1(0)\,\mu_2(0)\,\mu_3(1)\,\mu_4(1)\,\mu_5(1)\big) \qquad (5.2) \\ p(x_1{=}1\,|\,\mathbf{y}) &= \gamma\cdot\big(\mu_1(1)\,\mu_2(1)\,\mu_3(0)\,\mu_4(1)\,\mu_5(1) \\ &\qquad + \mu_1(1)\,\mu_2(1)\,\mu_3(1)\,\mu_4(0)\,\mu_5(0)\big) \qquad (5.3) \\ p(x_3{=}0\,|\,\mathbf{y}) &= \gamma\cdot\big(\mu_1(0)\,\mu_2(0)\,\mu_3(0)\,\mu_4(0)\,\mu_5(0) \\ &\qquad + \mu_1(1)\,\mu_2(1)\,\mu_3(0)\,\mu_4(1)\,\mu_5(1)\big) \qquad (5.4) \\ p(x_3{=}1\,|\,\mathbf{y}) &= \gamma\cdot\big(\mu_1(0)\,\mu_2(0)\,\mu_3(1)\,\mu_4(1)\,\mu_5(1) \\ &\qquad + \mu_1(1)\,\mu_2(1)\,\mu_3(1)\,\mu_4(0)\,\mu_5(0)\big). \qquad (5.5) \end{aligned}$$

These quantities, up to the scale factor γ, are computed by the decoding network of Fig. 5.3.

Backward computations of the forward-backward algorithm

We begin the detailed description with the middle row (the backwards computation) of Fig. 5.3. Module A_2 simply scales the input vector $[\mu_5(0), \mu_5(1)]^T$ to some fixed level. For the remaining part of this section, we shall not distinguish between differently scaled versions of a vector. Then module B_4 computes the vector $[\mu_4(0)\mu_5(0), \mu_4(1)\mu_5(1)]^T$. Module C computes the vector

$$\begin{bmatrix} \mu_3(0)\,\mu_4(0)\,\mu_5(0) \\ \mu_3(1)\,\mu_4(0)\,\mu_5(0) \\ \mu_3(0)\,\mu_4(1)\,\mu_5(1) \\ \mu_3(1)\,\mu_4(1)\,\mu_5(1) \end{bmatrix}, \qquad (5.6)$$

and from that the vector

$$\begin{bmatrix} \mu_3(0)\,\mu_4(0)\,\mu_5(0) + \mu_3(1)\,\mu_4(1)\,\mu_5(1) \\ \mu_3(1)\,\mu_4(0)\,\mu_5(0) + \mu_3(0)\,\mu_4(1)\,\mu_5(1) \end{bmatrix}.$$

From the latter, the module B_3 computes the vector

$$\begin{bmatrix} \mu_2(0)\big(\mu_3(0)\,\mu_4(0)\,\mu_5(0) + \mu_3(1)\,\mu_4(1)\,\mu_5(1)\big) \\ \mu_2(1)\big(\mu_3(1)\,\mu_4(0)\,\mu_5(0) + \mu_3(0)\,\mu_4(1)\,\mu_5(1)\big) \end{bmatrix}.$$

From that, the module B_2 computes

$$\begin{bmatrix} \mu_1(0)\,\mu_2(0)\big(\mu_3(0)\,\mu_4(0)\,\mu_5(0) + \mu_3(1)\,\mu_4(1)\,\mu_5(1)\big) \\ \mu_1(1)\,\mu_2(1)\big(\mu_3(1)\,\mu_4(0)\,\mu_5(0) + \mu_3(0)\,\mu_4(1)\,\mu_5(1)\big) \end{bmatrix},$$

which is proportional to $[p(x_1{=}0\,|\,\mathbf{y}), p(x_1{=}1\,|\,\mathbf{y})]^T$.

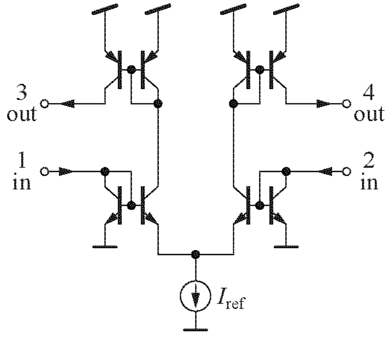


Figure 5.5 *Schematic of module A: a simple scaling circuit.*

Forward computations of the forward-backward algorithm

In the upper row (the forward computation) of Fig. 5.3, the module A_1 scales the input vector $[\mu_1(0), \mu_1(1)]^T$ to some fixed level, and the module B_1 computes $[\mu_1(0)\mu_2(0), \mu_1(1)\mu_2(1)]^T$. In the bottom row (combination), the module D computes from B_1's output and (5.6) the vector

$$\begin{bmatrix} \mu_1(0)\mu_2(0)\mu_3(0)\mu_4(0)\mu_5(0) + \mu_1(1)\mu_2(1)\mu_3(0)\mu_4(1)\mu_5(1) \\ \mu_1(0)\mu_2(0)\mu_3(1)\mu_4(1)\mu_5(1) + \mu_1(1)\mu_2(1)\mu_3(1)\mu_4(0)\mu_5(0) \end{bmatrix},$$

which finally is proportional to $[p(x_3{=}0|\mathbf{y}), p(x_3{=}1|\mathbf{y})]^T$.

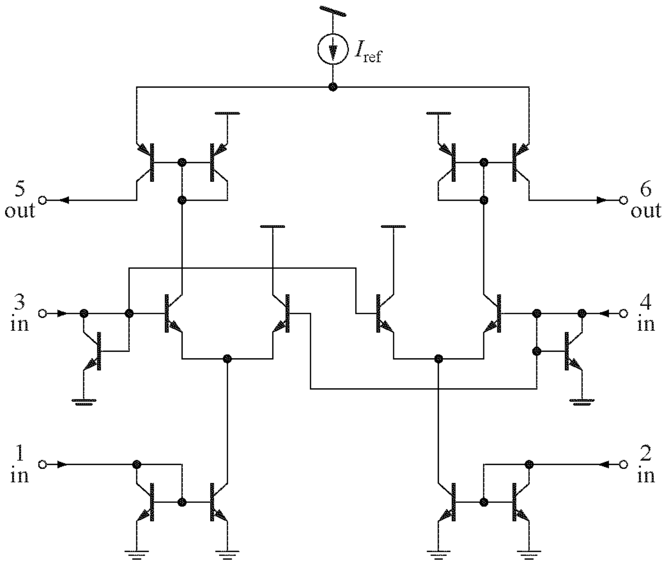
5.1.2 Implementation Using Discrete Transistors

Circuits implemented with discrete BJTs

The decoder of Fig. 5.3 was implemented on a printed circuit board (PCB) using discrete bipolar transistors. We selected the CA3096 transistor array [139] for both NPN and PNP transistors. On top of each generic module, an output current scaling circuit using PNP transistors was added to prevent the probablity signals fading into the noise floor. Fig. 5.5 to Fig. 5.8 show the individual circuit schematics. The numbers near the input pins and output pins correspond directly to the ones found in the block diagram of Fig. 5.3.

Demonstration purpose of the decoder circuit

The purpose of this simple decoder was purely demonstrational. Thus LED bargraph displays were added to monitor both input and output values in a linear scale and a logarithmic scale. Additionally, easy-to-use potentiometric input-probability value



Schematic of module B*: the equal gate.* **Figure 5.6**

controls were used. They allow to preset the probabilities from 0.5 to 0.999 and the sign, i.e., the choice of either a '1' or a '0', for each input value μ_i individually. A photograph of the demonstration unit built at our electronics laboratory is shown in Fig. 5.9.

Readout values are displayed by linear and a logarithmic LED bargraph display

The readout of the output values is purely static. No transient information can be obtained from the built demonstration unit. The comparison of the measured *a posteriori* probabilities with the theoretic values (using the calculations of the previous, introductory section) showed a remarkably close agreement, although no preselection of the individual bipolar transistors was carried out. Harris' transistor array CA3096 contains 3 NPN and 2 PNP BJTs. These transistors were partitioned automatically among the modules on the schematic by the packager of the PCB design software. Even if one had taken into account that two matched transistor pairs are present on each array, we could not have assured that all transistors to be matched in theory would be in the same package. For the assembly, the transistor arrays were just taken out of the box and soldered on the PCB. Hence, one would expect a very bad matching behaviour

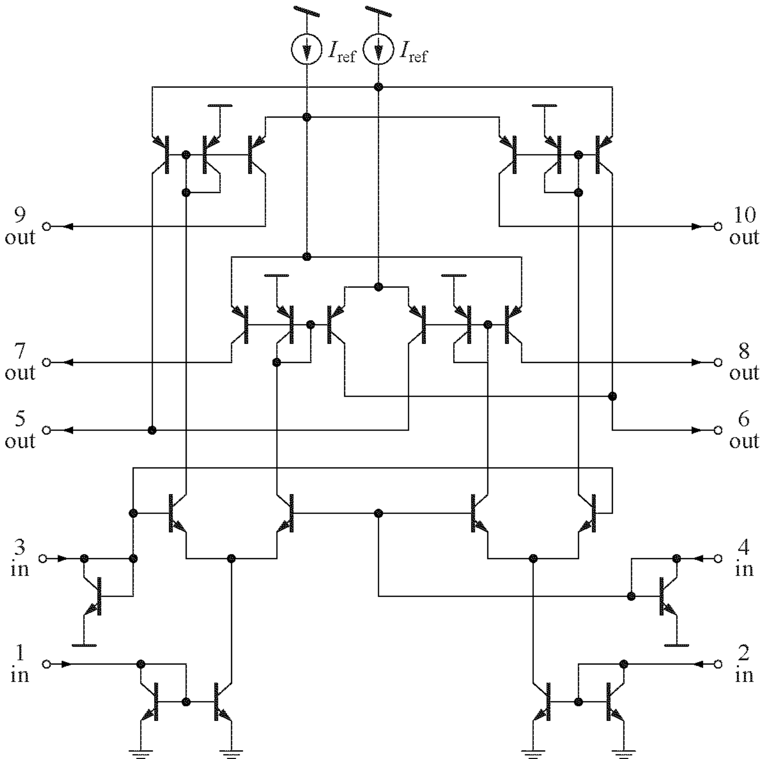


Figure 5.7 *Schematic of module* C*: butterfly connection with previous current duplication.*

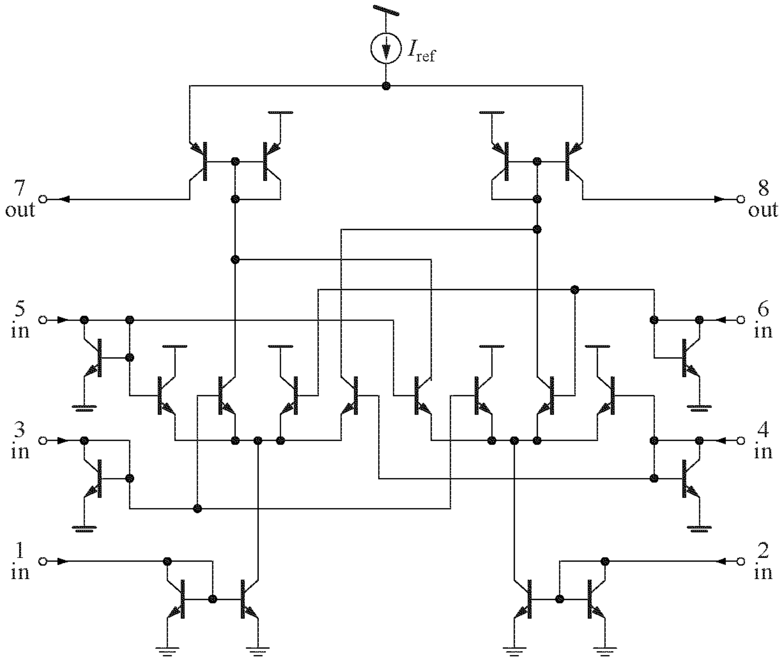


Figure 5.8 *Schematic of module* D*: combining forward and backward computations.*

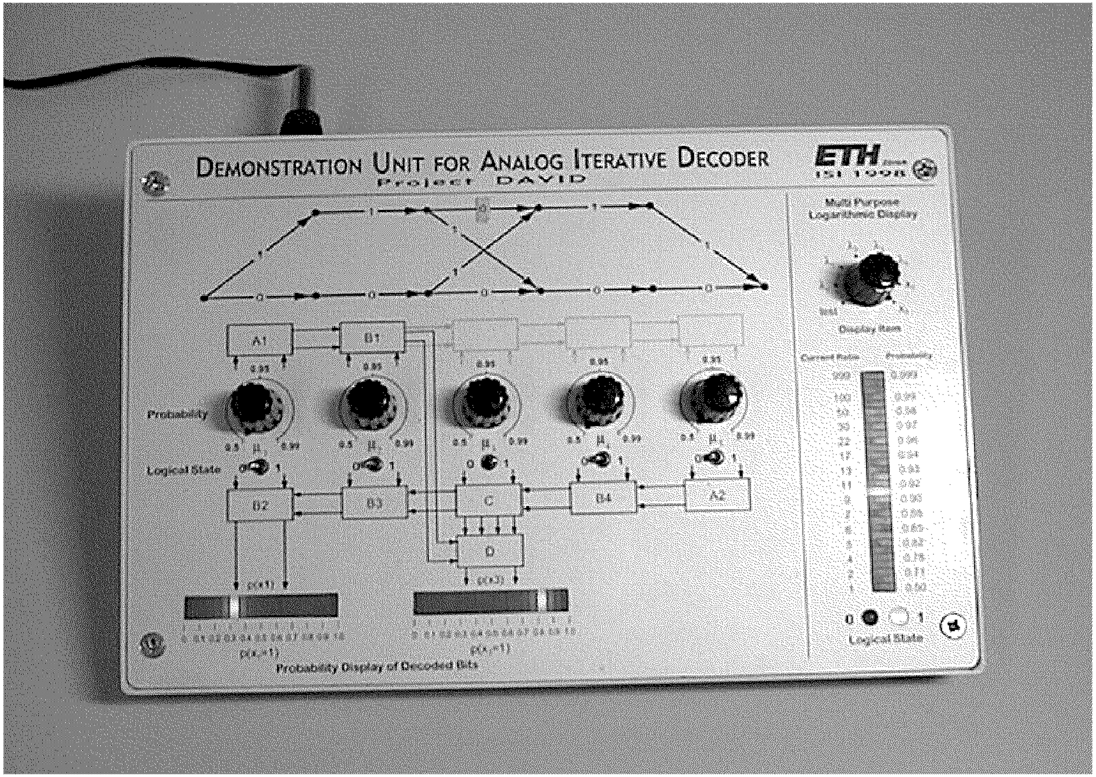

Demonstration Unit for Analog Iterative Decoder
Project DAVID
ETH
ISI 1998
A1
B1
Probability
Logical State
B2
B3
C
B4
A2
D
Probability Display of Decoded Bits
Multi Purpose Logarithmic Display
Display Item
Current Ratio
Probability
Logical State

Figure 5.9: *Demonstration unit of a MAP decoder for the [5,2,3] block code. The LED bargraphs on the bottom of the front panel represent the probabilities* $p(x_1 = 1|\mathbf{y})$ *and* $p(x_3 = 1|\mathbf{y})$ *on a linear scale. The display section on the right-hand side is a logarithmic-scale display of the selectable input and output values.*

of the overall system. Additionally, the transistors are on different substrates and thus may also experience different temperatures. Despite these harsh conditions, an overall static precision of about $\pm 5\,\%$ was measured using both the linear scale and the logarithmic scale outputs.

Decoder for a Tail-Biting Trellis Code 5.2

A first chip-level implementation

Our second example of a complete decoder was implemented on silicon using the 0.8 μm double-poly, double-metal BiCMOS process of AMS [140]. We discuss the code description of the tail-biting trellis code, the simulation results, the test setup, and the measurement results, and do a coarse comparison to an equivalent digital implementation.

General Description 5.2.1

Description of the Code

The tail-biting trellis code

The considered binary [18,9,5] code is a tail-biting trellis code as introduced in Section 2.1.5. The trellis diagram for the code consists of nine equal sections as the one shown in Fig. 5.10. First, these nine equal trellis sections are cascaded like an ordinary trellis. Then the outgoing states of the last section are identified with the starting states of the first section to form this closed structure. A valid codeword is a path that starts in an arbitrary state, goes through the entire trellis one time and ends in the same state.

The encoding of the dataword needs some special attention due to the tail-biting nature of the code. It can be carried out with the convolutional encoder of Fig. 5.11. First, the convolutional encoder is reset to the all-zeros state. Second, 9 information bits $u_1,\ldots,u_9$ are fed, one by one, into the convolutional encoder; in this process, 18 output bits (9 pairs) are generated, which we will refer to as $x'_1, x'_2, \ldots, x'_{18}$. Third, two dummy zero bits are fed into the convolutional encoder to drive it back to the all-zero state and thereby generating four extra output bits $x'_{19}, \ldots, x'_{22}$.

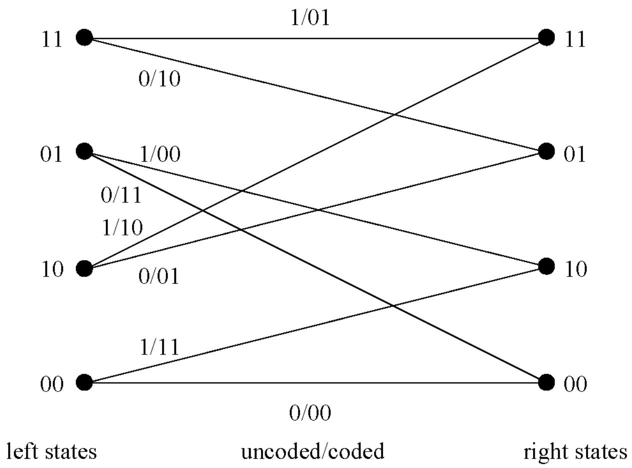


Figure 5.10 *One section of the binary [18,9,5] tail-biting trellis code.*

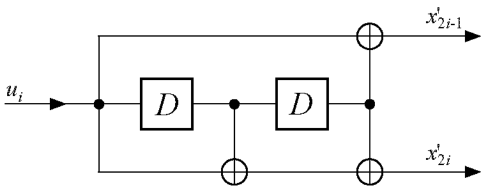


Figure 5.11 *A convolutional encoder for the binary [18,9,5] tail-biting trellis code.*

Finally, the codeword $\mathbf{x} = [x_1, x_2, \ldots, x_{18}]$ is formed by the rule

$$x_i = \begin{cases} x'_i \oplus x'_{i+18} & i = 1, \ldots, 4; \\ x'_i & i = 5, \ldots, 18. \end{cases} \tag{5.7}$$

Note that we could also encode the information bits differently: first we initialize the encoder state with the two last information bits, then we apply successively information bits for $k = 18$ time-steps. The output of the encoder is then directly the desired codeword. However, a disadvantage of this method is the necessary initialization of the encoder, but one needs less clock steps for the encoding process.

For any closed path in the tail-biting trellis, both the information bits and the corresponding coded bits can be read off the edge labels along the path.

Factor Graph and Block Diagram of the Decoder

Having the examples from Section 3.2 in mind, sketching the augmented factor graph of Fig. 5.12 for the tail-biting trellis code is straightforward. The tail-biting nature of the code is clearly visible in this drawing.

The factor graph of the tail-biting trellis code has a ring structure

In the following, it is assumed that the codewords are transmitted over a memoryless channel with transition probabilities $p(y|x)$, $x \in \{0, 1\}$. A complete decoding network for this code is given in Fig. 5.13, with the computation modules defined in Fig. 5.14. Each signal line in Fig. 5.13 represents a whole probability mass function. The inputs and outputs of the decoder are probability mass functions defined on a two-letter alphabet, whereas the remaining signals represent probability mass functions that are defined on a four-letter alphabet. Therefore, we have drawn the latter with heavier lines to make a clear distinction.

Data transmission is assumed on a BSC

The inputs to the decoder are the probabilities $p(y_i|0)$ and $p(y_i|1)$, $i = 1, \ldots, 18$, where $\mathbf{y} = [y_1, \ldots, y_{18}]$ is the channel output data. The outputs of the decoder are approximate *a posteriori* probabilities $\tilde{p}(u_i|\mathbf{y})$ for all information bits u_i. A final decoding decision may be obtained by comparing $\tilde{p}(u_i = 1|\mathbf{y})$ with $\tilde{p}(u_i = 0|\mathbf{y})$.

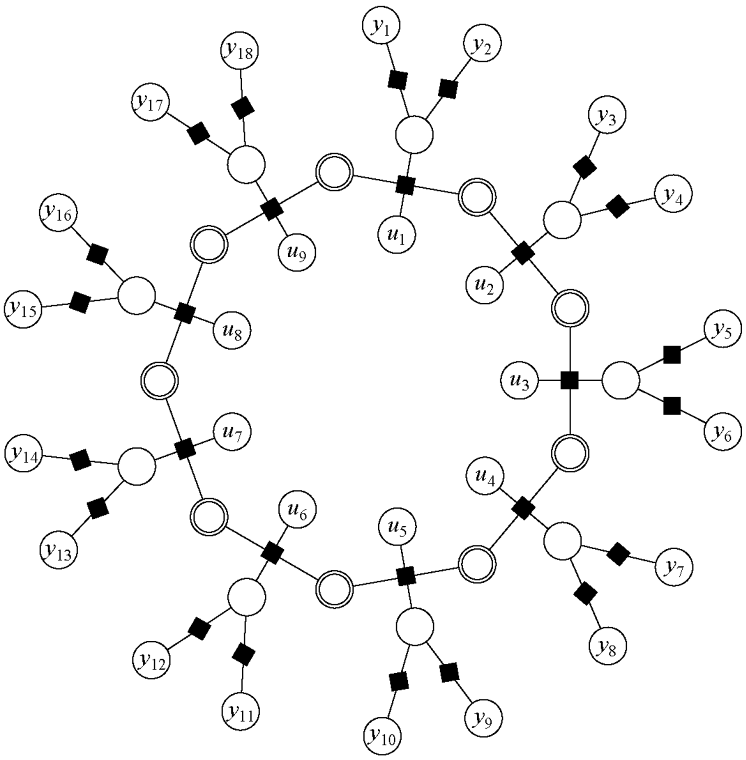


Figure 5.12 *Factor graph representation of the binary [18,9,5] tail-biting trellis code.*

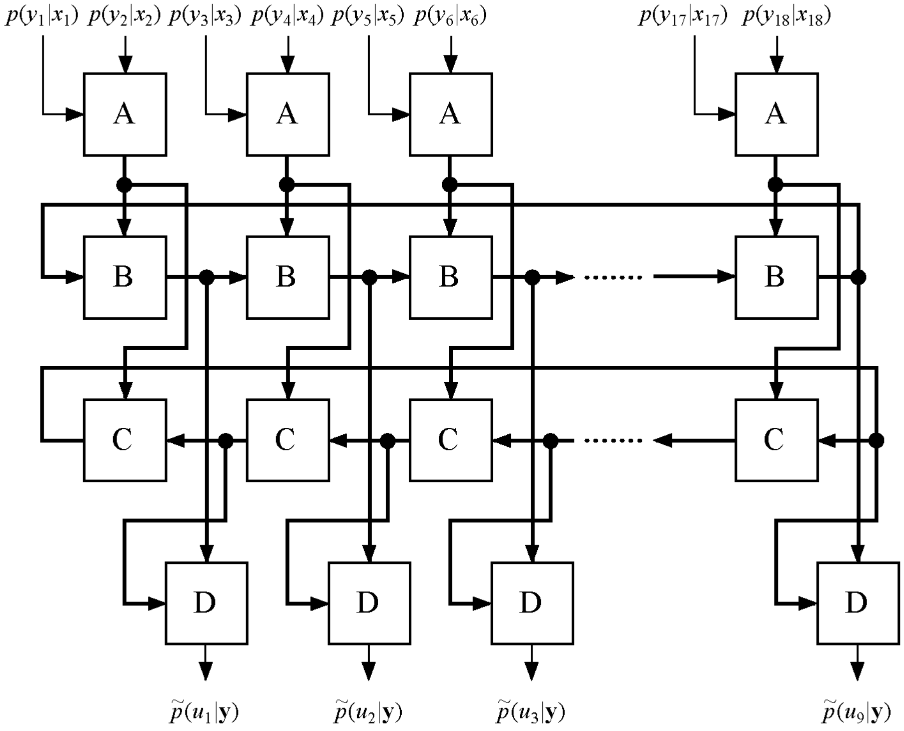


Figure 5.13 *Decoding network for the binary [18,9,5] tail-biting trellis code.*

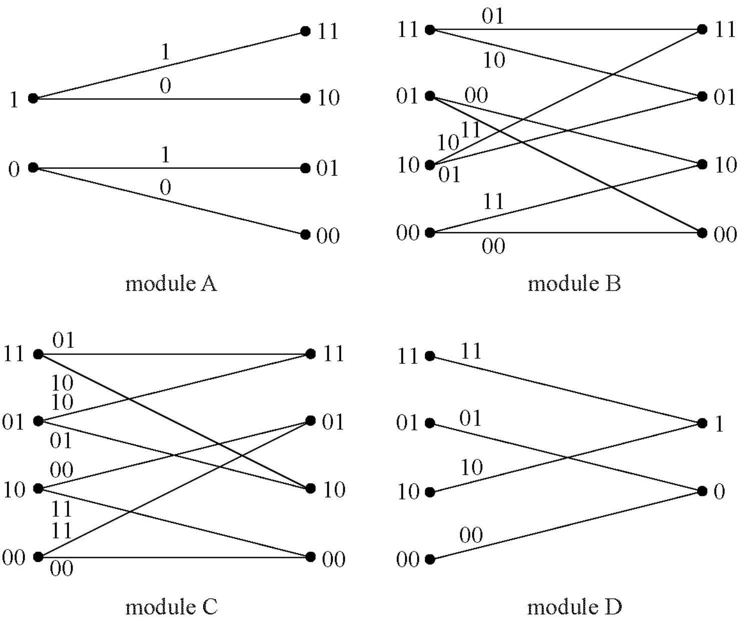


Figure 5.14 *Trellis representation of the binary indicator functions to be implemented in the decoder modules for the binary [18,9,5] tail-biting trellis code.*

Again, this decoder network is a direct implementation of the forward-backward algorithm [103] (adapted to a tail-biting trellis). The type-B modules in Fig. 5.13 perform the "forward" computation and the type-C modules perform the "backward" computation on the tail-biting trellis. The type-A modules precompute the branch metrics and the type-D modules compute the final probabilities for each information bit.

A direct implementation of the forward-backward algorithm

The outputs of the network are only *approximate a posteriori* probabilities, because the forward-backward algorithm computes *exact a posteriori* probabilities only for ordinary (not tail-biting) trellis codes. On the level of factor graphs, we should recall that the sum-product algorithm produces only exact *a posteriori* output probabilities if the code is a cycle-free factor graph, i.e., if it has the form of a tree. The approximation need not be very good, since finally only the sign of the difference $\tilde{p}(u_i = 1|\mathbf{y}) - \tilde{p}(u_i = 0|\mathbf{y})$ matters.

Sum-product algorithm on factor graphs with cycles delivers only approximate probabilities

Note that this decoding network contains two loops that correspond to the forward and the backward computations. In general, networks with multiple loops may not always converge to a stable state. However, networks as in Fig. 5.13 with no *interacting* loops are guaranteed to converge unless some of the input probabilities are zero [141].

Always convergence if only one loop exists in the factor graph

Circuit Design 5.2.2

The decoder network for the binary [18,9,5] tail-biting trellis code was designed as an analog network with about 940 BJTs and 650 pMOS transistors in the AMS 0.8 μm double-metal BiCMOS technology [140]. Fig. 5.15 and Fig. 5.51 to Fig. 5.53 show the circuit implementations of the modules of type A to D.

Circuits designed with AMS 0.8 μm BiCMOS process

Representatively, we discuss the type-B module: the core transistors for the multiplication part of the circuit are minimum-size BJTs ($3 \times 0.8\,\mu\text{m}^2$). Each BJT sits in its own well, which has to be separated from any other well by 7 μm on each side. This will create much unused active area on the chip layout which can be used for routing the interconnects. In fact, the size of the BJTs are the main factor determining the area of a building block.

Sizing of the bipolar transistors

Sizing of the pMOS transistors

The remaining transistors, located in the current mirrors on the top of the circuit of Fig. 5.51, are pMOS FETs with almost minimal size ($12 \times 1.6\,\mu\text{m}^2$). They were designed for maximum speed in a 5 V design and $V_{\text{Dsat}} = 2\,\text{V}$. Hence, the current mirrors operate in the strong-inversion region for the nominal currents. The current error was roughly estimated according to the Pelgrom formula [142]. Hand calculations showed that the standard deviation will then be far below 1 % at the nominal current of $200\,\mu\text{A}$, but the error will go up to about 33 % for very low currents. This is a minor problem, since the determining factor will be the error at equal current levels for all the elements of the probability distribution. Under these conditions, the matching is still reasonable if the discrete probability distributions have not too many elements, i.e. less than 10 elements. If this does not hold, the circuits have to be designed more carefully for the desired matching in the uniformly distributed situation.

Other modules are built similarly

The other modules in Fig. 5.14 shown in Fig. 5.51 to Fig. 5.53 are built similarly. For each edge in the trellis representation, a transistor can be identified in the middle row of the circuit diagram. Additionally, dummy transistors are introduced such that each state has the same number and the same types of outgoing branches.

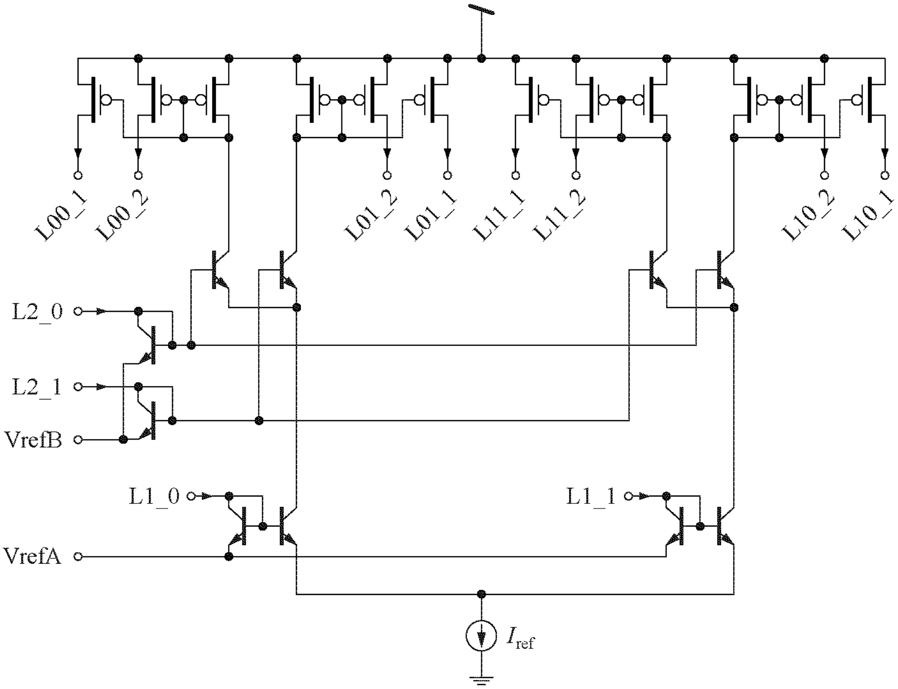
Back-to-back layout

The layout of the individual building blocks was made with the aim to allow direct back-to-back connections on every edge. By doing this, only the connections to close the tail-biting structure have to be drawn. Fig. 5.16 shows one vertical slice containing, from top to bottom, modules of type B, A, C, and D. A considerable part of the area is used by the power supply stripes.

5.2.3 Simulation Results

Transient simulation of the decoding process

The analog network of the whole decoder chip has been simulated using Cadence's Spectre simulation tool. For MOS transistors, the BSim 3v3 model was used. A typical simulation response is shown in Fig. 5.17. It demonstrates the correction of two toggled bits for a binary symmetric channel (BSC) with crossover probability 0.05. We have chosen the configuration with the two toggled bits separated by 3 correct bits because it is hardest for the decoder circuit to recover the correct information in this case. The transient curves in Fig. 5.17 show



Circuit implementation of the type-A module: branch-metric precomputation. **Figure 5.15**

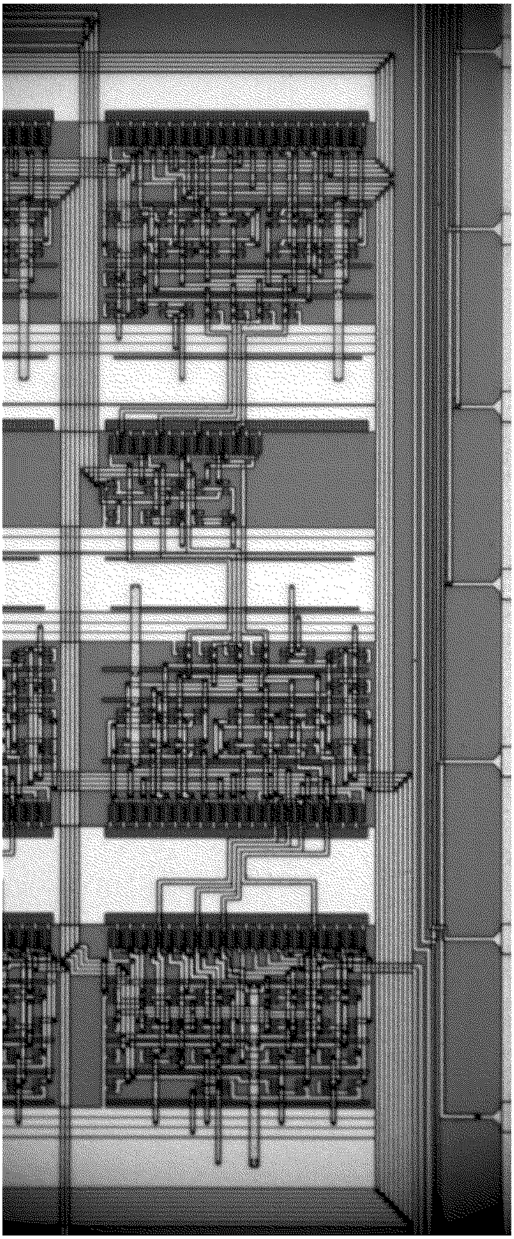


Figure 5.16 *A vertical slice of the layout containing, from top to bottom, modules of type* B, A, C, *and* D.

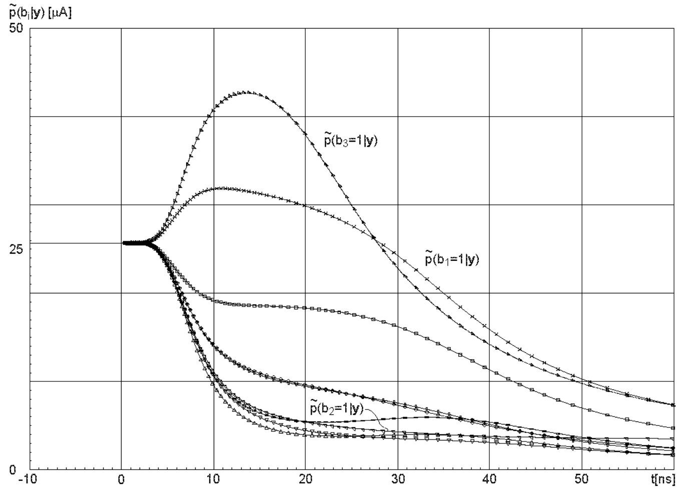


Figure 5.17

A typical simulation response for the decoder of Fig. 5.13: transient curves of the approximate a posteriori *probabilities* $\tilde{p}(u_j|\mathbf{y})$ *for* $j = 1, 2, \ldots, 9$ *given the transmitted all-zero codeword and two bit errors in the received channel information. (Probability 1 corresponds to* $50\,\mu$A.*)*

the computed approximate *a posteriori* probabilities $\tilde{p}(u_i|\mathbf{y})$ for all nine information bits. In this example, it takes about 30 ns until the sign of all output probabilities have reached their final value. The bias current of each module was chosen to be $50\,\mu$A. A probability of 1 corresponds to a current value of $50\,\mu$A, a probability of 0 corresponds to $0\,\mu$A. A single 5V supply was used, and the total (static and dynamic) power consumption was measured to be 50 mW.

Effects of different cross-over probabilities

To show the effects of different cross-over probabilities, i.e., different strengths of the conditional input probabilities, on the error-correcting capabilities, we made a sequence of five simulations with different ϵ. The circuits were biased this time with a current of $200\,\mu$A for each section, and we used also a different uncoded dataword $\mathbf{u} = [1,0,1,1,0,1,1,0,1]$ but left the two toggeled bits at the same position. Fig. 5.18 to Fig. 5.21 show the plots for BSCs with $\epsilon = \{40\%, 25\%, 5\%, 2.5\%\}$ respectively, whereas Fig. 5.22 represents the case with 4 erasures, i.e., the conditional input probability of the toggeled bits is 0.5. As we observe in the five plots, the case with 4 erasure

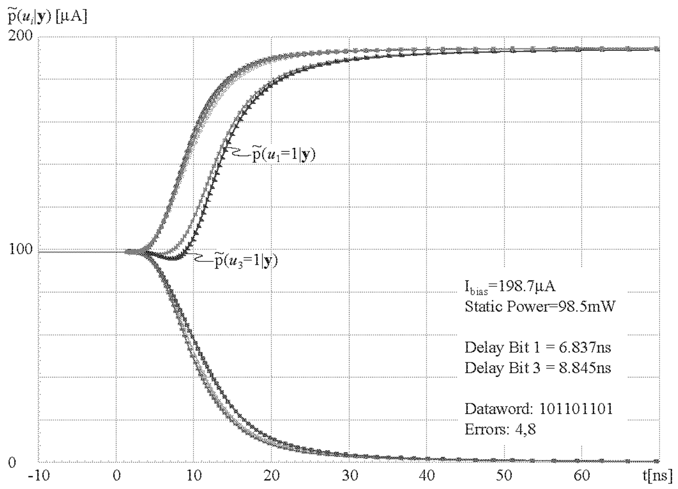


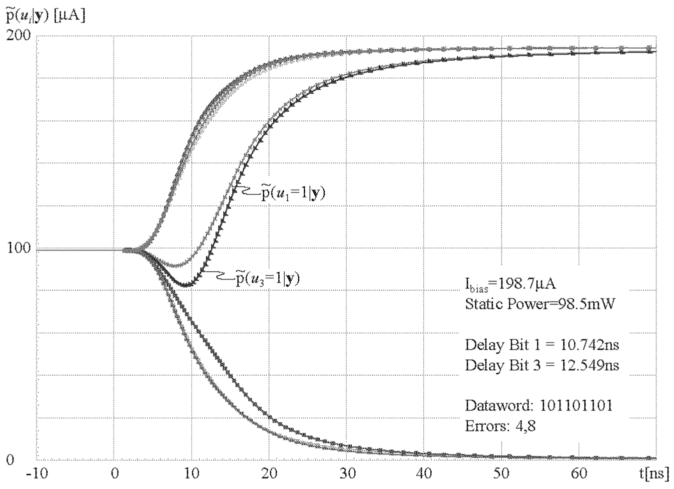
Figure 5.18 *Simulation of a decoder correcting two bit errors and* $\epsilon = 40\%$.

bits is the fastest one, since actually no error-correcting action was done. In the remaining four plots we observe that very strong errors make the decoding time considerably longer. On the other hand, very weak errors are easily corrected as we have seen before for the special case of erasures.

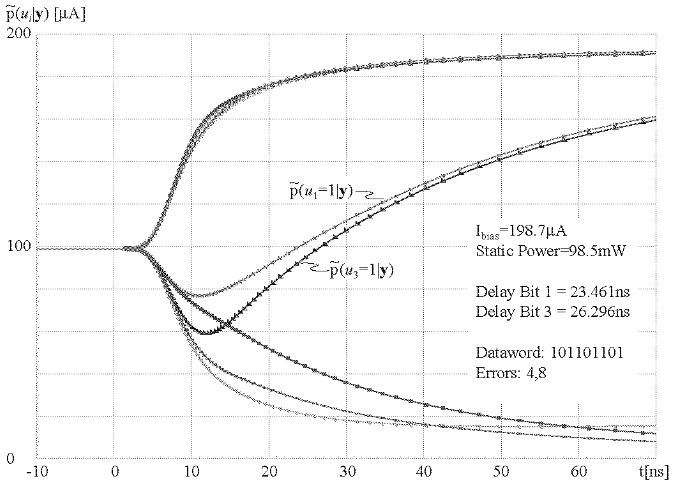
5.2.4 Test Setup

HP 83000-based testing

For testing the implemented circuit chip, an HP 83000 digital circuit tester was used as shown in Fig. 5.23. The tester commands the 18 off-chip high-speed D/A converters on the DUT adapter board for generating the input voltage waveforms. These input voltages are proportional to $p(y_i|x_i)$. In the complete transmission system, these probabilities stem from the output of the demodulator as shown in Fig. 2.10. Since the voltage signals have to be applied in parallel during the decoding process, the converters act as external analog memory, too. The input voltages are converted on chip into the current signals as needed by the decoder core. Additionally, the DUT adapter board generates bias currents and reference voltages for the test chip. And finally, the DUT adapter board contains nine



Simulation of a decoder correcting two bit errors and $\epsilon = 25\%$. **Figure 5.19**



Simulation of a decoder correcting two bit errors and $\epsilon = 5\%$. **Figure 5.20**

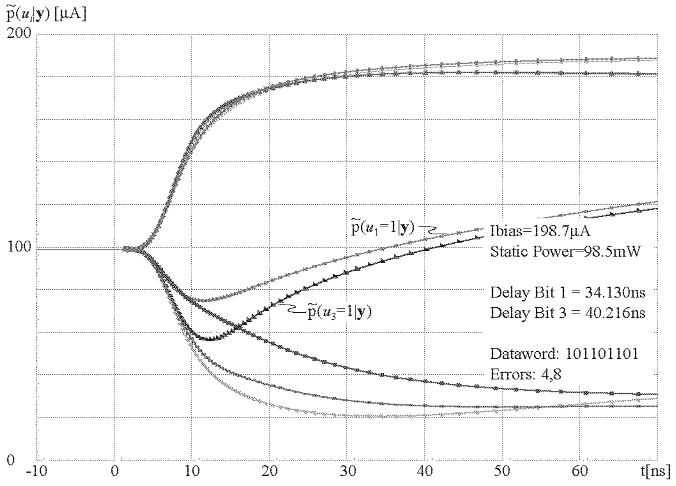


Figure 5.21 *Simulation of a decoder correcting two bit errors and* $\epsilon = 2.5\%$.

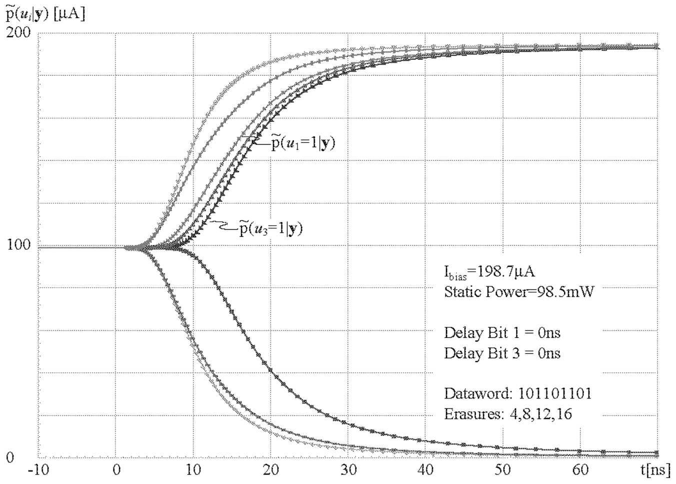


Figure 5.22 *Simulation of a decoder correcting 4 erased bits.*

I-V converter pairs and nine high-speed ECL comparators (bit slicers) for measurement purposes. The analog voltages may be measured by a high-speed oscilloscope, whereas the bit slicer outputs are directly fed back to the HP 83000 test system.

Measurement Results 5.2.5

Transient measurements of the decoding process

Measurements of the transient behaviour of the output probabilities, shown in Fig. 5.24, match well with the simulation results. The measured approximate output probabilities of bit 1 and bit 2 are drawn in this plot; the error correction of bit 1 can be seen clearly. Furthermore, the output of the external comparator for the hard decision on bit 1 is shown, making the decoding delay of approx. 31 ns evident. We observed heavy ringing of the output currents. On the one hand, this ringing was caused by coupling between the pins of the package, which could not be calibrated out completely. On the other hand, the cavity of the package was very wide and hence the long bond wires added considerable inductivity on all pins. This inductance prevents high-speed signals as well as fast changes of the current consumption to propagate properly and thereby generates oscillations. However, it was found that decoding speed and errors were not affected by this ringing. We could have prevented ringing by assigning the pinout properly and by using, for example, the chip-on-board (COB) mounting technique or flip-chip assembly, which almost eliminate the bond-wire inductances.

Measurement of the effect of several toggled bits

In Fig. 5.25, the result of another measurement is shown. The sourceword $\mathbf{u} = [1,0,1,1,0,1,1,0,1]$ is encoded and applied to a BSC with crossover probability 0.05. Consecutively, zero, one, two, and three bit errors were applied, where the three-error configuration corresponds to another codeword with two bit errors. The error correction capability for two applied errors can clearly be observed in Fig. 5.25. In this measurement setup, a probability of 1 corresponds to 200 μA and one oscilloscope division corresponds to 25 μA. The differences of the output amplitudes of bit 1 and bit 3 stem from device mismatch effects within the decoder core. As we have seen, the building blocks were designed with virtually minimal transistor sizes as in digital design. Extensive Monte-Carlo simulations were made, in which none of the simulated configurations failed and the standard deviation of the output currents was within 3 to 4 % of the

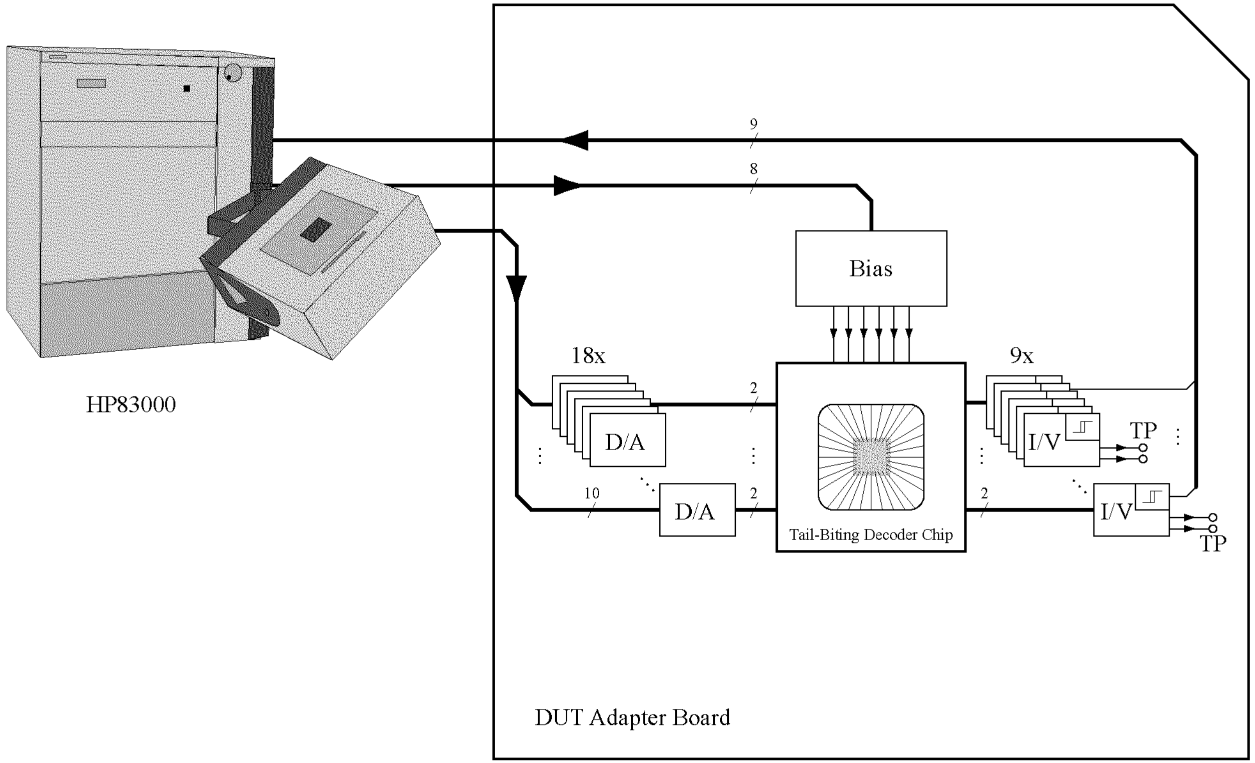


Figure 5.23: *The test setup based on the HP 83000 digital circuit tester.*

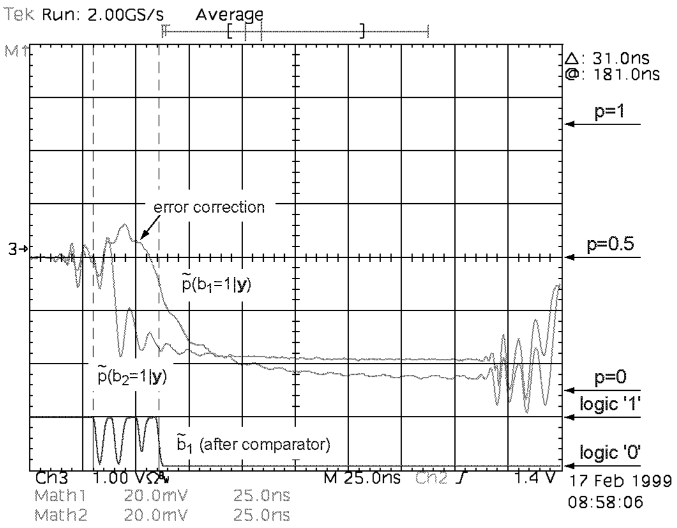


Figure 5.24 *A typical measured transient response of the output probabilities of bit* 1 *and bit* 2. 20 mV/div *correspond to* 10 μA/div *and probability 1 corresponds to* 50 μA.

nominal value. Therefore it is assured that the decoder is very robust against mismatch errors.

Problems during the measurements

Unfortunately, beside the ringing problem as visible in Fig. 5.24, we observed a second severe defect on the chip implementation. For very small input values, the V-I converters completely shut off the currents and thus created zero-probability paths in the tailbiting trellis. The decoder could not recover from a zero probability value. Hence, the malfunctioning V-I converters prevented us from measuring a BER curve, which would be the most important measurement to fully characterize the decoder chip.

Floorplan of the chip

A chip micrograph of the complete prototype implementation is shown in Fig. 5.26. The entire chip area is $2.8 \times 2.6\,\text{mm}^2$ including pads. The area of the decoder itself is $1.7 \times 0.7\,\text{mm}^2$, and it is situated in the lower right corner of the die. The remaining area is taken by the VI-converters needed for measurement purposes.

100 Mbit/s throughput measured

With a decoding time of 90 ns per decoded 9-bit sourceword, a data rate of 100 Mbit/s can be achieved, which includes am-

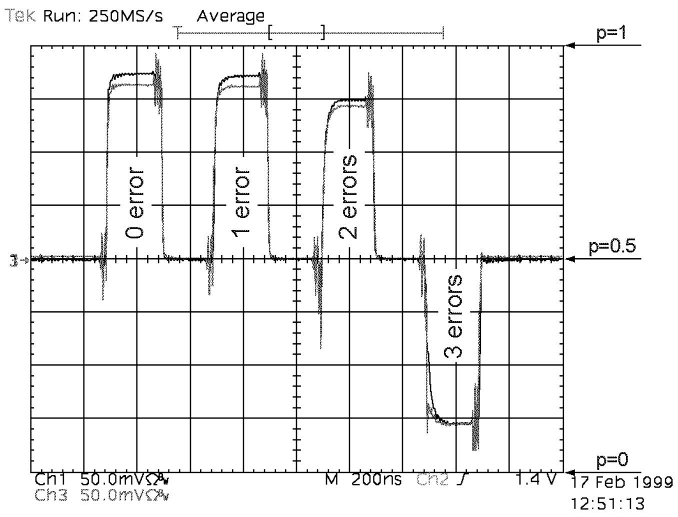


Figure 5.25 *Measured transient response of the output probabilities of bit* 1 *and bit* 3. *Consecutively, zero, one, two and three bit errors have been introduced. Each module is biased by* 200 μA *(corresponding to probability* 1*). In the plot,* 50 mV/div *correspond to* 25 μA/div.

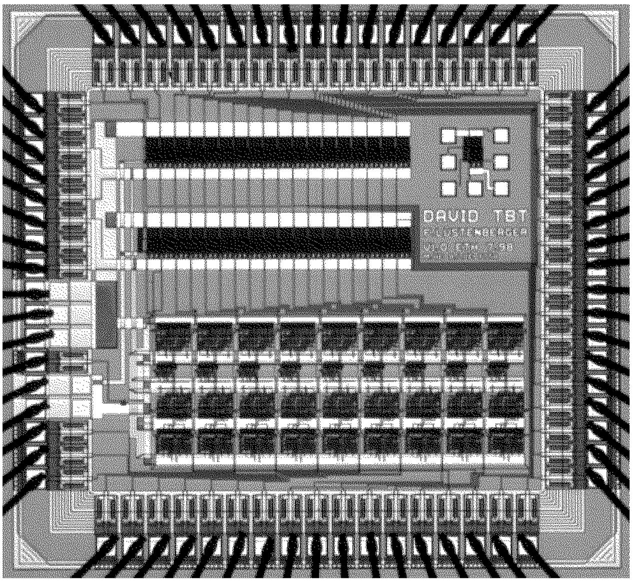


Figure 5.26

Chip micrograph of the protype chip. The actual decoder is on bottom right part of the chip, whereas the VI converters reside on the top half of the chip.

ple margin and reset time. The main chip characteristics are summarized in Table 5.1.

Power/Speed Comparison 5.2.6

Estimation of the digital complexity and the timing constraints

In the following, an estimate of the power consumption of an equivalent digital decoder is deduced. We found by high-level discrete-time simulations that the decoding algorithm converges to its final value after 15 iterations per section if parallel updates for each node type are assumed. This corresponds to a total of 4400 multiplications and 1100 additions. In order to achieve the desired bit rates, fast but low-power 5-bit multipliers and adders are assumed at each node of the factor graph of Fig. 5.12. The maximum operating frequency of these node processors is 167 MHz (15 iterations within 90 ns). Defining a gate as a basic digital circuit with up to n inputs and 1 output,

Technology	AMS 0.8 µm 2M2P BiCMOS
# BJT	940
# PMOS	650
Supply voltage	single 5 V
Power consumption @ $I_{\text{ref}} = 50\,\mu\text{A}$	50 mW (w/o V-I converters)
Chip area	$2.8 \times 2.6\,\text{mm}^2$
Core size	$1.7 \times 0.7\,\text{mm}^2$
Decoding speed (with margin)	100 MBit/s

Table 5.1 *Summary of the prototype chip characteristics.*

30 gates are needed for implementing the full adder and 110 for the one-step multiplier.

Estimation of the digital implementation

Furthermore, assuming that all gates have the same delay time t_{D}, an average node capacitance of 0.1 pF, an activity per gate of 1/4 (i.e. every node charges and discharges within 8 calculation steps) and an energy loss due to overlap currents of 20 %, the power dissipated per multiplication can be estimated as 2.5 mW. Similarly, 0.65 mW is required per 5-bit addition. Therefore, operating from a single 3V power supply and neglecting additional scaling operations and buffering, the overall power consumption for the decoder under consideration can be estimated to be above 11.5 W. A rough estimation has shown that adding the input network and the analog memories at most doubles power consumption and chip area. Therefore the power efficiency of our analog decoder is superior by a factor of more than 200 compared to its digital counterpart. Similar numbers could be found for the efficiency in the use of die area. A digital implementation of such iterative decoders is possible with today's CMOS processes if a delay per gate of $t_{\text{D}} = 0.45\,\text{ns}$ is assured for $n = 6$ processors in parallel. However, such *digital* implementations are limited to small codes, since otherwise the power consumption and die area become unpractically large.

Optimization of the gate level

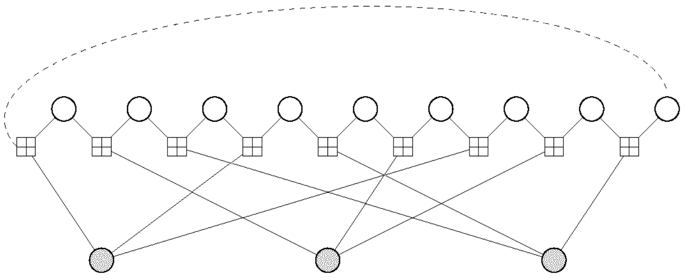
However, note that the digital implementation can potentially be optimized on the gate level. Instead of doing the full multiplication, table-lookup techniques can be applied; this is especially suitable for data quantized to only a few bits [143]. Additionally, the whole gate-level schematic can automatically be optimized by the digital design tools offered by most of the important vendors of IC design frameworks. By doing so, trade-offs can be evaluated easily.

Optimization in the update schedules

A second approach to optimized digital implementations are different update strategies. We might choose a more suitable update schedule in the digital domain for this decoder as discussed in Section 3.3.2. Instead of parallel updates of all nodes, the forward and backward trellises could be calculated serially with approx. 3 rounds each. This means that the presented analog decoding technique develops its full power for low-density parity-check codes, where the factor graph is highly connected. In this case, no simplifications can be made in the digital implementation.

Optimization on the algorithmic level

Another problem in comparing different implementation approaches, without having exact details of the actual design, are the different realizations of the same algorithm. The Max-Log-MAP [144], for example, approximates the MAP algorithm but significantly reduces the complexity by performing the calculations in the log-domain. Only additions and the max-operations are necessary in this case. Although the performance loss is about 0.3 dB at low SNR values, it may be used for low-power applications. As an illustration for this design uncertainty we cite the design study done by Vogt *et al.* [132]. They have compared different Turbo decoder realizations and found that the eficiency of these realizations differs by as much as a factor of 2.5.



○ parity bit ⊞ XOR function ● channel/info bit

Figure 5.27 *The structure of a basic repeat-accumulate (RA) code. Quasi-cyclic repeat-accumulate (QCRA) codes are formed by adding a tailbiting connection (dashed line) to the rectilinear part of the factor graph.*

5.3 Decoder for a Turbo-Style Code

A turbo-style QCRA code as the third decoder example

Our third complete decoder example is a decoder for a quasi-cyclic repeat-accumulate (QCRA) code. Repeat-accumulate (RA) codes are a special form of low-density parity-check codes with a factor graph as shown in Fig. 5.27. Tanner proposed the initial idea for the basic structure of the QCRA codes [145, 146]. This idea was refined by making numeric simulations to evaluate the performance of different realizations of the code [147]. In the following, we present the code structure, the design automation process that allows us the direct generation of the final layout, and simulation results. Unfortunately, the chip-on-board (COB) assembly of the chips failed due to bonding problems and we have still no working module to make measurements at the present time. So we just describe the test setup for the decoder chip at the end of this section.

5.3.1 General Code and Decoder Description

Code defined by LDPC matrix

The code chosen for our implementation is a linear [44,22,8] quasi-cyclic repeat-accumulate (QCRA) code that is characterized by the following parity-check matrix:

$$\mathbf{H} \triangleq \begin{bmatrix} \mathbf{I}^{(1)} & — & \mathbf{I}^{(0)} & — & — & \mathbf{I}^{(1)} \\ \mathbf{I}^{(0)} & \mathbf{I}^{(9)} & \mathbf{I}^{(1)} & \mathbf{I}^{(0)} & — & — \\ — & \mathbf{I}^{(0)} & — & \mathbf{I}^{(1)} & \mathbf{I}^{(0)} & — \\ \mathbf{I}^{(1)} & \mathbf{I}^{(4)} & — & — & \mathbf{I}^{(1)} & \mathbf{I}^{(0)} \end{bmatrix} \qquad (5.8)$$

where we use the following notation:

$$\mathbf{I}^{(0)} \triangleq \mathbf{I} = \begin{bmatrix} 1 & - & - & - & - & - & - & - & - & - & - \\ - & 1 & - & - & - & - & - & - & - & - & - \\ - & - & 1 & - & - & - & - & - & - & - & - \\ - & - & - & 1 & - & - & - & - & - & - & - \\ - & - & - & - & 1 & - & - & - & - & - & - \\ - & - & - & - & - & 1 & - & - & - & - & - \\ - & - & - & - & - & - & 1 & - & - & - & - \\ - & - & - & - & - & - & - & 1 & - & - & - \\ - & - & - & - & - & - & - & - & 1 & - & - \\ - & - & - & - & - & - & - & - & - & 1 & - \\ - & - & - & - & - & - & - & - & - & - & 1 \end{bmatrix}$$

$$\mathbf{I}^{(1)} \triangleq \begin{bmatrix} - & 1 & - & - & - & - & - & - & - & - & - \\ - & - & 1 & - & - & - & - & - & - & - & - \\ - & - & - & 1 & - & - & - & - & - & - & - \\ - & - & - & - & 1 & - & - & - & - & - & - \\ - & - & - & - & - & 1 & - & - & - & - & - \\ - & - & - & - & - & - & 1 & - & - & - & - \\ - & - & - & - & - & - & - & 1 & - & - & - \\ - & - & - & - & - & - & - & - & 1 & - & - \\ - & - & - & - & - & - & - & - & - & 1 & - \\ - & - & - & - & - & - & - & - & - & - & 1 \\ 1 & - & - & - & - & - & - & - & - & - & - \end{bmatrix}$$

etc. are shifted 11×11-element identity matrices, and '—' is a zero matrix of the appropriate size or '–' is a zero element in the matrix.

Turbine form of the factor graph

A corresponding factor graph representation is shown in Fig. 5.28. Note that the parity-check matrix **H** of (5.8) has been reordered, by doing linear combinations of rows and column permutations, to get the symbol numbering as shown in the factor graph of Fig. 5.28. We have chosen the even-numbered bits on the outer ring to be the information bits whereas the odd-numbered bits are parity bits. Note that the bits on the inner ring are auxiliary bits. Hence we have a systematic, linear, rate

1/2 code. The girth of the graph, i.e., the smallest number of branches of a closed loop in the factor graph, is $g = 10$. This is an important parameter, since the iterative decoding on factor graphs with cycles performs sub-optimally. The larger the girth, the smaller the influence of cycles on the decoding performance.

The factor graph serves directly as a model of the decoder circuit

The factor graph serves directly as the block diagram of the decoder circuit. Because of the form of the factor graph, we will denote the decoder as a 'Turbine decoder' in the following. In Fig. 5.28 we identify five types of nodes: fully bi-directional variable nodes on the innermost circle, fully bi-directional soft-XOR nodes with 3 or 4 ports on the second circle from the center, and partly bi-directional 2.5-port[1] variable nodes with and without an output for the information bits and the parity bits, respectively. Since the factor graph nodes work mostly fully bi-directionally and our building blocks allow only two-input circuits, we dissect them into building blocks as shown in Fig. 5.29. The lines in the figures represent actually two-element probability mass functions. The letters at the inputs of the building blocks indicate whether one enters by the x-inputs (bottom of the core transistor matrix) or by the y-inputs (on the left side of the core transistor matrix). In order to save on the input logarithm transistors as well as to omit unnecessary current duplicates, we try to reuse as many of the input terms as possible. However, this is not always possible. Some of the inputs have a letter in parentheses beside them such as, for example, (x). This indicates a domain change from a y input to an x input and vice versa. This increases the current consumption, since a scaler circuit of its own has to be used for that purpose.

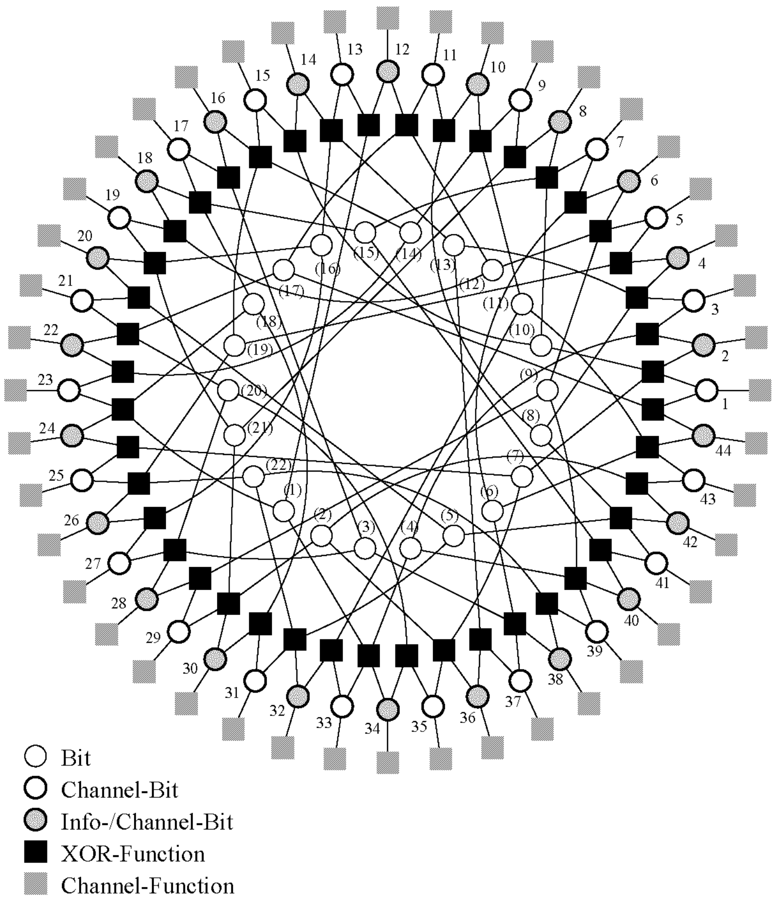
5.3.2 Circuit Design

General transistor sizing

Transistor sizing comparable to the tail-biting decoder

The transistor level schematics of the individual factor graph nodes are given in Fig. 5.30 and in Fig. 5.54 to Fig. 5.57. For most of them, the large schematics are divided into two parts that are shown on opposite pages and should be read together. The computation nodes have been implemented using the same

[1]The 'half port' is an input port only as needed for the input of the channel information



Factor graph of the binary linear [44,22,8] QCRA code. **Figure 5.28**

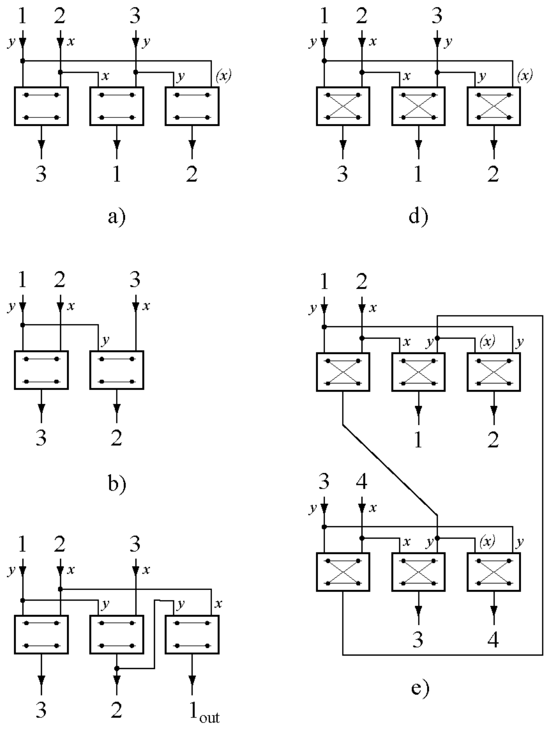


Figure 5.29 *Dissection of the factor graph nodes of Fig. 5.28 into elementary 2-input building blocks with a) fully bi-directional 3-port variable node, b) bi-directional 2.5-port variable node, c) bi-directional 2.5-port variable node with output bit slicer, d) fully bi-directional 3-port soft-XOR node, and e) fully bi-directional 4-port soft-XOR node.*

BiCMOS technology from AMS as the one we used for the tail-biting decoder described in Section 5.2 [140]. Again, all the bipolar transistors are minimum size ($3 \times 0.8\,\mu\mathrm{m}^2$). The pMOS transistors located in the current mirrors on the top of the circuits are also almost minimum size ($12 \times 1.6\,\mu\mathrm{m}^2$). They were designed for maximum speed in a 5V design and $V_{\mathrm{Dsat}} =$ 2V. Hence, the current mirrors operate in the strong-inversion region for the nominal currents of $200\,\mu$A.

Input variable nodes

Reset circuitry for the input variable nodes

The 2.5-port factor graph nodes for input variables of Fig. 5.28 need some special attention. In order to have predictable initial conditions for the sum-product algorithm, the input probabilities of the network are generally set to a uniform distribution. Therefore, additional initialization functionality has been implemented in the circuits of the affected 2.5-port computation nodes by means of clamping reset switches as shown in Fig. 5.54 and Fig. 5.55. By clamping the binary input distributions together, the input current sum is forced to be theoretically distributed uniformly on the two diode-connected input transistors of port-1. Unfortunately, the pMOS transistor $\mathrm{M}_{1\mathrm{r}}$ ($W/L = 20\,\mu\mathrm{m}/0.8\,\mu\mathrm{m}$) is not a perfect switch. More likely it behaves as a MOS resistor in the linear range, and the drain-source resistance is not negligible at a zero gate voltage. This causes a differential voltage-drop at the input port 1 that directly translates into an imbalance of the input distribution and thereafter also in the output distributions. Even if the transistor $\mathrm{M}_{1\mathrm{r}}$ had been made very wide, the parasitic channel resistance would not have dropped to an acceptable level. Therefore the additional transistors $\mathrm{M}_{2\mathrm{r}}$ and $\mathrm{M}_{3\mathrm{r}}$ ($W/L = 40\,\mu\mathrm{m}/0.8\,\mu\mathrm{m}$) are connected to the two output ports 2 and 3 to equalize the output distributions during the reset time. The transistors prove to be very efficient in equalizing the input distributions of the whole network even at their relatively small size.

Current-mode comparator

Bit slicer for the information bit nodes

In addition to the reset hardware, the 2.5-port variable node for the information bits also contains the output circuitry. This includes the summarization calculations, i.e., the pair-wise product of all incoming branches, and a bit slicer. The bit slicer

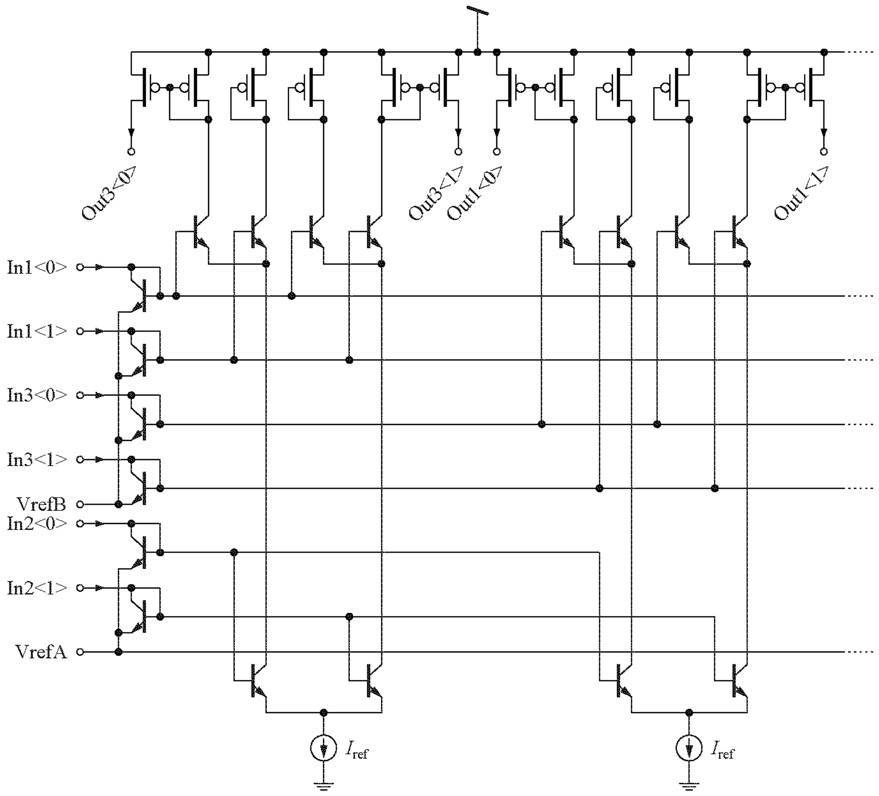
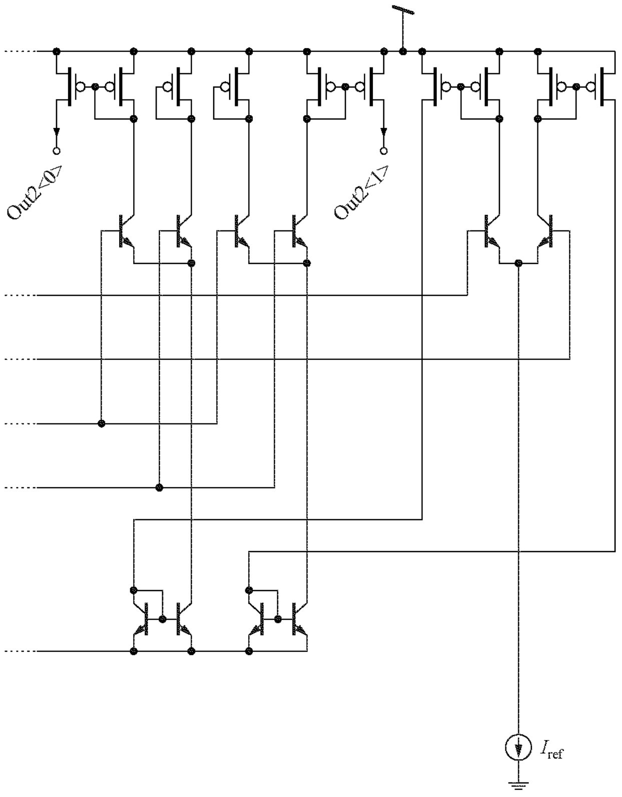


Figure 5.30 *Circuit implementation of the fully bidirectional 3-port variable node (left part). Denoted as 'Bit' in Fig. 5.28.*



Circuit implementation of the fully bidirectional 3-port variable node (right part). Denoted as 'Bit' in Fig. 5.28. **Figure 5.30**

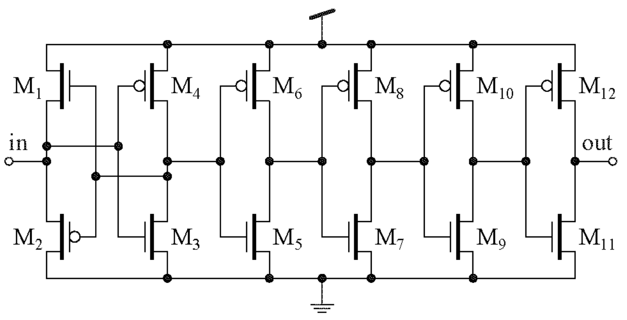


Figure 5.31 *Current comparator for the output bit slicer.*

is actually a current comparator that compares the the two elements of the output distribution. If $\tilde{p}(u_i = 1|\mathbf{y}) > \tilde{p}(u_i = 0|\mathbf{y})$ it puts a logic '1' at the output and a logic '0' otherwise. The actual comparator of Fig. 5.31 can decide only whether the signal current is positive (flowing into the circuit) or negative (flowing out of the circuit). Therefore a precomputation has to be done with the output currents according to

$$I_{\text{comp}} = I_z\tilde{p}(u_i = 1|\mathbf{y}) - \frac{I_z\tilde{p}(u_i = 1|\mathbf{y}) + I_z\tilde{p}(u_i = 0|\mathbf{y})}{2}. \quad (5.9)$$

This actually calculates a *dynamic threshold* value corresponding to the probability of 0.5 and subtracts the output value corresponding to the conditional probability of being a '1'. This value is not strictly positive anymore and may hence serve as the input of the current comparator.

Traff's current-mode comparator

The comparator circuit of Fig. 5.31 is a class-B circuit that uses positive feedback in the first stage. It was initially proposed by Traff [148]. Transistors M_1 and M_2 form a 'head-over' digital inverter followed by a traditional unit-size inverter. At the output of this bistable circuit we observe only small voltage variations that are subsequently amplified by a scaled inverter chain according to the grading 1:1:2:4 of the nominal transistor widths. The sizing of an unit inverter is $W/L = 9\,\mu\text{m}/0.8\,\mu\text{m}$ for pMOS transistors and $W/L = 5\,\mu\text{m}/0.8\,\mu\text{m}$ for nMOS transistors, respectively. This sizing corresponds to the one found in the digital cell library.

On-chip D/A converter

On-chip D/A converter to provide input values to the decoder core

To drive the inputs of the Turbine-decoder core, 7-bit current-output D/A converters with an input latch are placed on the chip. This is different from the previous decoder example, where the input signals are produced externally on the DUT adapter board by means of high-speed D/A converters. The main benefit from placing the converters directly near the input nodes on the chip is a better speed performance and no problems from zero currents, which would subsequently result in zero-probability paths. A current steering architecture has been chosen due to its simplicity and robustness [108, 137]. As shown in Fig. 5.32, the currents, which are normally dumped into a dummy load, are used for the complementary value of the binary input distribution. By programming the 7-bit value to the register we choose the conditional input probability $p(y_i|x_i = 1)$ according to

$$\begin{aligned} I\,p(y_i|x_i = 1) &= 4\,I_{\text{ref}}\,p(y_i|x_i = 1) \\ &= 4\,I_{\text{ref}}\left(\frac{b_1}{2}+\frac{b_2}{4}+\frac{b_3}{8}+\frac{b_4}{16}+\frac{b_5}{32}+\frac{b_6}{64}+\frac{b_7}{128}\right), \end{aligned} \tag{5.10}$$

where b_i are the programmed values of the D/A converter and I_{ref} is set to $I_{\text{MSB}}/4$. Conversely we get

$$\begin{aligned} I\,p(y_i|x_i = 0) &= 4\,I_{\text{ref}}\,p(y_i|x_i = 0) \\ &= 4\,I_{\text{ref}}\left[1-\left(\frac{b_1}{2}+\frac{b_2}{4}+\frac{b_3}{8}+\frac{b_4}{16}+\frac{b_5}{32}+\frac{b_6}{64}+\frac{b_7}{128}\right)\right] \end{aligned} \tag{5.11}$$

for the complement of $p(y_i|x_i = 1)$.

Robust design of the current-steering D/A converter

For the binary-weighted current sources we used simple cascode structures. The current switches are of very small size ($W/L = 2\,\mu\text{m}/0.8\,\mu\text{m}$) to allow fast switching. The summed currents are passed to the input of the decoding network by current mirrors. The current mirror transistors are all equipped with regulated-cascode structures [135]. Hence, the resulting high-impedance output of the D/A converter provides well defined input values to the decoding network. The layout of the complete converter, as shown in Fig. 5.33, is as compact as possible ($334 \times 434\,\mu\text{m}^2$). However, the D/A convert design follows a traditional design approach and the layout therefore uses

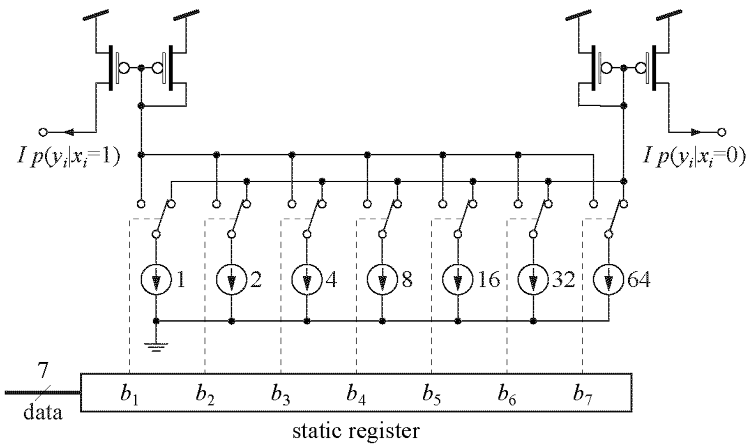


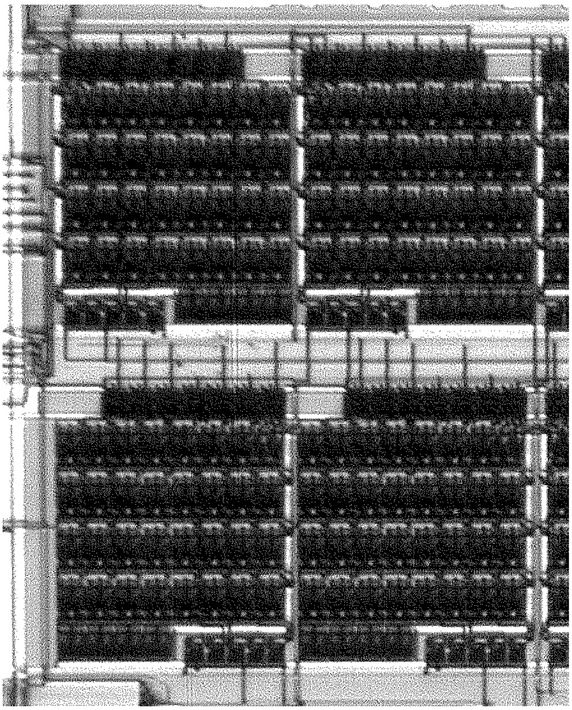
Figure 5.32 *Current-steering D/A converter with binary-weighted current sources.*

a considerable amount of silicon area due to matching considerations. 45 of these D/A converters are placed on the chip: 44 converters are used for the decoding network inputs, and for one converter the output signals are connected to the chip pads for measurement purposes.

5.3.3 Automating the Design Process

Generating the schematics and layout of a large analog network

Drawing schematics and layouts of analog decoding networks becomes too complicated for large codes. The design is very susceptible to drawing errors that will later on affect the overall behaviour of the circuit. Additionally, the design time may get far too long, and changes in the design may require a complete redrawing of all the elements of the network. Hence we have to look for a design flow that offers the construction of the decoding network. Our aim is thus a computer-assisted methodology that checks the critical parts, i.e., the interconnections in the schematic and in the layout of the circuit chip. We have developed such a method in the context of the chosen QCRA code, which we will describe in the following.



A part of the layout showing four current-steering D/A converters used for the input-value generation for the decoding network. One D/A converter measures $334 \times 434\,\mu\mathrm{m}^2$.

Figure 5.33

The parity-check matrix is the basis for the automated design flow

The parity-check matrix **H** of (5.8) together with the allocation of the information bits and the channel bits comprises the entire information needed to generate both the factor-graph representation and the actual decoder structure. Thus we can parse this matrix and extract the information needed for the design. Each row of the matrix **H** provides the information about the parity checks (XOR functions). The rows with 3 or 4 entries correspond to 3-port and 4-port fully bi-directional soft-XOR modules. In the same way we analyze the columns. In the case of the [44,22,8] QCRA code, the first 22 columns of the 66×44 parity-check matrix correspond to the non-observable, fully bi-directional internal 3-port variable nodes. The remaining 44 columns then correspond to the 2.5-port variable nodes. The module incorporates the output functionality if the corresponding bit is an information bit. After having parsed the entire parity-check matrix, we can generate a text file in the Verilog syntax by the same program. This file contains the structural description of the decoder. Subsequently, this Verilog file is imported as a schematic file into the Cadence design environment by the program *VerilogIn*. In a next step, the transistor-level schematics of the building blocks are filled into the empty place-holders of our design structure. Now we are ready for the transistor-level simulations, since we have a complete hierarchic transistor-level description of our circuit. The circuit simulator tools such as Spectre are well embedded into the design framework. Thus the schematic is a central pivot of the whole design flow.

Design of analog standard cells

The second part of the decoder design consists in drawing the full-custom layout of individual building blocks. They are designed to be placed in a library as analog standard cells. This library can then be used with a (digital) place-and-route (P&R) tool such as CellEnsemble or SiliconEnsemble of the Cadence IC design framework. Special attention has to be paid to the choice of the width of an individual cell such that the modules are placed correctly by the software tool. After having drawn the individual cells, the layout of the entire decoder can be generated automatically by the P&R tools of the design frame work. For the final chip, the additional bias-network blocks can be assembled semi-automatically inside the pad ring of the chip layout.

A flow-chart of the design procedure

We have sketched the procedure presented above in the flow-chart shown in Fig. 5.34. The described method has been sucessfully applied to the design of our test chip. Fig. 5.35 shows a clipping of the generated decoder core. Clearly visible is the massive power-feeding trunk on the right of the image that provides the supply for the decoder core.

Simulation Results 5.3.4

Fast transients of the decoder circuit

The analog network of the whole decoder chip has been simulated using Cadence's Spectre simulation tool. The BSim 3v3 model was used for the MOS transistors. A typical simulation response is shown in Fig. 5.36. It demonstrates the correction of three toggled bits (channel bits 2, 3, and 4) on a binary symmetric channel (BSC) with a crossover probability of 5%. We have chosen the configuration with three consecutively toggled bits because it is hardest for the decoder circuit to recover the correct information in this case. In fact, the toggled bits are located on the same minimum-girth loop, as we can easily see in Fig. 5.28. The transient curves in Fig. 5.36 show the computed approximate *a posteriori* probabilities $\tilde{p}(u_i|\mathbf{y})$ for all 22 information bits. In this example, it takes about 100 ns until the sign of all output probabilities have reached their final value. The bias current of each module was chosen to be 200 μA. A probability of 1 corresponds to a current value of 200 μA, a probability of 0 corresponds to 0 μA. A single 5V supply was used. The total maximum (static and dynamic) power consumption was estimated to be 1 W. In Fig. 5.36 we also observe the nonperfect action of the clamping reset switches. It always results in a light preference of the applied input value, i.e., a correct bit immediately starts in the right direction of the decoding trajectory.

Existence of variable decoding difficulty

As soon as the errors are no more placed on the same minimum-girth loop, the decoding network gets considerably faster. This behaviour is shown in Fig. 5.37 and Fig. 5.38. In these simulations we use the same simulation conditions as before. We observe an opposite situation if we introduce more toggled bits than the decoder is capable of correcting. An oscillatory behaviour is observed if 4 input bits on the same minimum-girth loop are toggled. The transient simulation of this input configuration is shown in Fig. 5.39. Note that we also observe the same oscillations in digital simulations. Hence these oscillations are an inherent problem of this code.

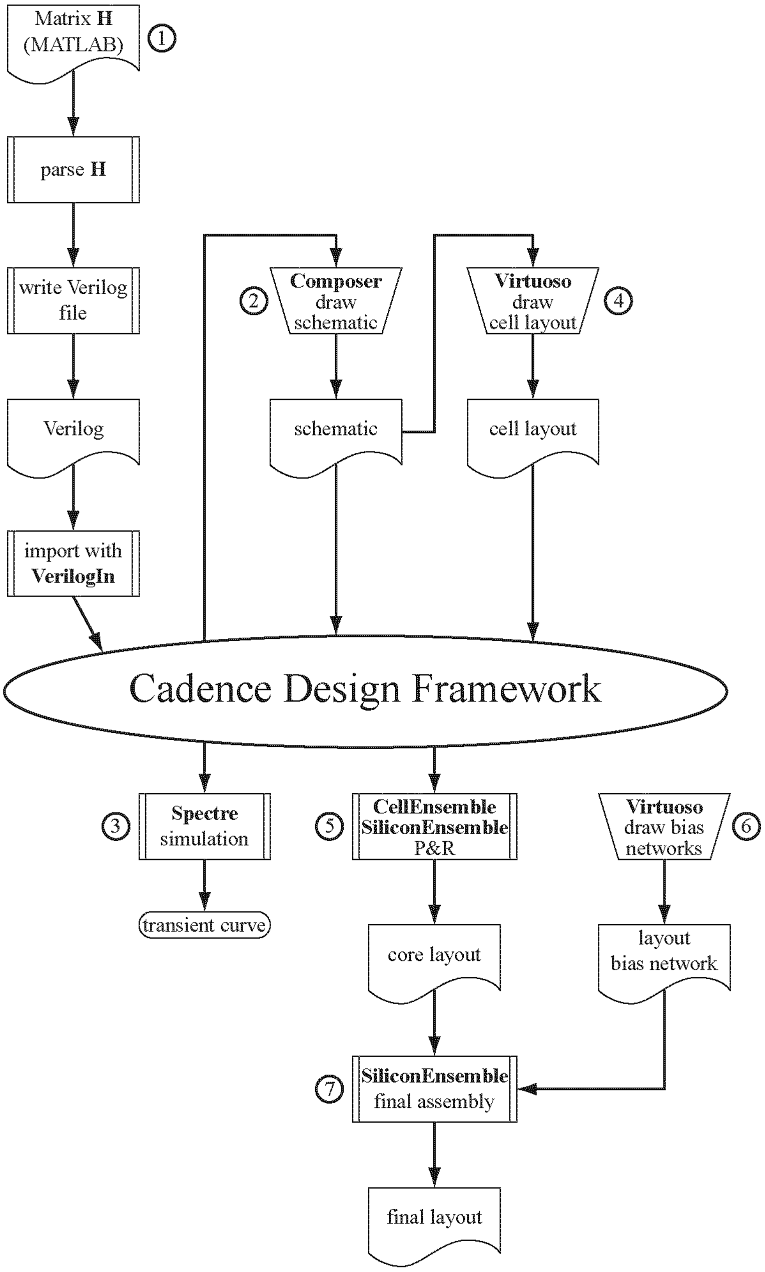
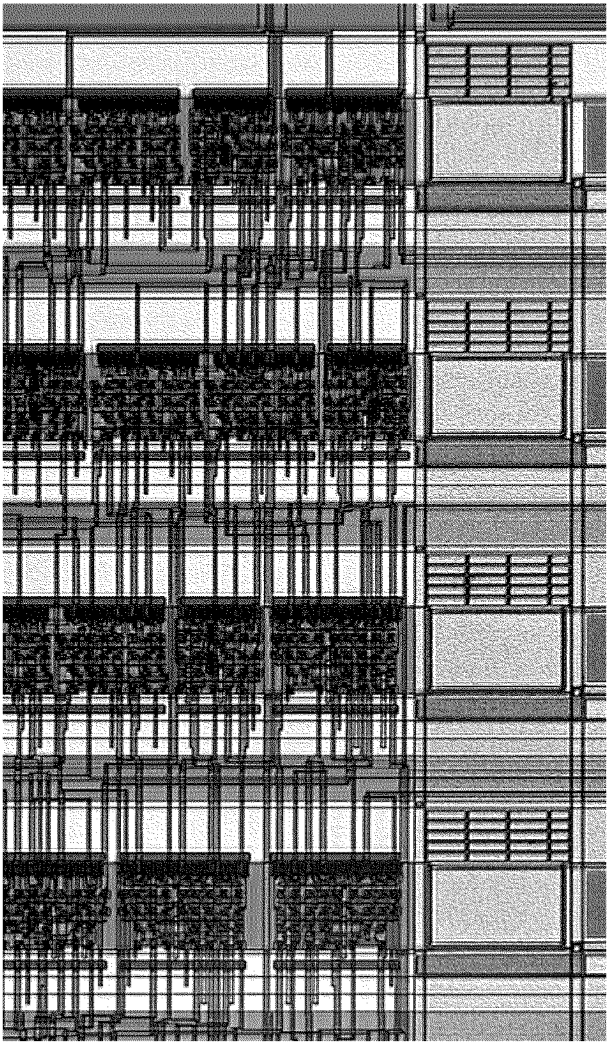


Figure 5.34 *Design flow used to automate the construction of large analog decoding networks.*



A part of the automatically generated layout of the turbine decoder. The massive metal lines on the right are part of the power-feeding trunk. **Figure 5.35**

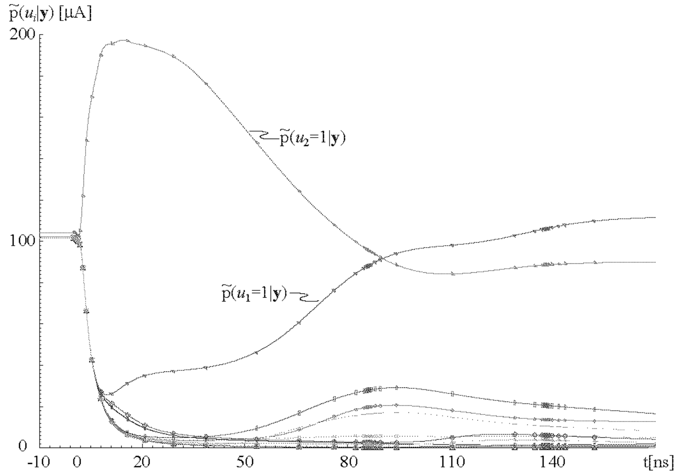


Figure 5.36 *A typical simulation response for the decoder of Fig. 5.28: transient curves of the approximate* a posteriori *probabilities* $\tilde{p}(u_j|\mathbf{y})$ *for* $j = 1,2,\ldots,22$ *given the encoded information word* $\mathbf{u} = \{1,0,0,0\ldots,0\}$ *and three bit errors in the received channel information (channel bits 2, 3, and 4). Probability 1 corresponds to* 200 μA.

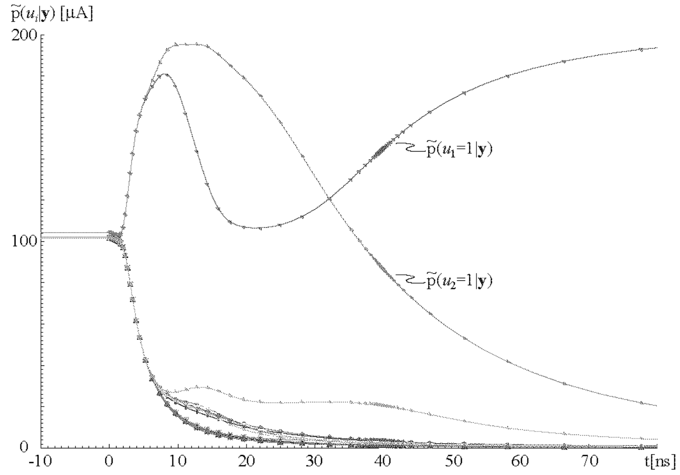
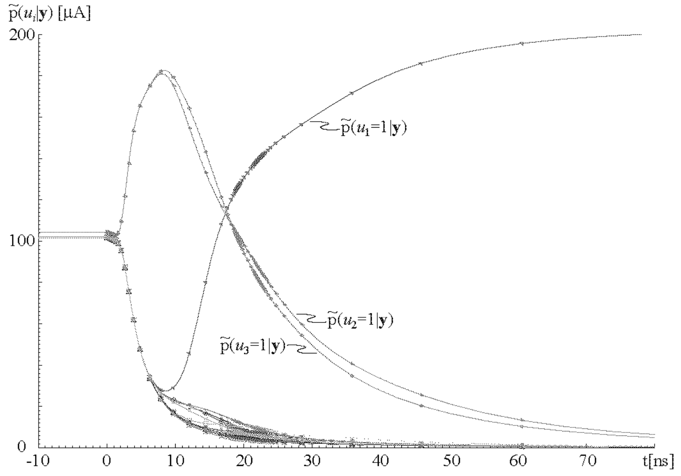
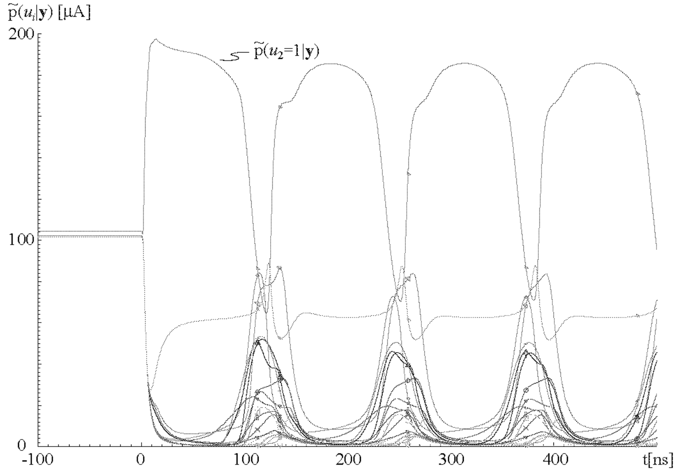


Figure 5.37 *Transient simulation using the same simulation conditions as before. Bits 1, 3, and 4 are toggled in this case.*



Transient simulation using the same simulation conditions as before. Bits 2, 4, and 6 are toggled in this case. **Figure 5.38**



Transient simulation using the same simulation conditions as before. Bits 1, 2, 3, and 4 are toggled in this case. Oscillations are also observed in time-discrete simulations. **Figure 5.39**

Technology	AMS 0.8 μm 2M2P BiCMOS
# BJT	3895
# PMOS	2884
Supply voltage	single 5V
Power consumption @ $I_{\text{ref}} = 200\,\mu\text{A}$	1 W (estimated)
Chip area	$5.28 \times 5.45\,\text{mm}^2$
Core size	$2.7 \times 2.5\,\text{mm}^2$
Decoding speed (with margin)	150 MBit/s (estimated)

Table 5.2 *Summary of the simulated chip characteristics of the prototype turbine decoder.*

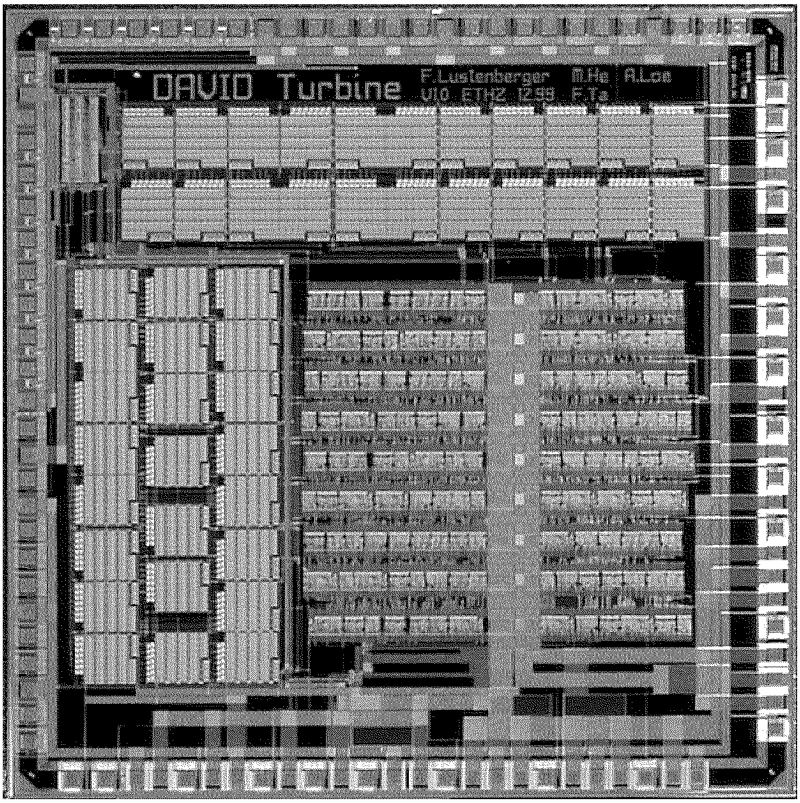
5.3.5 Test Setup

Floorplan of the turbine decoder chip

A chip micrograph of the complete prototype implementation is shown in Fig. 5.40. The entire chip area is $2.8 \times 2.6\,\text{mm}^2$ including pads. The area of the decoder itself is $1.7 \times 0.7\,\text{mm}^2$. The decoder is situated in the lower right corner of the die. The remaining area is taken by the D/A converters needed for the input value generation. The main simulated chip characteristics are summarized in Table 5.2.

The HP 83000 based test system

For testing the integrated circuit chip, an HP 83000 digital circuit tester is used as shown in Fig. 5.41. The silicon dies are mounted by the COB assembly technique on small PCBs. The COB adapter is then connected to the DUT adapter board. The digital tester directly commands the on-chip D/A converters using a 6-bit address bus and a 7-bit data bus. The DUT adapter board generates the bias currents and the reference voltages for the test chip. Additionally, the DUT adapter board contains eleven I-V converters for half of the analog probability outputs. The analog voltages may be measured by a high-speed oscilloscope, whereas the on-chip bit slicer outputs are directly fed back to the HP 83000 test system on a 22-bit wide digital bus.



Chip micrograph of the protype turbine decoder chip. The actual decoder is the bottom right part of the chip, whereas the D/A converters are placed in an L-shape on the remaining space. **Figure 5.40**

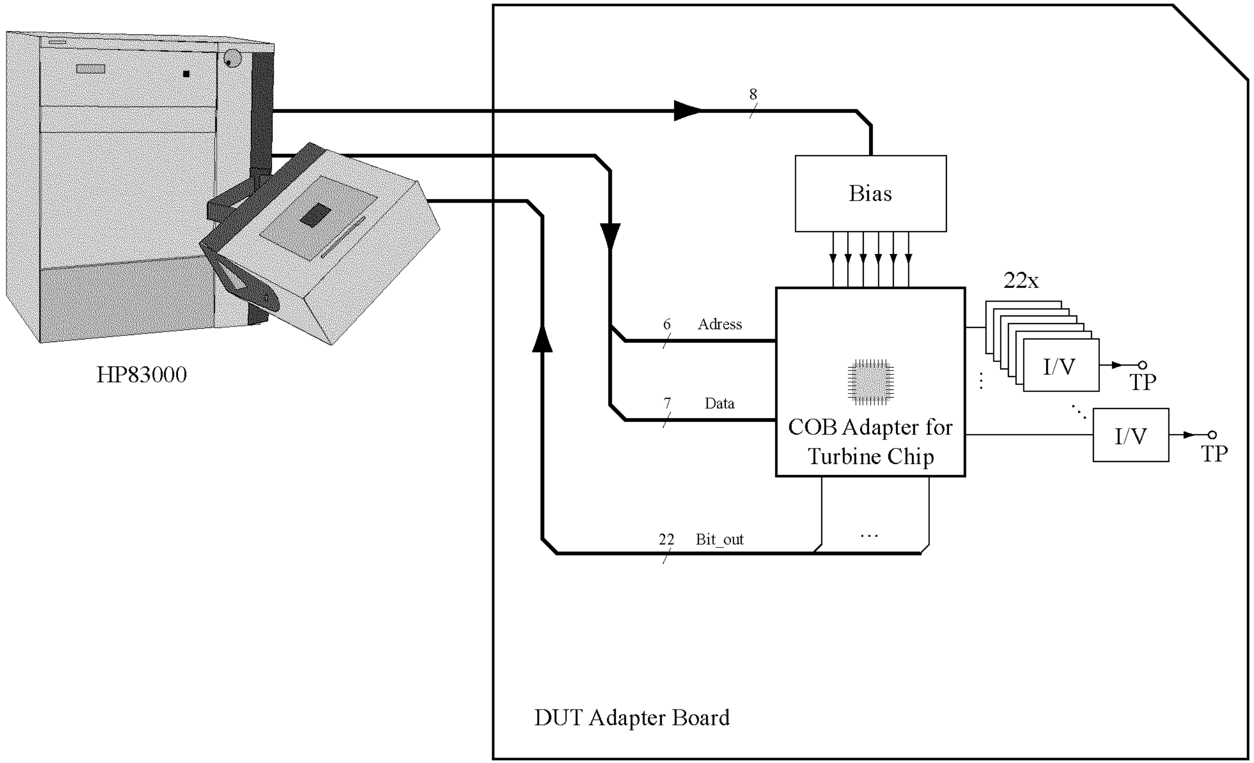


Figure 5.41: *The test setup of the turbine-decoder based on the HP 83000 digital circuit tester.*

5.4 Analog Viterbi Decoder Using Probablity Propagation Modules

Valuable Viterbi algorithm

The Viterbi algorithm [23,24] is an essential tool for every communications engineer. Several analog implementations have been made so far to speed up Viterbi decoders and lower their power consumption [28, 32, 37, 38]. In all these analog implementations, the digital add-compare-select (ACS) units, which are the key parts of the whole decoder, are replaced by analog ones, but the rest of the decoder, in particular the path memory and the survivor-path trace-back units, is still digital.

In the following, we will reformulate the original Viterbi algorithm to fit the framework of sum-product calculus and probability propagation networks. Using this new formulation we will then propose a modified architecture for the implementation of large probability-propagation-based Viterbi decoders which could be used, for example, for the decoding of highly complex trellis codes and channel tracking applications.

5.4.1 Reformulation of the Viterbi Algorithm

The general MAP sequence detection Problem

MAP sequence detection of a Markov process

The Viterbi algorithm was initially proposed in 1967 as a method of decoding convolutional codes [23]. It has been recognized since then that it is a general recursive solution to the MAP sequence detection problem of a finite-state discrete-time Markov process observed under memoryless noise conditions [24]. The encoding Markov process can be characterized by a state transition diagram as given, for example, in Fig. 2.3. The observed process is Markov in the sense that the probability $P(x_{k+1}|x_0,x_1,\ldots,x_k)$ of being in state x_{k+1} at time $k+1$ given all states up to the time k, depends only on the state x_k at time k:

$$P(x_{k+1}|x_0,x_1,\ldots,x_k) = P(x_{k+1}|x_k). \tag{5.12}$$

Problem description by a state-transition diagram

Furthermore, the state sequence of all possible transitions can be described by a trellis diagram. In fact, the Viterbi algorithm can be seen as the solution of the so-called shortest-path problem through a given graph, i.e., the trellis diagram.

The original definition of the Viterbi algorithm tries to maximize the joint probability measure

$$P(\mathbf{x},\mathbf{y}) = P(\mathbf{x})P(\mathbf{y}|\mathbf{x}) \tag{5.13}$$

$$= \prod_{k=0}^{K-1} P(x_{k+1}|x_k) \prod_{k=0}^{K-1} P(y_k|\underbrace{x_{k+1},x_k}_{\xi_k}), \tag{5.14}$$

where **x** is the state vector and **y** is the observed vector of length K, and using a convenient definiton of the transition ξ_k at time k as the pair of states (x_{k+1},x_k). Usually, K is called the *constraint length* of a decoder. If we now assign a distance measure for each branch of the trellis diagram, i.e., for each state transition ξ_k, according to

$$\lambda(\xi_k) \triangleq \ln P(x_{k+1}|x_k) + \ln P(y_k|\xi_k) \tag{5.15}$$

we can rewrite (5.14) as

$$\ln P(\mathbf{x},\mathbf{y}) = \sum_{k=0}^{K-1} \lambda(\xi_k), \tag{5.16}$$

which has to be maximized to get the sequence most probably generated by the encoding process, given our observation.

Viterbi algorithm for an AWGN channel

Path metrics for an AWGN channel

We consider in the following a coded transmission system where the channel can be modeled as an AWGN channel. Hence, using the definition (2.12) of the probability density function of a Gaussian process, we may now define the actual path metrics as

$$\mu \propto p(y|x) \propto e^{-\frac{(y-x)^2}{2\sigma^2}}, \tag{5.17}$$

$$\lambda = -(y-x)^2 \propto \ln\mu, \tag{5.18}$$

where λ is the squared Euclidean distance measure between the sent and the received symbol and μ is proportional to the conditional probability $p(y|x)$ which is simply a shifted Gaussian distribution. According to (5.16), the path length expression

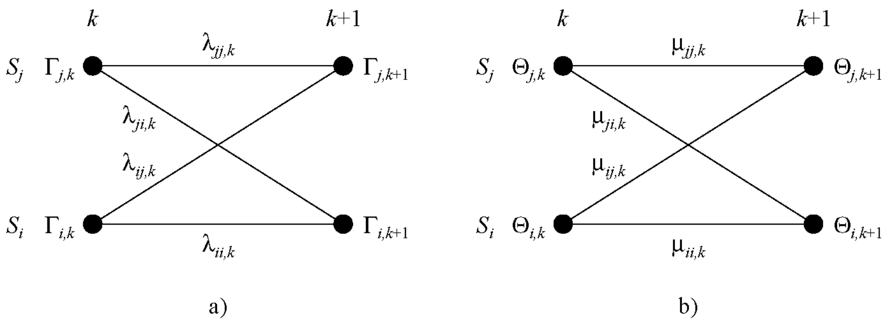


Figure 5.42 *A simple butterfly trellis section used for defining the update rules of the Viterbi algorithm. Fig. a) shows the notation of the max-sum formulation and b) shows the sum-product case.*

may be written in different forms:

$$\ln \prod_{k=0}^{K-1} \mu_k = \sum_{k=0}^{K-1} \ln \mu_k \propto \sum_{k=0}^{K-1} \lambda_k. \tag{5.19}$$

Solution of the shortest path problem

Since we are heading for the global maximum of the path length, the maximum has also to be satisfied locally. So we can finally formulate the well-known iterative max-sum Viterbi algorithm update rule for encoders with a single input bit:

$$\Gamma_{i,k+1} = \max\left[\lambda_{ii,k} + \Gamma_{i,k}, \lambda_{ji,k} + \Gamma_{j,k}\right], \quad 0 \leq i, j \leq K-1 \tag{5.20}$$

for the update from time instance k to time instance $k+1$ (see Fig. 5.42a)). The variables Γ are the accumulated path metrics for each state of the trellis diagram. The result of the selection operation (max) at each state is stored in a memory as 1 bit of digitial information, hence the name storage-survivor memory. This local decision information is used later on for tracing back the most probable (or shortest) path.

Changing the definition of the path metrics

Instead of maximizing $\sum_k \lambda_k$ we could equally well maximize $\ln \prod_k \mu_k$ of the original Viterbi algorithm formulation of (5.14). This formulation of the Viterbi algorithm is also known as the max-product formulation. It has been used for a long time to approximate the sum-product algorithm because of its much lower computational complexity. Instead of approximating the sum-product algorithm with the max-product calculus, we could equally well do the opposite. By some slight modifications of the max-product update rules, we can state the iterative update rules of the sum-product formulation of the Viterbi algorithm:

$$\Theta_{i,k+1} = \mu_{ii,k} \cdot \Theta_{i,k} + \mu_{ji,k} \cdot \Theta_{j,k}, \quad 0 \le i,j \le K-1. \quad (5.21)$$

with the notation according to Fig. 5.42b). Again, the variables Θ are the accumulated path metrics for each state of the trellis diagram.

Comparison of two versions of the Viterbi algorithm

If we compare the original max-product formulation of the Viterbi algorithm with the sum-product algorithm approximation, we observe strict equivalence if $\max(\mu_1, \mu_2) = \mu_1 + \mu_2$. So let us briefly analyze the following expression assuming an AWGN channel and $\mu_1 > \mu_2$ or equivalently $a_1 < a_2$, where the parameters a_1 and a_2 were introduced for mathematical simplicity only:

$$\frac{\mu_1 + \mu_2}{\mu_1} = 1 + \frac{\mu_2}{\mu_1} = 1 + \underbrace{e^{\frac{-(a_2^2 - a_1^2)}{2\sigma^2}}}_{\lim_{\sigma \to 0}(.)=0} \approx 1 = \frac{\max(\mu_1, \mu_2)}{\mu_1} \quad (5.22)$$

Free tuning parameter σ

The parameter σ is a free tuning factor and has the meaning of a signal-to-noise ratio (SNR). But do not confuse this parameter σ with the SNR of the AWGN channel. Now we observe that by diminishing σ to zero, the path metrics are pushed more and more apart from each other. At the limit when $\sigma = 0$ we get exact equivalence with the maximization function. Hence, the sum-product Viterbi algorithm formulation has an independent tuning parameter, which allows to get results equivalent to those of the traditional max-sum formulation (or min-sum formulation if the path metrics are defined by $-\ln \mu$).

Simulation results for the two formulations

Simulation results like the one in Fig. 5.43 for a rate $R = 1/2$ [171,133] convolutional code confirm this theoretical equivalence. If we choose a fixed $\sigma < 0.1$, corresponding to an $\mathrm{SNR}_{\mathrm{decoder}} > 20\,\mathrm{dB}$, the coding loss gets negligible. Conversely, one may run into numeric problems if one chooses σ too small. In a nutshell: no adaptation of the parameter σ is necessary in order to get good decoding results with the sum-product formulation.

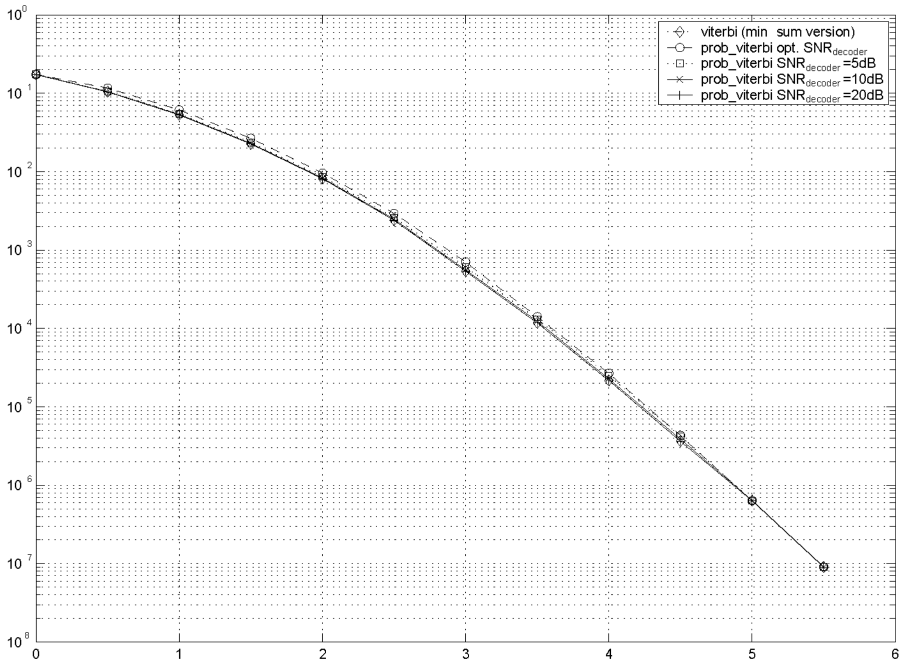


Figure 5.43 *Comparison of simulation results of a Viterbi decoder for a rate $R = 1/2$ [171,133] convolutional code using a traditional ACS and a Viterbi decoder using the equivalent SPS unit. No difference is visible between the min-sum Viterbi decoder and the probability-based Viterbi decoder if $\sigma < 0.1$ ($\mathrm{SNR}_{\mathrm{decoder}} > 20$ dB).*

Viterbi decoder using sum-product-select modules

A new probability-propagation-based decoder architecture

In the following subsection we will postulate a new Viterbi decoder architecture that uses our probability propagation modules to do the path metric calculations. Depending on the definition of the branch metrics, the ACS unit of a traditional Viterbi decoder performs an operation known as the min-sum calculus (or the max-sum calculus) in the context of iterative decoding [149]. But as we have seen before, we may change the definition of the branch metrics from the *squared Euclidean distance* measure to a *conditional probability* measure. By doing this, the min-sum calculus can easily be transformed into the sum-product calculus.

The block diagram level of the Viterbi decoder

Following the notation in [37], Fig. 5.44 shows the simplified block diagrams for both a traditional Viterbi decoder and the newly proposed probabilistic Viterbi decoder. The branch computation unit (BMC) computes the appropriate branch metrics with respect to the received symbols. These metrics are passed either to the ACS or the sum-product-select (SPS) unit. The processed output (1-bit digital information) is then passed to the storage-survivor-memory (SSM). This memory keeps track of the decisions made by all ACS units or the SPS units, respectively, and traces back to find the most probably sent sequence of information bits. The SSM is strictly the same circuit for both versions of the analog Viterbi decoder.

Today mostly voltage-mode based implementations

To date, most of the analog Viterbi decoder implementations are based on voltage-mode circuits such as switched-capacitor circuits. To our knowledge, current-mode implementations of Viterbi decoders have just started to emerge. A first complete current-mode implementation using switched-current (SI) memory cells in the ACS loop and state-vector renormalization was presented by Demosthenous and Taylor [37]. However, the decoder can be simplified and thus speeded up in many ways by applying the sum-product calculus as in our approach.

5.4.2 Proposed Implementation

The proposed circuit implementation

A probabilistic Viterbi decoder core for a rate $R = 1/2$ convolutional code with constraint length K using SPS units instead of the traditional ACS units is shown in Fig. 5.45. As

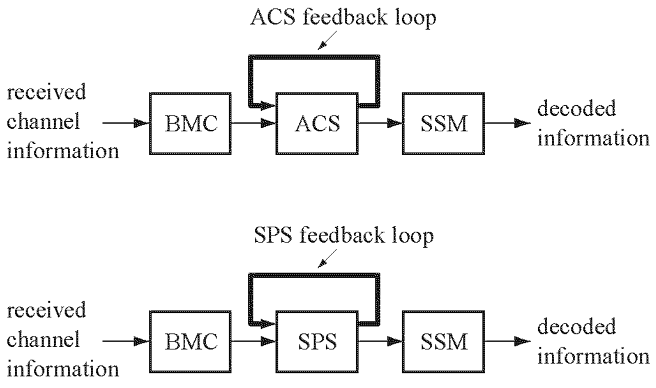


Figure 5.44 *Simplified block diagram of a traditional analog Viterbi decoder (top) and a probabilistic analog viterbi decoder with building blocks of Fig. 3.13 in the SPS unit (bottom).*

opposed to the ACS circuit implementation of [37], this circuit implementation is inherently immune to branch-metric-representation overflows or underflows, since the branch metric signal-currents represent probability distributions whose sum-current vector is bounded. A renormalization of small currents is also inherent to the circuit topology of the basic sum-product building block as presented in Chapter 4.

Discussion of the circuit

At the core of the SPS unit we find the branch-metric multiplier (Q_{11} to Q_{14}) consisting of only one bipolar transistor (or one MOS transistor in weak inversion) per element-wise multiplication of two discrete probability distributions. The dashed transistors Q_{13} and Q_{14} are present due to symmetry considerations and do not contribute to the output signal, as described in Chapter 4. A small overhead is added by the extra transistors at the input doing the logarithmic compression of the input probability distribution. For practical binary cases with a constraint length $K \geq 7$ and rate $R = 1/2$, the overhead of four input transistors is negligible compared to the 2^{K+1} transistors needed for the multiplying part. The bases of Q_{11} to Q_{14} are connected (shown as dashed lines in Fig. 5.45) to one of the logarithmically compressed input metrics according to the edge label of the trellis diagram. Before adding the two corresponding paths of the trellis diagram, the product currents are copied twice by the pMOS transistors M_{10} to M_{24}. One copy is used for the

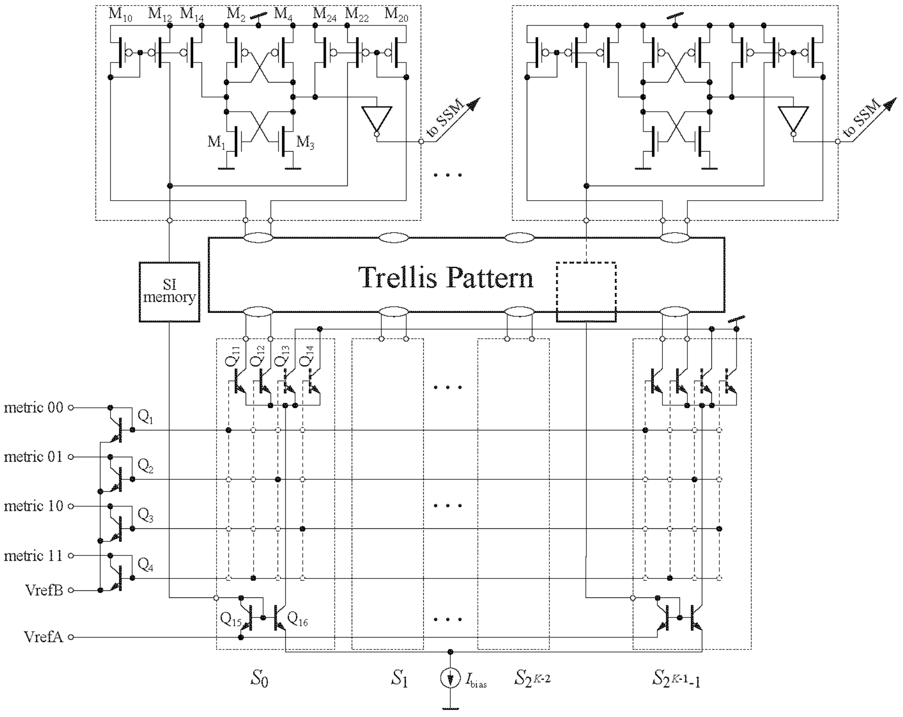


Figure 5.45 *Circuit implementation of a 2^{K-1}-state SPS unit for binary symbols including its feedback loop.*

current summation, and one copy is fed into the current comparator consisting of transistors M_1 to M_4. The result of this comparison is then passed to the SSM unit by a digital inverter where it is stored and used for the survivor-path trace-back process. The sum current of each state is stored in a ping-pong SI memory cell to form the feedback loop of the SPS unit (drawn as blackbox in Fig. 5.45). Transistors Q_{15} and Q_{16} – together with the remaining 2^{K-2} sections for each state – form a renormalization circuit. The sum current of all states together is thus defined by I_{bias} and affects the speed of the whole decoder.

CMOS implementation possible

Although a BiCMOS process is used for the probabilistic analog Viterbi decoder, the architecture is very attractive for high- and highest-speed implementations of such decoders, since the circuits have fewer transistors in the data path than other analog Viterbi decoder implementations. Speed can be adapted in a wide range by changing the bias current. On the other hand, an ordinary CMOS process can be used to build an ultra-low-power analog Viterbi decoder with transistors operating in weak inversion. In this case, all bipolar NPN transistors are replaced by nMOS transistors.

Precision considerations

For the design of the circuit, we have to consider the same precision requirements as in the case of a digital implementation. A resolution of 5 to 7 bits on the log-likelihood level, i.e., for the squared Euclidean distances, is generally sufficient for digital circuit implementations. Smaller quantization of the data induces noticeable losses in the coding gain. Having in mind the precision discussion of Section 4.4.1, a design fulfilling these specs is feasible even with relatively small transistor sizes. However, systematic errors should be avoided by doing a correct layout of the circuit chip. This point is even more important, since we reuse the same computation section all the time.

5.5 High-Level Study of Plain CMOS Implementations

CMOS drives semiconductor technology

Today's semiconductor technology is driven by digital applications in CMOS. Hence, BiCMOS technology lags by at least one and a half generations behind the leading CMOS technologies. For heavily parallel analog circuit applications, the minimal feature size, which is the key parameter for digital applictions, is not always the most important parameter. Wiring density in all three geometrical dimensions is equally or even more important for complex connection patterns. A second argument for using CMOS technology instead of BiCMOS technology is its lower production cost. For economic reasons, everything that can be done in CMOS will be implemented in this technology.

We would like to implement our circuits in CMOS technology

For all the practical and economical reasons mentioned above, it will be advantageous to implement the circuits of the considered analog probability propagation networks in CMOS technology. Weak inversion operation of the MOS transistors enables the direct implementation of the circuits, but by nature they are relatively slow due to the low current densities in the transistors. Raising the current level does not help, since widening the transistors to enable weak-inversion operation at a higher current level increases the parasitic gate-source capacitances proportionally and the speed measure g_{m}/C remains almost at the same level. However, with the transistor sizes of today's advanced CMOS technologies rapidly shrinking towards the sub-0.1 μm range, the transistors are operating more and more in moderate or even weak inversion even for high-speed operation. We will support this claim briefly after having introduced a very simple MOS model which is continuous from weak inversion to strong inversion.

Redundancy improves decoding performance

In a second subsection we will introduce the concept of different factor-graph representations and thus different decoder architectures for the same underlying code. By doing this we will get code descriptions with redundant equations. By explicitly constructing code descriptions with over-constrained equation sets we will get better decoding performance if we use quadratic-law MOS tranistors in our probability calculation modules. Part of the work presented in this subsection was done in two diploma projects under our supervision: Moser [150] investigated the $[8,4,4]$ Hamming code and its different realizations and Fromherz and Schinca [151] implemented the continuous MOS model in the high-level simulation tool.

5.5.1 Continuous CMOS Model from Weak to Strong Inversion

Continuously from exponential to quadratic behaviour

The operational regions of exponential behaviour in weak inversion and the quadratic behaviour of strong inversion in CMOS circuits are not separated abruptly. There is a smooth transition between these two regions of operation. Thus, it will be interesting to see by how much the bit-error rates are degraded if the MOS transistors are operated in moderate or strong inversion. To simulate these effects, we set up an object-oriented high-level simulation environment. All transistors of the core muliplier matrix are in saturation and are modeled according to [109, 152]

$$I_D = 2n\beta U_T^2 \ln^2\left(1+\exp\frac{V_G - nV_S - V_{th}}{2nU_T}\right), \qquad (5.23)$$

where $\beta = \mu C_{ox} W/L$, the slope factor n, and all other symbols have their usual meaning (see also the List of Symbols at the end of this text). The finite output resistance of the MOS transistors and all other non-idealities were neglected. We will use this simple model in the high-level simulations of Section 5.5.3 to compare different decoder implementations.

Towards Weak Inversion with State-of-the-Art CMOS Processes

Definition of the inversion coefficient

The degree of inversion of a single transistor can be determined by the so-called inversion coefficient IC, which was introduced by Vittoz [152]. It describes the ratio between the actual drain current and the specific current, which is the factor before the squared logarithm function of (5.23):

$$\mathrm{IC} = \frac{I_D}{2n\beta U_T^2}. \qquad (5.24)$$

Advanced CMOS processes work more and more in moderate or even weak inversion

For $\mathrm{IC} \ll 1$ the transistors operate in weak inversion; generally $\mathrm{IC} < 0.1$ is enough for a reasonable approximation of the exponential behaviour. By lowering the transistor length L and leaving the transistor width W and the drain current I_D the same, we actually diminish the inversion coeffient proportionally. At the same time, the parasitic gate capcitance C_G is also proportionally reduced. Moreover, the frequency measure g_m/C_G goes up by $1/L^2$ since g_m is assumed directly proportional to I_D in the weak-inversion region. If we want to achieve a given operation frequency and use a more advanced process with smaller minimal feature size L_{min}, we will observe that the inversion coefficients of the transistors will be reduced by a factor $1/L_{min}^3$. So just using a more sophisticated CMOS process pushes the operation of the transistors rapidly towards the weak-inversion region. Hence, advanced CMOS processes will allow the high-speed operation of our building blocks in weak-inversion and thus without any approximations. Additionally, these CMOS processes generally provide many metal layers, which enables very complicated connection patterns both on local and global scale. Thus we may implement very complex systems on one single chip.

5.5.2 Redundant Equations and Code Realizations

In the following, we will introduce the concept of redundant parity-check equations by proposing different realizations of the [8,4,4] extended Hamming code. We will use these code descriptions afterwards for high-level simulations to compare the decoding performance using different operation regions of the MOS transistor.

The [8,4,4] extended Hamming code

The [8,4,4] extended Hamming code is characterized in its basic form by the four equations

$$x_5 = u_1 \oplus u_2 \oplus u_3, \tag{5.25a}$$

$$x_6 = u_1 \oplus u_2 \oplus u_4, \tag{5.25b}$$

$$x_7 = u_1 \oplus u_3 \oplus u_4, \tag{5.25c}$$

$$x_8 = u_2 \oplus u_3 \oplus u_4, \tag{5.25d}$$

where $\oplus$ denotes the addition modulo-2 operation. These parity-check equations encode the dataword $\mathbf{u} = [u_1, u_2, u_3, u_4]$ onto the codeword $\mathbf{x} = [u_1, u_2, u_3, u_4, x_5, x_6, x_7, x_8]$. The code is a so-called systematic code because the databits $u_1, \ldots, u_4$ appear directly in the codeword; $x_5, \ldots x_8$ are called parity bits.

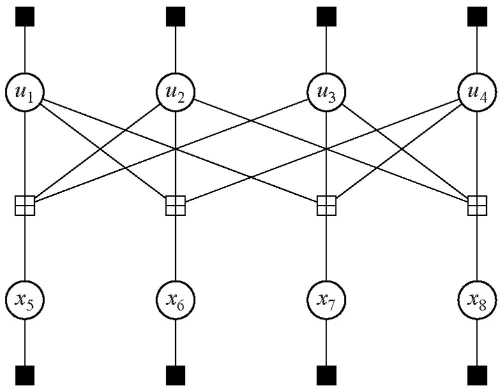


Figure 5.46

Factor graph of the extended [8,4,4] Hamming code in its basic form.

Changing the code realization by introducing state variables

In Fig. 5.46, the basic realization of the [8,4,4] Hamming code is shown. Note that there exist other realizations, i.e., other factor graphs of the same code. They may lead to different decoding performance as we will show in the following subsection. To illustrate these differences, two additional examples are given in Fig. 5.47 and Fig. 5.48. For the second realization, four hidden states s_1 to s_4 are introduced in a tail-biting manner:

$$s_1 = u_3 \oplus u_4, \tag{5.26a}$$
$$s_2 = u_4 \oplus u_1, \tag{5.26b}$$
$$s_3 = u_1 \oplus u_2, \tag{5.26c}$$
$$s_4 = u_2 \oplus u_3. \tag{5.26d}$$

Using these hidden states, the parity-check equations (5.25a) to (5.25d) become

$$x_5 = u_3 \oplus s_3, \tag{5.27a}$$
$$x_6 = u_2 \oplus s_2, \tag{5.27b}$$
$$x_7 = u_1 \oplus s_1, \tag{5.27c}$$
$$x_8 = u_4 \oplus s_4. \tag{5.27d}$$

The factor graph of Fig. 5.47 is a straightforward image of the above eight equations (5.26a) to (5.27d).

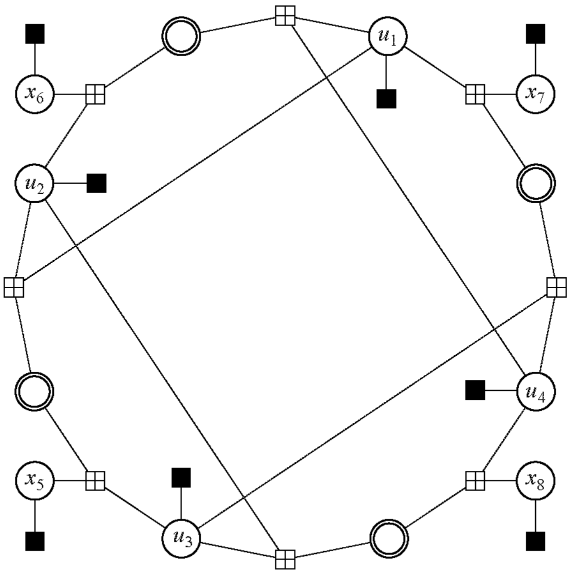


Figure 5.47 *Realization 2 of the extended [8,4,4] Hamming code with four state variable nodes.*

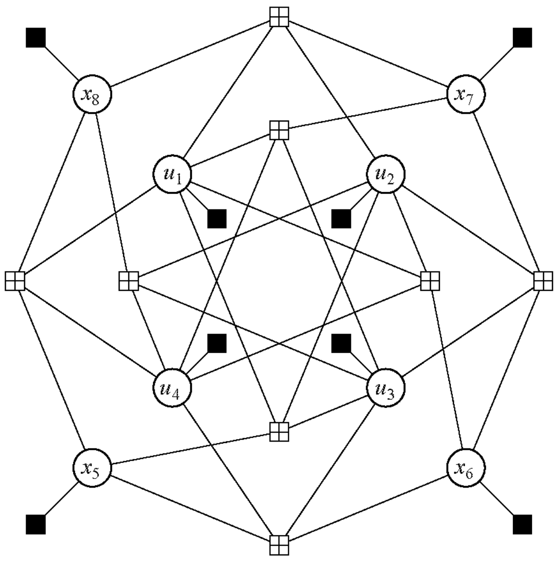


Figure 5.48

Realization 3 of the extended [8,4,4] Hamming code with four redundant parity checks.

Redundant equations for the extended Hamming code

Realization 3 has been obtained by adding four additional parity check equations to the equation set (5.25a) to (5.25d) [150]. These redundant equations add additional complexity in the decoder, but they do not change the code itself. The codeword is still built using the original parity-check equations (5.25a) to (5.25d).

5.5.3 High-Level Simulation Results

CMOS-only decoder for the extended Hamming code

High-level simulations for the extended Hamming code

A first simulation set was run with the three realizations of the [8,4,4] Hamming code presented in the previous subsection [150]. The transistors were designed to operate in weak or in strong inversion. The bits were encoded according to the parity-check equations (5.25a) to (5.25d) and then transmitted over a channel with additive white Gaussian noise (AWGN). The results of Fig. 5.49 indicate clearly that different realiza-

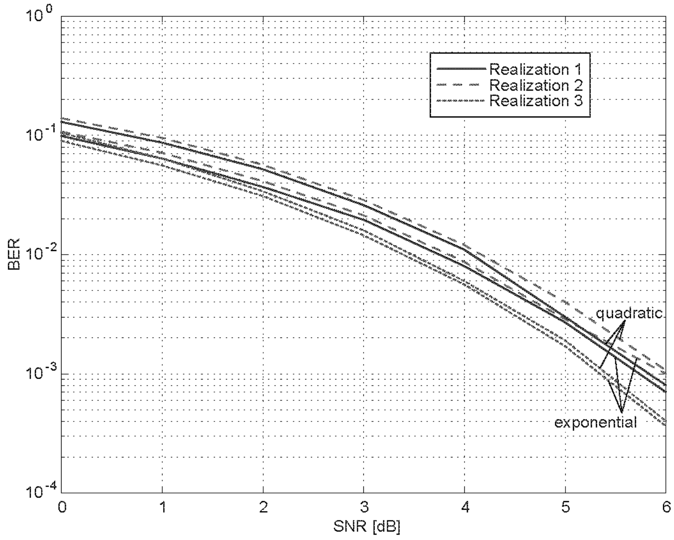


Figure 5.49 *High-level simulations for a (8,4,4) Hamming code where the encoded bits are transmitted over an AWGN channel. Comparison between exponential and quadratic characteristic of MOS transistors.*

tions of the same code behave differently during decoding operation. Realization 3 with four redundant equations performs best, both in weak and strong inversion, but the other realizations also perform rather well in the quadratic (strong inversion) region, with a coding loss of less than 1 dB compared to the ideal exponential characteristic. These results suggest that the use of redundant equations may even help to overcome coding loss when using MOS transistors in the quadratic (strong inversion) region [151].

CMOS-only decoder for the tail-biting trellis code

A slight coding loss for the tail-biting decoder if operated with MOS devices only

A second experiment was conducted using the binary [18,9,5] tail-biting trellis code of Section 5.2. All the NPN BJTs were simply replaced by properly sized nMOS transistors. As is shown in Fig. 5.50, the decoder of Fig. 5.13 operated in the quadratic region loses at most 0.5 dB compared to its ideal ex-

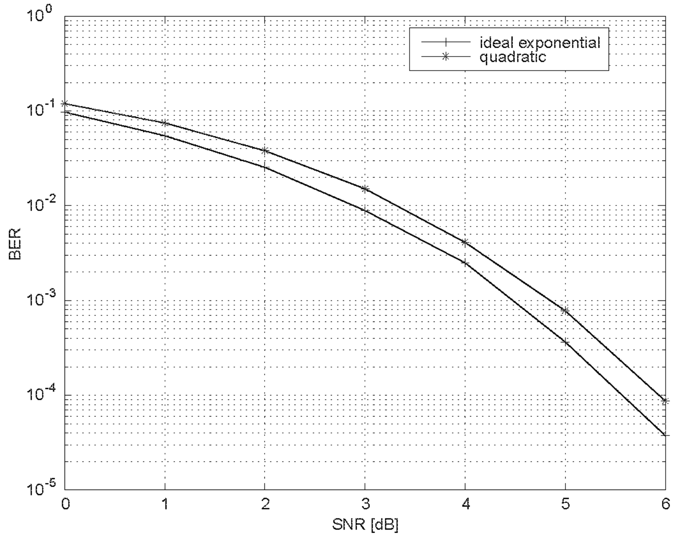


Figure 5.50

High-level simulations for a binary [18,9,5] tail-biting trellis code where the encoded bits are transmitted over an AWGN channel. Comparison between exponential and quadratic characteristic of MOS transistors.

ponential implementation. However, note that no mismatch effects were included so far in this high-level simulation setup. But it is expected that these mismatch-effects only introduce a slight loss in decoding as it was discussed in Section 4.4.1.

Exchanging MOS devices for BJTs can be found in other circuits

Comparable attempts to force circuits, originally designed for bipolar technology, to operate in the quadratic (strong inversion) region of plain CMOS technology have been made. For example, Gilbert's array-normalizing circuit [131] has been used in a fuzzy logic controller to scale current vectors [153]. Although the circuit does not anymore scale the quantities correctly, the order of precedence is preserved in the monotonically increasing transfer function, which is sufficient for building fuzzy logic circuits.

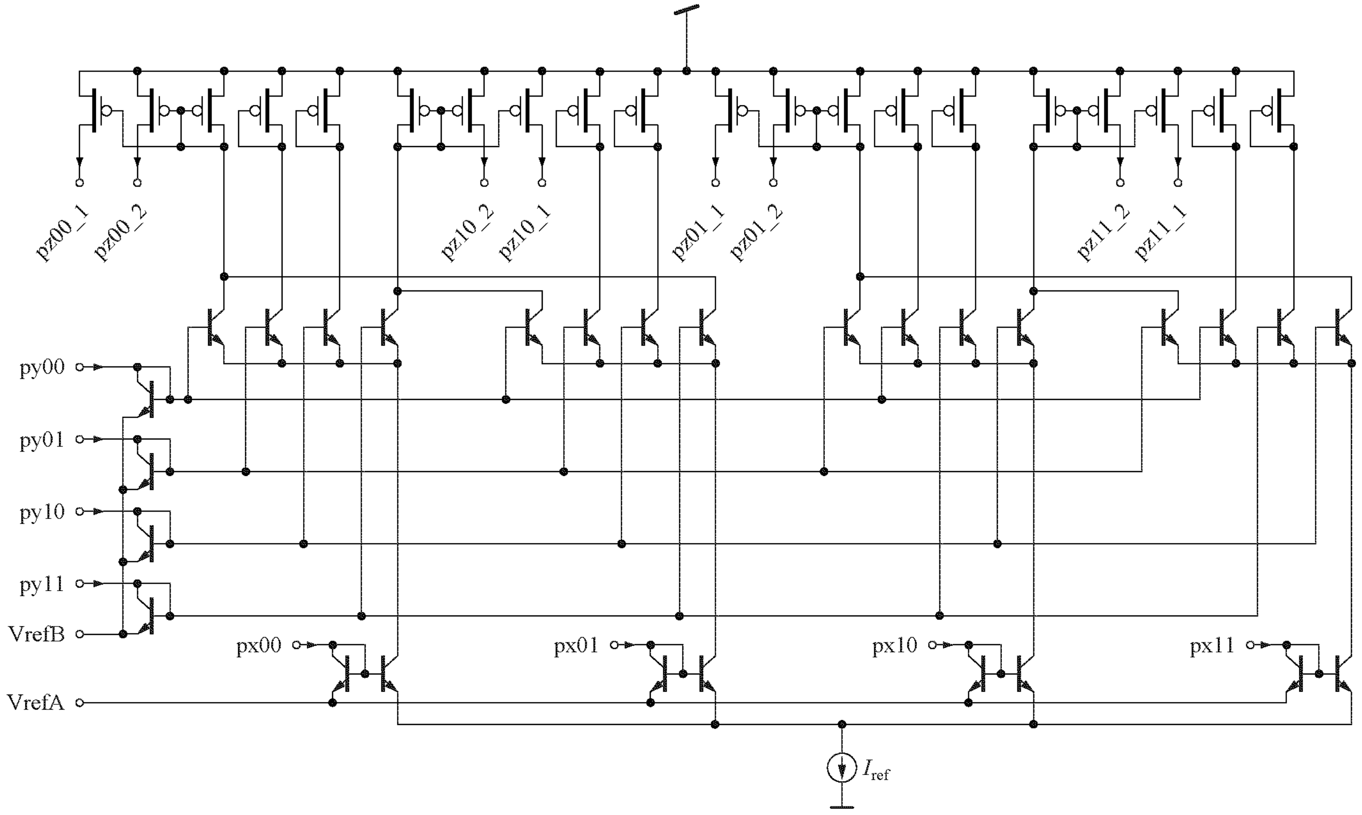


Figure 5.51: *Circuit implementation of the type-B module: forward trellis.*

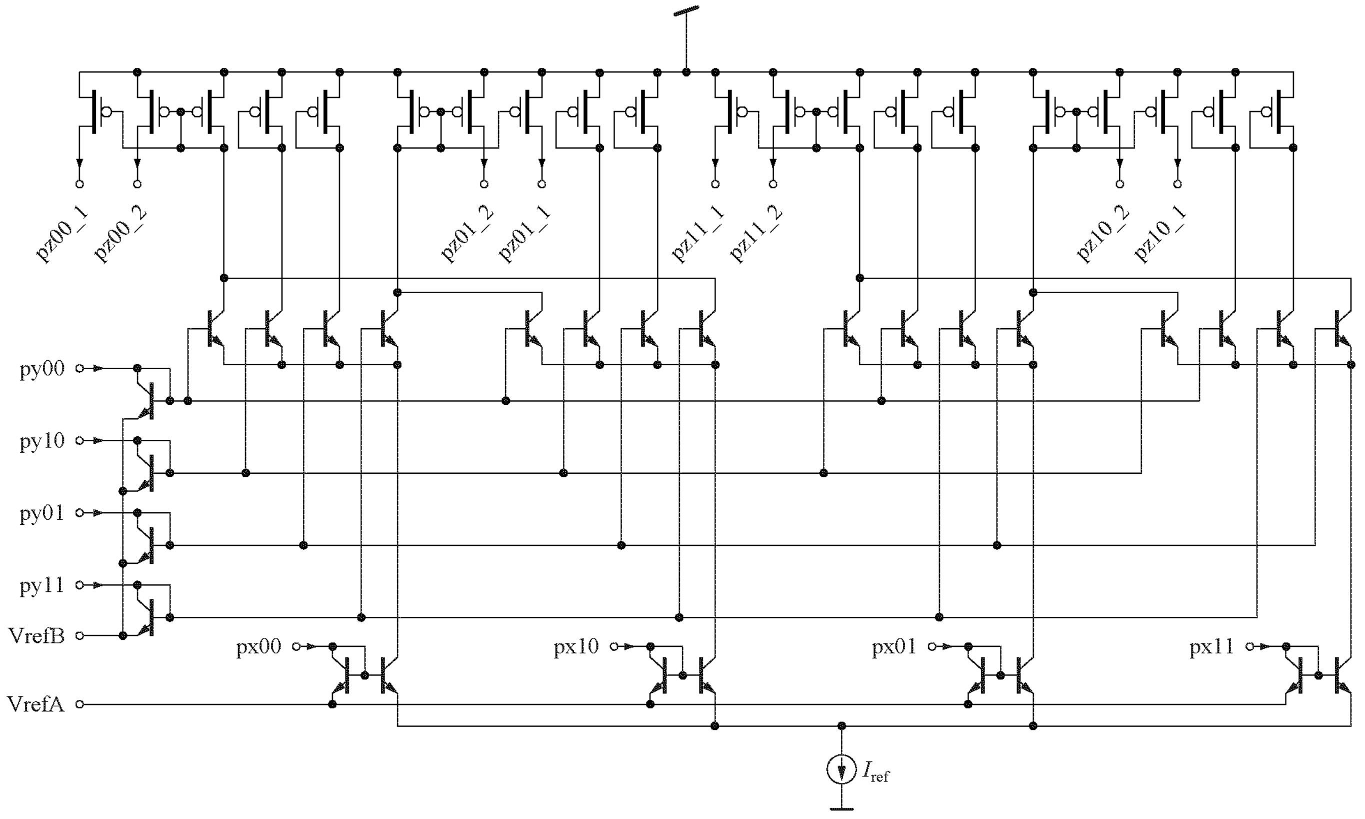


Figure 5.52: *Circuit implementation of the type-C module: backward trellis.*

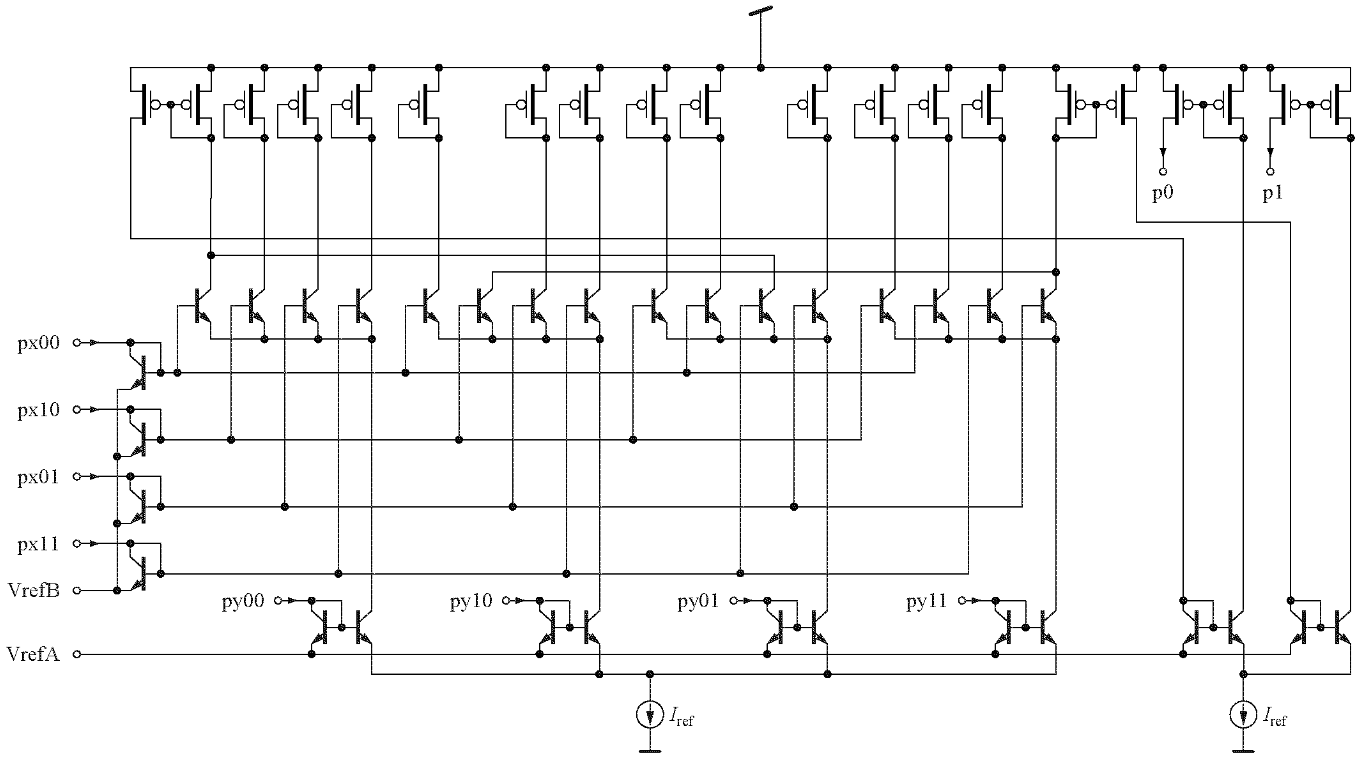
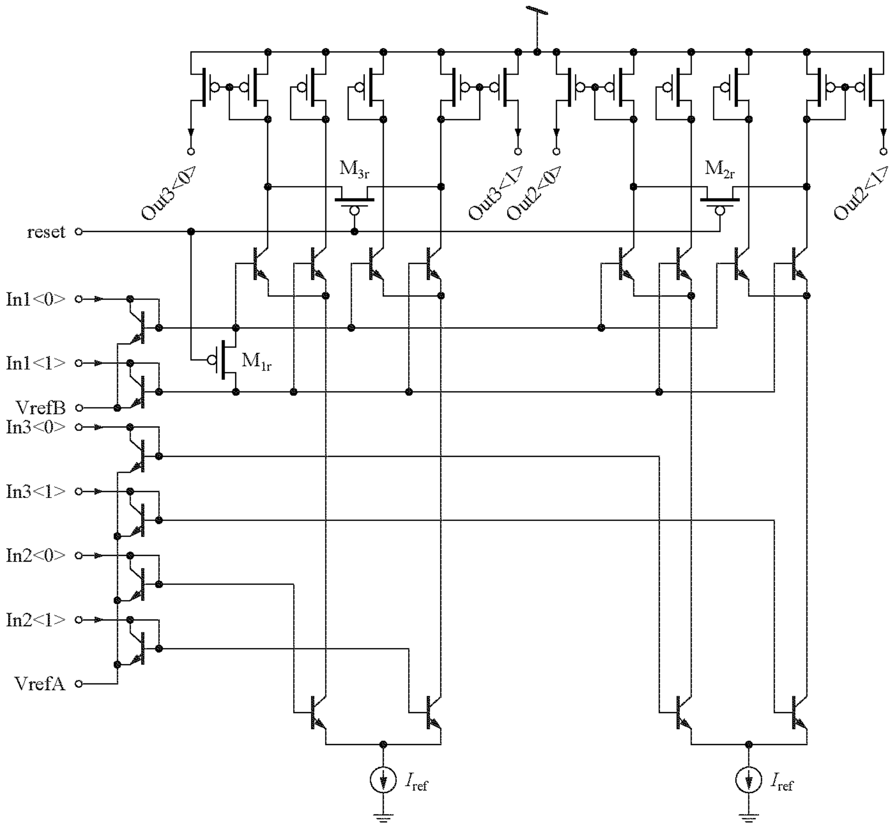


Figure 5.53: *Circuit implementation of the type-D module: final summarization.*

Appendix — Schematics of the Turbine Decoder 5.7



Circuit implementation of the fully bidirectional 2.5-port variable node. Denoted as 'Channel-Bit' in Fig. 5.28. **Figure 5.54**

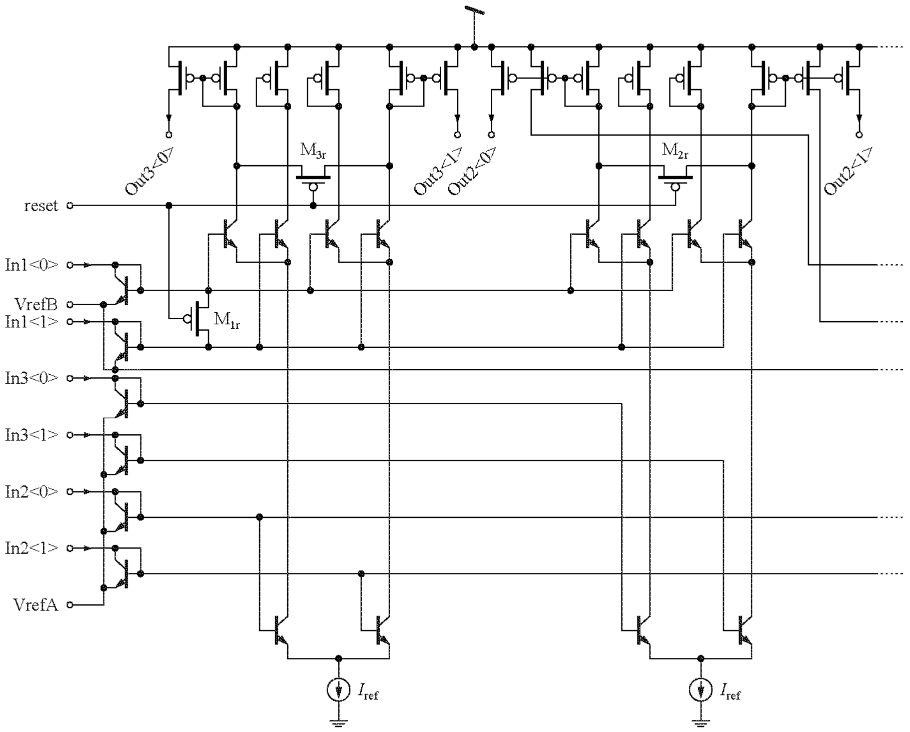


Figure 5.55 *Circuit implementation of the fully bidirectional 2.5-port variable node with output bit slicer. Denoted as 'Info-/Channel-Bit' in Fig. 5.28.*

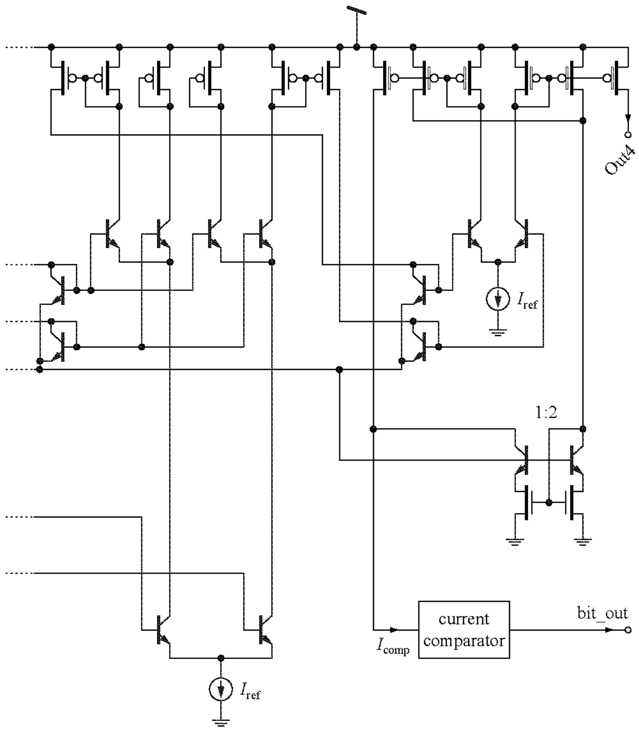


Figure 5.55 *Circuit implementation of the fully bidirectional 2.5-port variable node with output bit slicer. Denoted as 'Info-/Channel-Bit' in Fig. 5.28. A special 'boxed' pMOS-transistor symbol is used for the cascoded transistors on the top right part of the circuit.*

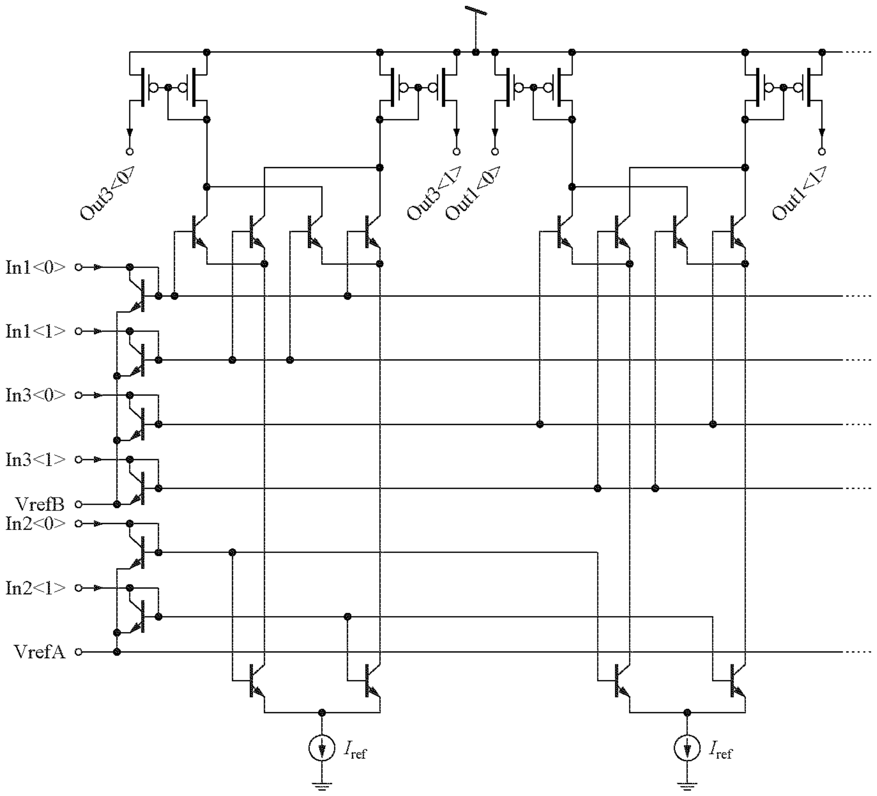
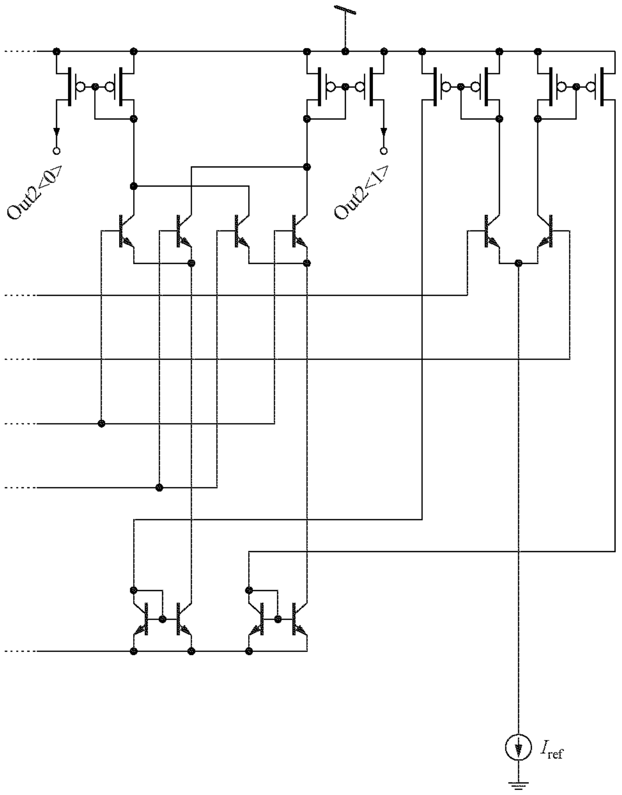


Figure 5.56 *Circuit implementation of the fully bidirectional 3-port soft-XOR node (left part).*



Circuit implementation of the fully bidirectional 3-port soft-XOR node (right part). **Figure 5.56**

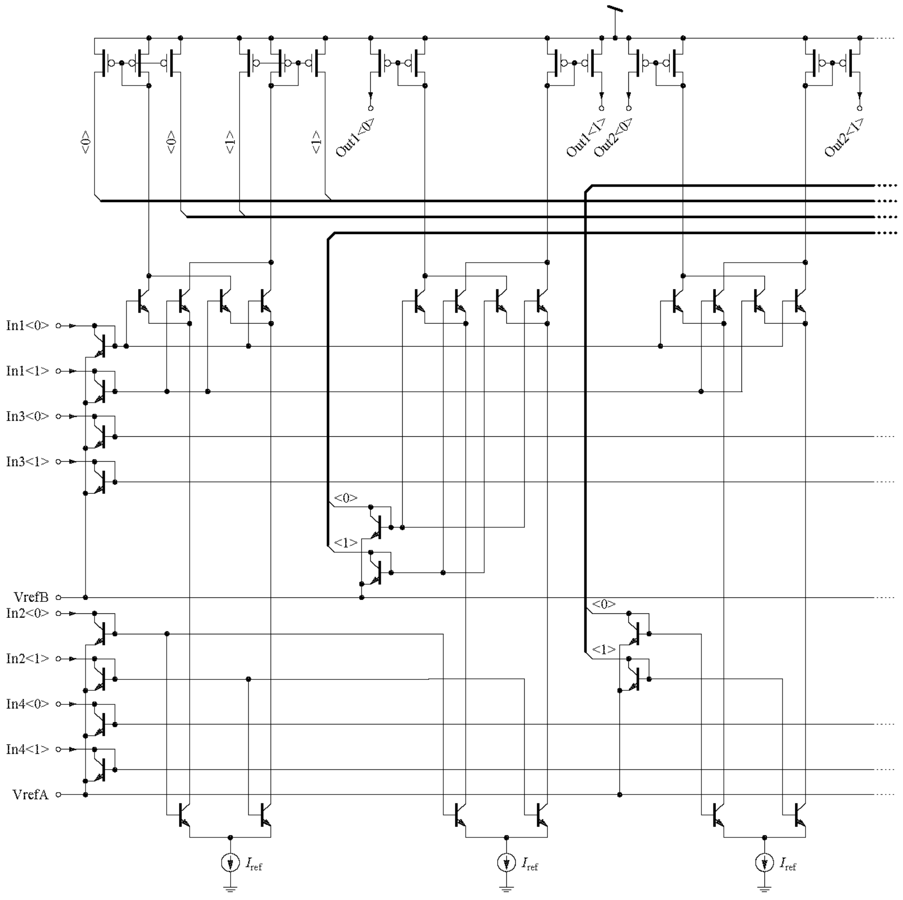
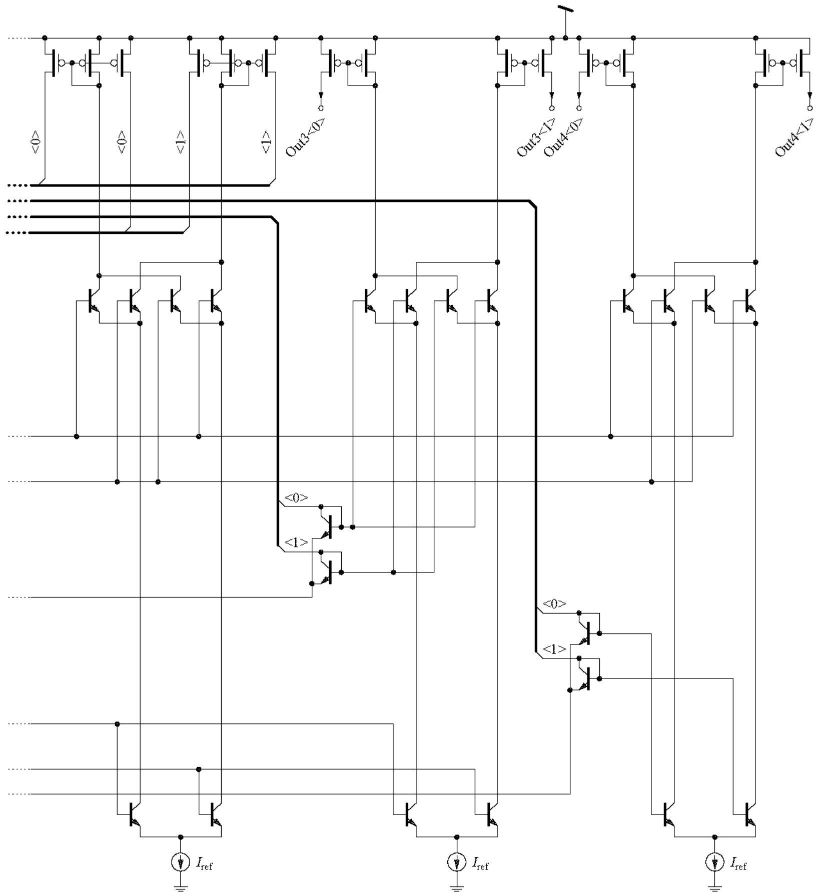


Figure 5.57 *Circuit implementation of the fully bidirectional 4-port soft-XOR node (left part).*



Circuit implementation of the fully bidirectional 4-port soft-XOR node (right part). **Figure 5.57**

Chapter 6

Concluding Remarks

Summary of the Results 6.1

A new technique for an analog sum-product algorithm implementation

In this dissertation we have presented a technique for efficiently implementing the sum-product algorithm (or probability propagation algorithm) in analog VLSI technology. The described new type of analog computing networks exhibits a natural match between probability theory and transistor physics. The elementary modules of which these networks are composed include probabilistic versions of all standard logic gates as well as more general non-binary sum-product modules. The obvious application of such networks is the decoding of error-correcting codes as described in this dissertation. However, any factor graph where all function nodes of degree larger than one are $\{0, 1\}$-valued can be mapped onto such analog networks.

Translinear transistor core of the building blocks

The transistor-level implementation of the building blocks are very simple current-mode vector multipliers and current-mode selective adders that process discrete probability distributions. The core circuits can be interpreted as translinear circuits or log-domain signal processors. Basically one transistor is needed to build the pair-wise product of two elements of discrete probability distributions.

Bio-inspired circuit design approach

The presented networks follow a bio-inspired approach and therefore omit many plagues of traditional analog circuit design such as data-representation overflows, temperature dependence, linear approximations of non-linearities, component-variations, and tedious manual design flows. The circuits exploit rather than fight the inherent non-linearities of the used exponential characteristic of both bipolar junction transistors and weakly inverted MOS transistors. By building large, highly connected networks out of very simple and low-precision computation nodes, a high precision and high processing throughput is reached on the system level. Due to their simplicity and computational efficiency, our analog networks exhibit a distinct advantage in the speed-power-ratio compared to their comparable digital counter-parts. According to our (still limited) experience, this advantage amounts to at least two orders of magnitude.

Automated analog design-flow

We have also presented a design methodology that allows a direct mapping of the parity-check-matrix description of a given code to the factor-graph representation and further on to the structural description of the decoding network. It is based on the standard digital design-flow of chip-design environments such as is available in Cadence design tools. By automatically constructing the decoder circuits rather than doing a full-custom design by hand, we circumvent tedious manual verifications and fundamentally speed up the design process. This also opens the prospect of fabricating large first-time-right decoder systems.

Practical implementations perform well

The practical decoder examples presented in this dissertation showed a system-level behaviour that is remarkably robust against all sorts of non-idealities even for the discrete transistor-device implementation. The measured results also showed a close agreement with the transistor-level simulations. Furthermore, we verified many theoretical design aspects with these practical implementations. Finally, the high-level design studies are a rich source of ideas for future work on this subject.

Probablity-propagation networks are also useful for other applications

Beside the decoding applications, which are the main subject of this dissertation, the probability propagation networks may be applied in various other related domains such as the tracking of hidden-Markov models, widely used for many pattern recognition tasks, and the inference on Bayesian networks, which appear in the context of artificial intelligence problems.

6.2 Ideas for Further Work and Outlook

To round off this dissertation, we briefly describe a few open issues and ideas for future research. They mainly represent the thoughts of the author where future work might go to.

Decoder for a full-size LDPC code. So far, only relatively small decoding networks were implemented. They incorporate only some ten or hundred factor graph nodes. However, for a good BER performance, the length of a code needs to be quite large. Generally, a code with a length of some thousand bits per codeword will already be sufficient for a practical application. Hence, we need to construct larger decoding networks, which gives rise to many issues such as the exacerbating simulation times and the testability of such networks. These problems have not been investigated in depth yet, but they will be of increasing importance for future designs.

Full CMOS implementation. The chip implementations described in this dissertation rely on the exponential characteristic of BJTs. However, it is economically and technically interesting to use standard CMOS processes. As we have seen in Section 5.5, the advanced CMOS processes shift the operation region of the transistors more and more towards moderate inversion or even weak inversion. This opens the new prospect of implementing high-speed analog decoders in CMOS technology only.

Use of redundant code equations. A second possibility for purely CMOS analog decoders is the use of redundant equations in the code description. As we have seen in the high-level simulation results, this direction is very promising since we do not have to rely on the most advanced CMOS technologies.

Application of the probability-propagation calculus to other problems. In this dissertation we have only described decoding networks. However, we have seen that many problems can be described by factor graphs which in turn can be directly converted to an analog probability-propagation network. It would be very interesting to apply the design technique to other application fields such as artificial-intelligence problems that might appear in on-line fault-detection circuits of complex systems.

Adaptive filters. By changing the signal representation from a probability-based interpretation to a real-valued interpretation, the well-known equal gate and soft-XOR gate may be operated as real-value adders and real-value multipliers, respectively [154]. Hence, they represent the basic operations of discrete-time filters. By making the filter taps adaptive, we could easily build adaptive FIR and IIR filters. Adaptive FIR filters are commonly used for equalizing wire-line channels.

Joint channel-equalizer/-decoder. In the communications community, there exist several concepts of jointly equalizing a given channel and decoding the transmission code. But all of them work in the digital domain and it thus may make no sense to do decoding in the analog way, whereas the remaining part of the receiver works digitally. So why not build most of the receiver front-end using our analog probability networks? For example, the decision-feed-back equalizer (DFE) is a good candidate for an analog network implementation, since all the basic operations can be implemented using our generic building blocks. By doing so, we get one step closer to the antenna or the line interface of a data communication system without flipping too much back and forth between analog to digital.

All-analog receiver system. Our experience so far is that many individual blocks of a receiver system can be implemented in analog electronics. Despite the fact that many renowned researchers postulate software radio, i.e., a system that consists merely of an A/D converter as close as possible to the antenna and digital processors for signal processing, we think that for certain demanding applications analog signal processing in an intelligent manner is the way to go. Our long-term aim is an all-analog receiver system, i.e., to have no digital signals before the decoder block, since the analog decoder can make an inherent A/D conversion. This would potentially provide very efficient highest-speed and ultra-low-power communications systems as needed by today's *e*-society.

Appendix A

Selected Circuit Structures

Transistor Terminals and Voltages A.1

In order to facilitate the analysis of a given circuit and to prevent from sign problems in circuits using complementary transistors, we define all voltages as positive values for the normal active region. In the case of a bipolar process the voltages of the transistor terminals are defined with the rails as their reference as is shown in Fig. A.1. For the case of MOS transistors, which are in principle symmetrical structures, we define the voltages with respect to the potential of the well or substrate. Hence, the devices are four-terminal devices. But often, we draw only the three main terminals drain, gate, and source. In the case of an n-well technology, the substrate (which is also called bulk) is p-doped semiconductor material an thus is generally tied to the negative rail to prevent forward biased pn-junctions. Conversely, the n-doped substrate of a p-well technology is generally connected to the positive supply rail. Since the terminal voltages of a MOS transistor are bulk-referenced, the functionality of a given terminal is only defined by the voltage level. The current leading terminal with the smaller voltage becomes the source of the transistor whereas the other terminal becomes the drain.

For the equations describing the behaviour of the semiconductor devices, consult one of the excellent textbooks that exist today: for MOS tranistors see [109, 155] and for BJT [138, 156–158]. The references [109, 156–158] are very device physics oriented, whereas the remaining books are more design oriented.

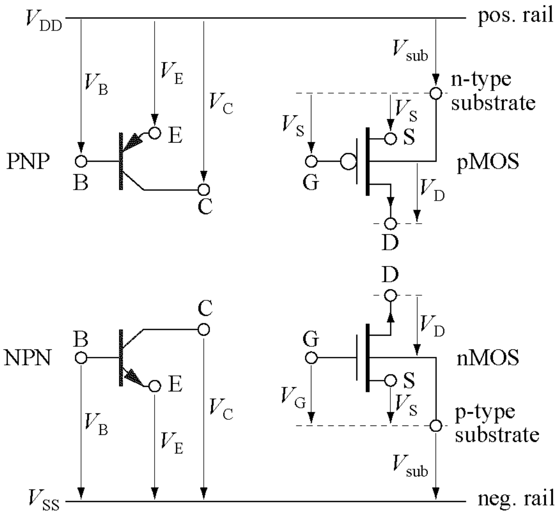


Figure A.1 *The definition of the terminals and its corresponding voltages for both BJT (on the left side) and CMOS (on the right side).*

A.2 Cascode Structures

A composite 'super transistor'

The output resistance of a single saturated MOS transistor (or forward-biased transistor in the case of a BJT) is generally fairly limited. In the case of a MOS transistor it is directly proportional to the transistor length L. In order to reduce the effect of a drain-current change due to drain-source-voltage variations, the length L has to be made fairly long for practical applications. Hence, the maximum operation frequency is dimished by the same amount. A simple solution to that problem is the so-called cascode structure. The many available cascode structures lead to 'super transistors' with improved output resistance. The composite transistors can be used in replacement for any transistor of a given circuit. The symbol of such a super transistor, that we have used for our purpose, is shown in Fig. A.2a). Many different forms of cascode circuits exist [108, 130, 135, 137, 138]. In this context we cite just two: the simple cascode structure is shown in Fig. A.2b) and the regulated cascode structure in Fig. A.2c). All the reasoning that has been advanced so far for MOS transistors is also valid for BJTs, with only slight differences in the calculations.

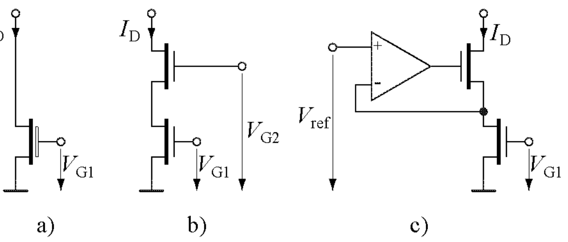


Figure A.2 *Cascoded nMOS transistors: a) the used symbol for a cascoded nMOS transistor, b) a simple cascode structure, and c) a regulated cascode structure.*

A simple cascode structure

The simple cascode consists of only two transistors: the main transistor below and the stacked cascode transistor. This cascode transistor stabilizes the drain voltage of the main transistor. The output resistance of the main transistor is multiplied by the transconductance of the cascode transistor times the output resistance of this same transistor. Hence, by still having fairly small transistor sizes, the output resistance of the composite transistor is considerably augmented. The transistors are designed such that they both operate in saturation. The biasing voltage V_{G2} is chosen according to the maximum drain current and kept constant in time.

The regulated cascode structure

For high-precision circuits, the simple cascode structure may not raise the output resistance sufficiently. One solution to this problem consists of stacking more than one cascode stage. However, this also raises the minimum supply voltage requirements drastically. A much more intelligent solution to that problem was proposed by Säckinger *et al.* [135]. The so-called regulated cascode structure actively controls the drain voltage of the main transistor. By doing so, much higher output resistances can be achieved. However, the operational amplifier adds circuitry overhead. The most simple implementation of this amplifier would consist of only one source follower transistor. But generally more sophisticated implementations of the amplifier have to be chosen. Examples for such implementations can be found in [41] and the references therein.

A.3 Current Mirrors

Current mirrors are a key element in analog integrated circuits. They are used to duplicate currents or to fold or cascade parts of the circuit in order to reduce the supply-voltage requirements. All the structures described below can equally well be implemented with bipolar transistors. The dimensioning of the transistors is only slightly different. An overview of different current mirror structures can be found in [108, 130, 137, 138].

The most simple current mirror consists of just two transistors

The most simple structure of an nMOS current mirror is shown in Fig. A.3a). Both transistors have to be in saturation and we rely on perfect matching for an ideal operation. A brief analysis shows that the copy errors due to the finite output resistance of the mirror transistors are relatively large [130, 137, 138]. The output resistance is improved by making the tranistors longer. Note that the current mirror can be operated in both strong inversion or weak inversion.[1] However, a simple anlysis shows that current mirrors operated in strong inversion match better than those operated in weak inversion [142, 152]. For a given WL, where W is the width of the transistor, the matching of the transistors is best if the current mirror is designed to operate with a large V_{GS}, i.e., to force the transistor to deep strong inversion. Matching can be improved by augmenting the active transistor area WL. This reduces the relative errors of random fabrication errors.

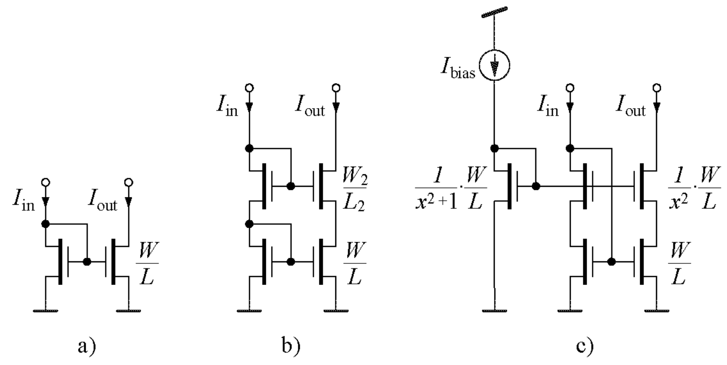
Design of the simple current mirror

The design of a current mirror is generally started by imposing a certain voltage swing at the nominal current. This leads to a W/L in a given semiconductor technology. The minimum voltage between drain and source that still allows the operation in the saturated region can be derived from the given gate-source voltage. Finally, the active transistor area WL is adapted until the desired level of matching is achieved. Note, however, that the parasitic gate capacitance is augmented by the same amount. Hence, the increase of the parasitic capacitance will reduce the maximum operation speed. Generally, several parameters have to be traded off during the design process.

Cascoded current mirror

Beside raising the transitor length to augment the output re-

[1]The degree of saturation and the degree of inversion are two orthogonal axes of the operational modes of MOS transistors. We can freely choose one of the four quadrants that is best suited for the purpose of the transistor in the circuit



Different nMOS current mirrors: a) the most simple current mirror, b) a simple cascoded current mirror, and c) a low-voltage version of a cascoded current mirror.

Figure A.3

sistance of the mirror transistors, we could apply the cascode techniques described in Appendix A.2. This solution is shown in Fig. A.3b). The diode connected transistors are always in saturation, thus no special attention has to be paid to that part of the circuit. The transistors of the output section are designed such that the transistors are saturated for the nominal current. Although very simple and self-biasing, a drawback of the circuit are two stacked diode-connected transistors. They limit the minimal supply voltage for a correct operation heavily.

Low-voltage version of the cascoded current mirror

A solution for low-voltage operation is shown in Fig. A.3c) [136]. Compared to the simple cascoded current mirror, we omit the threshold voltage of the cascode transistor. However, we need a special biasing circuit which is fortunately very simple. The sizing of the biasing transistor, which is located on the left side of the low-voltage current mirror schematic, is according to Fig. A.3c). Again, all transistors are dimensioned to operate in the saturation region.

List of Abbreviations

A/D	Analog-to-Digital
ACS	Add-Compare-Select; computation unit of Viterbi decoders
APP	*a posteriori* Probability
BCH	algebraic code type named after the inventors Bose, Chaudhuri, and Hocquenghem
BCJR	decoding algorithm named after the inventors Bahl, Cocke, Jelinek, and Raviv
BER	Bit Error Rate
BiCMOS	Bipolar and CMOS transistors in the same silicon process
BJT	Bipolar Junction Transistor
BMC	Branch Metric Computation; computation unit of the Viterbi decoder
BSC	Binary Symmetric Channel
CAD	Computer-Aided Drawing
CAE	Computer-Aided Engineering
CMOS	Complementary MOS
COB	Chip-On-Board; mounting technology for integrated circuits directly on a PCB
D/A	Digital-to-Analog
DFE	Decision-Feedback Equalizer
DMC	Discrete Memoryless Channel
DR	Dynamic Range
ESD	ElectroStatic Discharge
FBA	Forward-Backward Algorithm
FIR	Finite Impulse Response; see also IIR
HMM	Hidden Markov Model
I/V	current-to-voltage conversion
IC	Integrated Circuit
IIR	Infinite Impulse Response; see also FIR
JTAG	Joint Test Action Group. This group created the IEEE 1149.1 standard defining test access ports and boundary scans
LDPC	Low-Density Parity-Check
LDS	Linear Discrete-time System
LED	Light Emitting Diode
MAP	Maximium *a posteriori* Probability

MOS	Metal–Oxide–Semicondictor
MOSFET	Metal–Oxide–Semicondictor Field-Effect Transistor
P&R	Place-and-Route; tool in the digital chip-design flow
PCB	Printed Circuit Board
PSD	Power Spectral Density
QCRA	Quasi-Cyclic Repeat-Accumulate; LDPC code type
RA	Repeat-Accumulate; LDPC code type
SC	Switched-Capacitor
S/H	Sample-and-Hold
SI	Switched-Current
SNR	Signal-to-Noise Ratio
SPS	Sum-Product-Select; computation unit of a MAP Viterbi decoder using probability propagation modules. See also ACS
SSM	Storage Survivor Memory; unit in Viterbi decoders
TA	Transconductance Amplifier
TL	Translinear Loop
TN	Translinear Network
V/I	voltage-to-current conversion
VI	voltage-current
VLSI	Very Large Scale Integration

List of Symbols

Symbols related to coding

Roman Symbols

$\mathbf{u}; u_i$	Uncoded data vector; uncoded bit number i
$\hat{\mathbf{u}}; \hat{u}_i$	Output of the decoder; decoded bit number i. Estimate of $\mathbf{u}$, u_i
$\mathbf{x}; x_i$	Coded data vector; coded bit number i
$\mathbf{y}; y_i$	Received data vector; received bit number i
$s(t)$	Input of the physical channel at time t
$r(t)$	Output of the physical channel at time t
b	Bit
C	(i) Code; (ii) Channel capacity
d	Hamming distance
E_b	Energy of an information bit
E_c	Energy of a channel bit (chip)
F^k	k-dimensional vector space over the field F
g	Girth of the graph; smallest number of branches to form a closed loop in a factor graph
$\mathbf{G}$	Generator matrix of a block code
GF(q)	Galois-field with q elements
$\mathbf{H}$	Parity-check matrix of a block code
K	Constraint length of a convolutional code
m	(i) Memory order of a convolutional encoder; (ii) size of alphabet of X or Y
n	size of X or Y alphabet
N_0	One-sided noise power spectral density
$p_X(x); p(x)$	Probability of an actual realization x of the random variable X; short-hand notation of the same probability
$\tilde{p}(x)$	Approximate probability of x
$Q(x)$	Q-function of the Gaussian statistic
R	Code rate
T_s	Symbol duration; sampling interval

Greek Symbols

ϵ	Transition probability of binary symmetric channel
λ	Squared Euclidean path-metric in the Viterbi algorithm
$\pi(.)$	Permutation; interleaver of a Turbo code
σ_n^2	Variance of white Gaussian noise
μ	Probabilistic path-metric in the Viterbi algorithm
$\mu_i(b)$	Short-hand notation for $p(y_i \| x_i = b)$
$\mu_{x\to f}$	Sum-product message from a variable node x to a function node f
$\mu_{f\to x}$	Sum-product message from a function node f to a variable node x
$\boldsymbol{\rho}$	Sampled outputs of the matched filter detector
ξ_k	State transition at time k

Symbols related to electronics

Roman Symbols

A	Active area
A_{E}	Active emitter area of a BJT
C; C_i	Capacitor; Capacitor number i
C_{ox}	Unit oxide capacitance of a MOS transistor
f	Frequency
g_{m}	Transconductance of a transistor
i	(i) Small-signal current; (ii) Index
I	Large-signal current
I_{C}	Collector current of a BJT
I_{D}	Drain current of a MOS transistor
I_0	Specific current of a MOS transistor
I_S	Specific current of a BJT
J_0	Specific current density of a MOS transistor
J_S	Specific current density of a BJT
I_{ref}	Reference current
IC	Inversion coefficient of the channel of a MOS transistor
L	Length of a transistor
L_{min}	Minimum feature size (minimum channel length) of a MOS transistor

M; M_i	MOS transistor; MOS transistor number i
n	(i) Slope factor of MOS; (ii) Emission coefficient of a BJT; (iii) index
P	Power dissipation
Q; Q_i	Bipolar junction transistor; BJT number i
R; R_i	Resistor; Resistor number i
s	Thickness
t	Time
T	Absolute Temperature
v	Small-signal voltage
V	Large-signal voltage
V_{BE}	Base-emitter voltage of a BJT
V_{D}	Drain voltage of a MOS transistor
V_{Dsat}	Drain voltage to achieve saturation in the drain current
V_{G}	Gate voltage of a MOS transistor
V_{S}	Source voltage of a MOS transistor
V_{th}	Threshold voltage of a MOS transistor
W	Transistor width
U_T	Thermal voltage kT/q; 25.9 mV at 300 K

Greek Symbols

β	(i) Current gain of a BJT; (ii) Transfer parameter of a MOS transistor
$\varepsilon, \varepsilon_{\text{rel}}$	Relative current error
λ	(i) Channel-length modulation factor; (ii) Thermal conductivity coefficient
μ	Carrier mobility

Bibliography

C. E. Shannon. “A mathematical theory of communication.” *Bell Systems Technical Journal*, vol. 27, pp. 379–423 (part I), 623–656 (part II), July 1948. [1]

C. Berrou, A. Glavieux, and P. Thitimajshima. “Near Shannon-limit error-correcting coding and decoding: turbo codes.” In *Proceedings of the International Conference on Communications*, pp. 1064–1070. Geneva, May 1993. [2]

S. Benedetto and G. Montorsi. “Unveiling turbo codes: some results on parallel concatenated coding schemes.” *IEEE Transactions on Information Theory*, vol. 42, pp. 409–428, March 1996. [3]

S. Benedetto and G. Montorsi. “Iterative decoding of serially concatenated convolutional codes.” *Electronics Letters*, vol. 32, pp. 1186–1188, June 1996. [4]

R. G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, 1963. [5]

D. J. C. MacKay and R. M. Neal. “Good codes based on very sparse matrices.” In *Cryptography and Coding. 5th IMA Conference* (edited by C. Boyd), no. 1025 in Lecture Notes in Computer Science, pp. 100–111. Springer, 1995. [6]

R. M. Tanner. “Codes with sparse graphs: transform analysis and construction.” In *Proceedings of the IEEE International Symposium on Information Theory*, p. 116. Cambridge, MA USA, 1998. [7]

M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielmann. “Improved low-density parity-check codes using irregular graphs and belief propagation.” *Proceedings of the IEEE International Symposium on Information Theory*, p. 117, Aug. 1998. [8]

G. M. Shepherd. *The Synaptic Organization of the Brain*. Oxford University Press, New York, 1974. [9]

G. M. Shepherd. *Neurobiology*. Oxford University Press, New York, 1983. [10]

C. A. Mead. *Analog VLSI and Neural Systems*. Addison Wesley Computation and Neural Systems Series. Addison Wesley, Reading, MA, 1989. ISBN 0-201-05992-4. [11]

[12] X. Arréguit, F. A. van Schaik, F. V. Baudin, M. Bidiville, and E. Raeber. "A CMOS motion detector system for pointing devices." *IEEE Journal of Solid-State Circuits*, vol. 31, no. 12, pp. 1916–1921, Dec. 1996.

[13] P. Masa, P. Heim, E. Franzi, X. Arréguit, F. Heitger, P. Ruedi, P. Nussbaum, P. Piiloud, and E. Vittoz. "10 mW CMOS retina and classifier for handheld, 1000 images/s optical character recognition system." In *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 204–205. San Francisco, CA, Feb 1999.

[14] A. Mortara, P. Heim, P. Masa, E. Franzi, P. F. Ruedi, F. Heitger, and J. Baxter. "An opto-electronic 18 b/revolution absolute angle and torque sensor for automotive steering applications." In *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 182–183. San Francisco, CA, Feb. 2000.

[15] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*, vol. 16. North-Holland, 1977. ISBN 0 444 85193 3.

[16] S. Lin and D. J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall Series in Computer Applications in Electrical Engineering. Prentice Hall, Englewood Cliffs, NJ, 1983.

[17] ITU-T. "Recommendation v.34 — a modem operating at data signalling rates of up to 33600 bit/s for use on the general switched telephone network and on leased point-to-point 2-wire telephone-type circuits." Tech. rep., International Telecommunication Union, Geneva, February 1998. Available at `http://www.itu.int/itudoc/itu-_t/rec/v/v34.html`.

[18] ITU-T. "Recommendation v.90 — a digital modem and analogue modem pair for use on the public switched telephone network (PSTN) at data signalling rates of up to 56000 bit/s downstream and up to 33600 bit/s upstream." Tech. rep., International Telecommunication Union, Geneva, September 1998. Available at `http://www.itu.int/itudoc/itu-_t/rec/v/v90.html`.

[19] W. Y. Chen. *DSL: Simulation Techniques and Standards Development for Digital Subscriber Line Systems*. Macmillan Technology Series. Macmillan Technical Publishing, Indianapolis, IN, 1998.

[20] T. Richardson, A. Shokrollahi, and R. Urbanke. "Design of provably good low-density parity check codes.", April 1999. Submitted to IEEE Transactions on Information Theory.

[21] C. Schlegel. *Trellis Coding*. IEEE Press, NewYork, 1997.

D. J. C. MacKay. "Good error-correcting codes based on very sparse matrices." *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, March 1999. [22]

A. J. Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Transactions on Information Theory*, vol. 13, pp. 260–269, April 1967. [23]

G. D. Forney, Jr. "The Viterbi algorithm." *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, March 1973. [24]

S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara. "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding." *IEEE Transactions on Information Theory*, vol. 44, no. 3, pp. 909–926, May 1998. [25]

J. G. Proakis. *Digital Communications*. McGraw-Hill, third edn., 1995. [26]

R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, 1984. [27]

A. S. Acampora and R. P. Gilmore. "Analog Viterbi decoding for high speed digital satellite channels." *IEEE Transactions on Communications*, vol. 26, no. 10, pp. 1463–1470, Oct. 1978. [28]

T. W. Matthews and R. R. Spencer. "An analog CMOS Viterbi detector for dgital magnetic recording." In *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 214–215. San Francisco, CA, 1993. [29]

M. H. Shakiba, D. A. Johns, and K. W. Martin. "Analog implementation of class-IV partial-response Viterbi detector." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 91–94. London, Mai 1994. [30]

M. H. Shakiba, D. A. Johns, and K. W. Martin. "A 200 MHz 3.3 V BiCMOS class-IV partial-response analog Viterbi decoder." In *Proceedings of the IEEE Custom Integrated Circuit Conference*, pp. 567–570. Santa Clara, May 1995. [31]

M. H. Shakiba, D. A. Johns, and K. W. Martin. "An integrated 200-MHz 3.3-V BiCMOS class-IV partial-response analog Viterbi decoder." *IEEE Journal of Solid-State Circuits*, vol. 33, no. 1, pp. 61–75, Jan. 1998. [32]

M. H. Shakiba, D. A. Johns, and K. W. Martin. "General approach to implementing analogue Viterbi decoders." *Electronics Letters*, vol. 30, no. 22, pp. 1823–1824, Oct. 1994. [33]

M. H. Shakiba, D. A. Johns, and K. W. Martin. "BiCMOS circuits for analog viterbi decoders." *IEEE Transactions on Circuits* [34] *and Systems–II: Analog and Digital Signal Processing*, vol. 45, no. 12, pp. 1527–1537, Dec. 1998.

[35] A. Demosthenous and J. Taylor. "Current-mode approaches to implementing hybrid analogue/digital Viterbi decoders." In *Proceedings of the International Conference on Electronics, Circuits and Systems*, vol. 1, pp. 33–36. Rhodos, 1996.

[36] A. Demosthenous, C. Verdier, and J. Taylor. "A new architecture for low power analogue convolutional decoders." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 37–40. Hong-Kong, 1997.

[37] A. Demosthenous and J. Taylor. "Low-power CMOS and BiCMOS circuits for analog convolutional decoders." *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 46, no. 8, pp. 1077–1080, Aug. 1999.

[38] K. He and G. Cauwenberghs. "An area-efficient analog VLSI architecture for state-parallel Viterbi decoding." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. II, pp. 432–435. Orlando, Florida, May 1999.

[39] Z. Wang and S. B. Wicker. "An artificial neural net Viterbi decoder." *IEEE Transactions on Communications*, vol. 44, pp. 165–171, Feb. 1996.

[40] C. Verdier, A. Demosthenous, J. Taylor, and M. Wilby. "An integrated analogue convolutional decoder based on the Hamming neural classifier." In *Proceedings of Neural Networks and Their Applications*, pp. 150–155. 1996.

[41] M. Helfenstein. *Analysis and Design of Switched-Current Networks*. Ph.D. thesis, ETH Zürich, Konstanz, 1997.

[42] H. P. Schmid. *Single-Amplifier Biquadratic MOSFET-C Filters*. Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, October 2000.

[43] G. S. Moschytz. *MOS Switched-Capacitor Filters: Analysis and Design*. IEEE Press, New York, 1984.

[44] G. C. Temes and R. Gregorian. *Analog MOS Integrated Circuits for Signal Processing*. John Wiley & Sons, New York, 1986.

[45] C. Toumazou, J. B. Hughes, and N. C. Battersby, editors. *Switched-Currents: an analogue technique for digital technology*. IEE/Peter Peregrinus Ltd., 1993. ISBN 0-86341-294-7.

[46] G. J. Minty. "A comment on the shortest-route problem." *Oper. Res.*, vol. 5, p. 724, 1957.

[47] L. Bu and T.-D. Chiueh. "Solving the shortest path problem using an analog network." *IEEE Transactions on Circuits*

and Systems–I: Fundamental Theory and Applications, vol. 46, no. 11, pp. 1360–1363, November 1999.

R. C. Davis. "Diode-configured Viterbi algorithm error correcting decoder for convolutional codes." U.S. Patent 4545054, Oct. 1985. [48]

R. C. Davis and H.-A. Loeliger. "A nonalgorithmic maximum likelihood decoder for trellis codes." *IEEE Transactions on Information Theory*, vol. 39, pp. 1450–1453, July 1993. [49]

H. F. Schantz. "An overview of neural OCR networks." *Journal of Information Systems Management*, vol. 8, no. 2, pp. 22–27, 1991. [50]

H. F. Schantz. "Neural network-OCR/ICR recognology. Theory and applications." *Document Image Automation*, vol. 13, no. 3, pp. 20–23, 1993. [51]

D. Jacquet and G. Saucier. "Design of a digital neural chip: application to optical character recognition by neural network." In *Proceedings of the European Design and Test Conference EDAC-ETC-EUROASIC*, pp. 256–260. 1994. [52]

J. Wang and J. Jean. "Segmentation of merged characters by neural networks and shortest path." *Pattern Recognition*, vol. 27, no. 5, pp. 649–658, May 1994. [53]

D. A. Kelly. "Neural networks for handwriting recognition." In *Proceedings of the SPIE*, vol. 1709, pp. 143–154. 1992. [54]

M. Schenkel, I. Guyon, and D. Henderson. "On-line cursive script recognition using time-delay neural networks and hidden markov models." *Machine Vision and Applications*, vol. 8, no. 4, pp. 215–223, 1995. [55]

R. Seiler, M. Schenkel, and F. Eggimann. "Off-line cursive handwriting recognition compared with on-line recognition." In *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 4, pp. 505–509. 1996. [56]

J. Rouat. "Spatio-temporal pattern recognition with neural networks: application to speech." In *Proceedings of Artificial Neural Networks - ICANN '97*, pp. 43–48. 1997. [57]

G. K. Venayagamoorthy, V. Moonasar, and K. Sandrasegaran. "Voice recognition using neural networks." In *Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98*, pp. 29–32. Rondebosch, South Africa, Sept. 1998. [58]

K. M. Olson and G. A. Ybarra. "Performance comparison of neural network and statistical pattern recognition approaches to [59] automatic target recognition of ground vehicles using SAR imagery." *Proceedings of the SPIE*, vol. 3161, pp. 159–170, 1997.

[60] S. B. Cho. "Pattern recognition with neural networks combined by genetic algorithm." *Fuzzy Sets and Systems*, vol. 103, no. 2, pp. 339–347, April 1999.

[61] N. Wiberg. "Approaches to neural-network decoding of error-correcting codes." Linköping Studies in Science and Technology, Thesis No. 425, 1994.

[62] Y.-J. Wu, P. M. Chau, and R. Hecht-Nielsen. "A supervised learning neural-network coprocessor for soft-decision maximum-likelihood decoding." *IEEE Transactions on Neural Networks*, vol. 6, pp. 986–992, July 1995.

[63] S. H. Bang and B. J. Sheu. "A neural network for detection of signals in communication." *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, vol. 43, pp. 644–655, Aug. 1996.

[64] L. A. Zadeh. "Fuzzy sets." *Information and Control*, vol. 8, pp. 328–353, 1965.

[65] M. Wu, W.-P. Zhu, and S. Nakamura. "A hybrid fuzzy neural decoder for convolutional codes." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 235–238. IEEE, Monterey, CA, June 1998.

[66] N. Wiberg, H.-A. Loeliger, and R. Koetter. "Codes and iterative decoding on general graphs." *European Transactions on Telecommunications*, vol. 6, pp. 513–525, Sept./Oct. 1995.

[67] N. Wiberg. *Codes and Decoding on General Graphs*. Ph.D. thesis, Univ. Linköping, Sweden, 1996.

[68] J. Hagenauer and M. Winkelhofer. "The analog decoder." In *Proceedings of the IEEE International Symposium on Information Theory*, p. 145. Cambridge, MA USA, Aug. 1998.

[69] J. Hagenauer, E. Offer, C. Méasson, and M. Moerz. "Decoding and equalization with analog non-linear networks." *European Transactions on Telecommunications*, vol. 10, no. 6, pp. 659–680, Nov./Dec. 1999.

[70] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. "An analog $0.25\,\mu$m BiCMOS tailbiting MAP decoder." In *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 356–357. San Francisco, CA, Feb. 2000.

[71] H.-A. Loeliger, M. Helfenstein, F. Lustenberger, and F. Tarköy. "Probability propagation and decoding in analog VLSI." In *Pro-*

ceedings of the IEEE International Symposium on Information Theory, p. 146. Cambridge, MA, Aug. 1998.

M. Helfenstein, H.-A. Loeliger, F. Lustenberger, and F. Tarköy. "Verfahren und Schaltung zur Signalverarbeitung, insbesondere zur Berechnung einer Wahrscheinlichkeitsfunktion." Swiss Patent Application no. 1998 0375/98, Feb. 1998. Filed Feb. 17, 1998. [72]

H.-A. Loeliger, F. Lustenberger, F. Tarköy, and M. Helfenstein. "Decoding in analog VLSI." *IEEE Communications Magazine*, vol. 37, no. 4, pp. 99–101, April 1999. [73]

F. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarköy, and G. S. Moschytz. "All-analog decoder for a binary (18,9,5) tail-biting trellis code." In *Proceedings of the European Solid-State Circuits Conference*, pp. 362–365. Duisburg, Sep. 1999. [74]

F. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarköy, and G. S. Moschytz. "An analog decoding technique for digital codes." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. II, pp. 428–431. Orlando, FL, June 1999. [75]

M. Helfenstein, F. Lustenberger, H.-A. Loeliger, F. Tarköy, and G. S. Moschytz. "High-speed interfaces for analog, iterative decoders." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. II, pp. 424–427. Orlando, FL, June 1999. [76]

H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy. "Probability propagation and decoding in analog VLSI.", Sept. 2000. Accepted for publication in IEEE Transactions on Information Theory, available at `http://www.isi.ee.ethz.ch/~lustenbe/papers/IT_2000.pdf`. [77]

H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy. "Analog probability propagation networks — Part I: Fundamentals." In preparation. [78]

F. Lustenberger, H.-A. Loeliger, M. Helfenstein, and F. Tarköy. "Analog probability propagation networks — Part II: Decoder examples." In preparation. [79]

C. A. Mead. "Neuromorphic electronic systems." *Proceedings of the IEEE*, vol. 78, pp. 1629–1636, Oct. 1990. [80]

J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead. "Winner-take-all networks of $O(n)$ complexity." In *Advances in Neural Information Processing Systems 1* (edited by D. Tourestzky), pp. 703–711. Morgan Kaufmann Publishers, San Mateo, CA, 1988. [81]

[82] J. P. Lazzaro. "Low-power silicon spiking neurons and axons." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 2220–2223. San Diego, CA, 1992.

[83] J. P. Lazzaro, J. Wawrzynek, and R. P. Lippmann. "A micropower analog circuit implementation of hidden markov model state decoding." *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1200–1209, Aug. 1997.

[84] R. Sarpeshkar, L. Watts, and C. A. Mead. "Refractory neuron circuits." Computation and Neural Systems Memo CNS TR-92-08, California Institute of Technology, Pasadena, CA, 1992.

[85] Y. Arima, M. Murasaki, T. Yamada, A. Maeda, and H. Shinohara. "A refreshable analog VLSI neural network chip with 400 neurons and 40K synapses." *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1854–1861, Dec. 1992.

[86] A. F. Murray, L. Tarassenko, H. M. Reekie, A. Hamilton, M. Brownlow, S. Churcher, and D. J. Baxter. "Pulsed silicon neural networks: Following the biological leader." In *VLSI Desing of Neural Networks* (edited by U. Ramacher and U. Rückert), pp. 103–123. Kluwer Academic Publishers, 1991.

[87] R. F. Lyon and C. A. Mead. "An analog electronic cochlea." *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36, no. 7, pp. 1119–1134, July 1988.

[88] J. Lazzaro and C. A. Mead. "Circuit models of sensory transduction in the cochlea." In *Analog VLSI Iimplementation of Neural Systems* (edited by C. A. Mead and M. Ismail), pp. 85–101. Kluwer Academic Publishers, 1989.

[89] L. Watts, D. A. Kerns, R. F. Lyon, and C. A. Mead. "Improved implementation of the silicon cochlea." *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 692–700, May 1992.

[90] F. Lustenberger. *Cochlée artificielle en silicium*. Semester project, École Polytechnique Federal de Lausanne, Lausanne, 1994.

[91] A. van Schaik, E. Fragnière, and E. Vittoz. "Improved silicon cochlea using compatible lateral bipolar transistors." In *Advances in Neural Information Processing Systems* (edited by D. Touretzky), pp. 671–677. MIT Press, Cambridge MA, 1996.

[92] C. A. Mead. "Adaptive retina." In *Analog VLSI implementation of neural systems* (edited by C. A. Mead and M. Ismail), pp. 239–246. Kluwer Academic Publishers, 1989.

[93] M. A. Mahowald. "Silicon retina with adaptive photodetectors." In *Proceedings SPIE, Visual Information Processing: From Neurons to Chips*, vol. 1473, pp. 52–58. 1991.

M. A. Mahowald. "Analog VLSI chip for stereocorrespondence." In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 347–350. London, 1994. [94]

W. Bair and C. Koch. "Real-time motion detection using an analog VLSI zero-crossing chip." In *Proceedings SPIE, Visual Information Processing: From Neurons to Chips*, vol. 1473, pp. 59–65. 1991. [95]

A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, third edn., 1991. [96]

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm.", June 2000. Submitted and revised for publication in IEEE Trans. Inform. Theory. Available at `http://www.comm.utoronto.ca/frank/factor`. [97]

R. M. Tanner. "A recursive approach to low complexity codes." *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981. [98]

G. D. Forney, Jr. "Codes on graphs: Generalized state realizations.", November 1998. Draft. [99]

G. D. Forney, Jr. "Codes on graphs: Normal realizations." In *Proceedings of the IEEE International Symposium on Information Theory*, p. 9. June 2000. [100]

R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, New York, NY, 1989. [101]

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm.", July 1998. Private communication. [102]

L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. "Optimal decoding of linear codes for minimizing symbol error rate." *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, March 1974. [103]

National Semiconductors. "An application guide for Op Amps. Application Note 20." In *Linear Applications Handbook*, pp. 19–30. 1994. [104]

B. Gilbert. "Translinear circuits: A proposed classification." *Electronics Letters*, vol. 11, no. 1, pp. 14–16, January 1975. [105]

E. Seevinck. *Analysis and Synthesis of Translinear Integrated Circuits*, vol. 31 of *Studies in electrical and electronic engineering*. Elsevier, Amsterdam, first edn., 1988. ISBN 0-444-42888-7. [106]

B. Gilbert. "Translinear circuits: An historical overview." *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 2, pp. 95–118, March 1996. Special Issue: Translinear Circuits. [107]

[108] A. B. Grebene. *Bipolar and MOS Analog Integrated Circuit Design*. John Wiley & Sons, 1984. ISBN 0-471-08529-4.

[109] Y. Tsividis. *Operation and Modelling of The MOS Transistor*. McGraw-Hill, second edn., 1999. ISBN 0-07-116791-9.

[110] T. Serrano-Gotarredano, B. Linares-Barranco, and A. G. Andreou. "A general translinear principle for subthreshold MOS transisitors." *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, vol. 46, no. 5, pp. 607–616, May 1999.

[111] K. Bult. *Analog CMOS square-law circuits*. Ph.D. thesis, Twente University of Technology, 1988.

[112] E. Seevinck and R. J. Wiegerink. "Generalized translinear circuit principle." *IEEE Journal of Solid-State Circuits*, vol. 26, no. 8, pp. 1098–1102, Aug. 1991.

[113] R. J. Wiegerink. *Analysis and synthesis of MOS translinear circuits*. Ph.D. thesis, Twente University of Technology, 1992.

[114] R. W. Adams. "Filtering in the log-domain." Preprint 1470, presented at 63rd Audio Engineering Society Conferenc, May 1979.

[115] E. Seevinck. "Companding current-mode integrator: a new circuit principle for continous-time monolithic filters." *Electronics Letters*, vol. 26, pp. 2046–2047, Nov. 1990.

[116] D. Frey. "Log domain filtering: an approach to current-mode filtering." *IEE Proceedings, Part G*, vol. 140, pp. 406–416, Dec. 1993.

[117] D. Perry and G. W. Roberts. "The design of log-domain filters based on the operational simulation of LC ladders." *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 43, no. 11, pp. 763–774, Nov. 1996.

[118] Y. Tsividis. "Externally linear, time-invariant systems and their application to companding signal processors." *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 44, no. 2, pp. 65–85, Feb. 1997.

[119] C. Toumazou, J. Ngarmnil, and T. S. Lande. "Micropower log domain filter for electronic cochlea." *Electronics Letters*, vol. 30, pp. 1839–1841, Oct. 1994.

[120] C. Enz and Y. Cheng. "MOS transistor modeling issues for RF circuit design.", 1999. Workshop on Advances in Analog Circuit Design (AACD'99).

B. Gilbert. “A precise four-quadrant multiplier with subnanosecond response.” *IEEE Journal of Solid-State Circuits*, vol. 3, pp. 365–373, 1968. [121]

K. Kimura. “Some circuit design techniques using two cross-coupled pairs.” *IEEE Transactions on Circuits and Systems–I: Fundamental Theory and Applications*, vol. 41, no. 5, pp. 411–423, May 1994. [122]

C. F. Chan, H. Ling, and O. Choy. “A one volt four-quadrant analog current mode multiplier cell.” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 9, pp. 1018–1019, Sept. 1995. [123]

G. Colli and F. Montecchi. “Low voltage low power CMOS four-quadrant analog multiplier for neural network applications.” In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 496–499. 1996. [124]

W. Gai, H. Chen, and E. Seevinck. “Quadratic-translinear CMOS multiplier-divider circuit.” *Electronics Letters*, vol. 33, no. 10, pp. 860–861, May 1997. [125]

R. J. Wiegerink. “A CMOS four-quadrant analog current multiplier.” In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 2244–2247. 1991. [126]

K. Kimura. “A bipolar low-voltage quarter-square multiplier with a resistive-input based on the bias offset technique.” *IEEE Journal of Solid-State Circuits*, vol. 32, no. 2, pp. 258–266, Feb. 1997. [127]

H. R. Mehrvarz and C. Y. Kwok. “A novel multi-input floating-gate MOS four-quadrant analog multiplier.” *IEEE Journal of Solid-State Circuits*, vol. 31, no. 8, pp. 1123–1131, Aug. 1996. [128]

J. Ramirez-Angulo. “± 0.75 V BiCMOS four-quadrant analog multiplier with rail-rail input signal-swing.” In *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 242–245. 1996. [129]

K. R. Laker and W. M. C. Sansen. *Design of analog integrated circuits and systems*. McGraw-Hill, third edn., 1994. ISBN 0-07-113458-1. [130]

B. Gilbert. “A monolitic 16-channel analog array normalizer.” *IEEE Journal of Solid-State Circuits*, vol. 19, pp. 956–963, 1984. [131]

J. Vogt, K. Koora, A. Finger, and G. Fettweis. “Comparison of different turbo decoder realizations for IMT-2000.” In *Proceedings of the Global Telecommunications Conference*, vol. 5, pp. 2704–2708. Rio de Janeireo, Brazil, Dec. 1999. [132]

[133] F. Poegel. "Private email communication: The resolution of signals in different decoder architectures.", July 2000. This topic will appear in Frank's PhD thesis which he is currently finishing at TU Dresden, Germany.

[134] COMATLAS. *Datasheet of the Turbo-code codec CAS 5093.*

[135] E. Säckinger and W. Guggenbühl. "A high-swing, high-impedance MOS cascode circuit." *IEEE Journal of Solid-State Circuits*, vol. 25, no. 1, pp. 289–298, Feb. 1990.

[136] P. J. Crawley and G. W. Roberts. "High-swing MOS current mirror with arbitrarily high output resistance." *Electronics Letters*, vol. 28, no. 4, pp. 361–363, Feb. 1992.

[137] D. A. Johns and K. Martin. *Analog Integrated Circuit Design.* John Wiley & Sons, 1997.

[138] P. R. Gray and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits.* Wiley, New York, third edn., 1993.

[139] Harris Semiconductors. *Datasheet of the CA3096 NPN/PNP Transistor Array*, December 1997.

[140] Austria Mikrosystem International GmbH. *Process Parameters and Design Rules of the* 0.8 μm *silicon BiCMOS process*, 1999. See also http://www.amsint.com.

[141] A. M. Aji, G. B. Horn, and R. J. McEliece. "Iterative decoding on graphs with a single cycle." In *Proceedings of the IEEE International Symposium on Information Theory*, p. 276. Cambridge, MA, Aug. 1998.

[142] M. J. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. "Matching properties of MOS transistors." *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, Oct. 1989.

[143] T. Richardson and R. Urbanke. "The capacity of low-density parity check codes under message-passing decoding.", September 2000. Accepted for publication in IEEE Transactions on Information Theory.

[144] P. Robertson, E. Villebrun, and P. Hoeher. "A comparison of optimal and sub-optimal decoding algorithms in the log domain." In *Proceedings of the International Conference on Communications*, vol. 2, pp. 1009–1013. Seattle, WA, June 1995.

[145] R. M. Tanner. "On quasi-cyclic repeat-accumulate codes." In *Proc. 37th Allerton Conf. on Communications, Control, and Computing*. Monticello, Illinois, Sept. 1999.

[146] R. M. Tanner. "Transforming quasi-cyclic codes with sparse graphs.", Jan. 2000. Submitted to IEEE Trans. Inform. The-

ory. Available at `http://www.cse.ucsc.edu/~tanner/pubs.html`.

P. O. Vontobel. "Investigation of quasi-cyclic repeat-accumulate codes suitable for a chip implementation." Internal report, Signal and Information Processing Laboratory, ETH Zurich, 2000. [147]

H. Traff. "Novel approach to high-speed CMOS current comparators." *Electronics Letters*, vol. 28, no. 3, pp. 310–312, Jan. 1992. [148]

G. D. Forney, Jr. "The forward-backward algorithm." In *Proc. 34th Allerton Conf. on Communications, Control, and Computing*, pp. 432–446. Allerton House, Monticello, Illinois, Oct. 1996. [149]

S. M. Moser. *Investigation of Algebraic Codes of Small Block Length using Factor Graphs*. Master's thesis, Signal- and Information Processing Laboratory, ETH Zurich, Zurich, March 1999. [150]

G. Fromherz and E. Schinca. *Konvergenzverhalten des Summe-Produkt-Algorithmus in Standard-CMOS-Technologie*. Master's thesis, Signal- and Information Processing Laboratory, ETH Zurich, Zurich, March 1999. [151]

E. A. Vittoz. "MOS and Bipolar transistors." Electronics Laboratories Advanced Engineering Course on CMOS and BiCMOS VLSI Design '94, Aug. 1994. [152]

A. Rodriguez-Vasquez, R. Navas, M. Delgado-Restituto, and F. Vidal-Verdu. "A modular programmable CMOS analog fuzzy controller chip." *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 46, pp. 251–265, March 1999. [153]

M. Helfenstein, H.-A. Loeliger, F. Lustenberger, and F. Tarköy. "Verfahren zur mathematischen Verarbeitung zweier Werte in einer elektrischen Schaltung." Swiss Patent Application no. 1999 1448/99, Feb. 1999. Filed Aug. 6, 1999. [154]

Y. Tsividis. *Mixed Analog-Digital VLSI Devices and Technology: An Introduction*. McGraw-Hill, 1995. ISBN 0-07-065402-6. [155]

S. M. Sze. *Physics of Semiconductor Devices*. John Wiley & Sons, New York, second edn., 1982. ISBN 0-471-09837-X. [156]

S. M. Sze. *Semiconductor Devices, Physics and Technology*. John Wiley & Sons, New York, 1985. ISBN 0-471-83704-0. [157]

S. Wang. *Fundamentals of Semiconductor Theory and Device Physics*. Prentice Hall Series in Electrical and Computer Engineering. Prentice-Hall, Englewood Cliffs, NJ, 1989. ISBN 0-13-344425-2. [158]

Curriculum Vitæ

I was born in Lucerne, Switzerland, on May 31, 1969. After finishing high-school at the *Kantonssschule Alpenquai, Lucerne*, in 1989 (Matura Typus C) and a one-year interruption for military services, I enrolled in Micro Engineering at the Swiss Federal Institute of Technology EPF Lausanne. I received the Diploma (M.Sc.) degree in Micro Engineering (*Ing. en Microtechnique dipl. EPFL*) in 1995 for the design, implementation, and testing of an artificial silicon cochlea in CMOS technology. In April 1995 I joined the Signal and Information Processing Laboratory (ISI) of ETH Zurich, where I worked as a teaching assistant for two years. During this time, I attended the post-diploma program in Information Technology which I completed with a Dipl. NDS degree in Information Technology in 2000. From autumn 1997 to summer 2000 I participated as research assistant at the interdisciplinary research project 'Design of Analog VLSI Iterative Decoders' (DAVID). Beside the work at the DAVID project presented in this dissertation, my main interests include general analog and bio-inspired circuit design, micro-systems design, system-oriented VLSI design and analog design automation.