Trellis Decoding

Trellis Decoding

From Algorithm to Flexible Architectures

Matthias Kamuf

Lund 2007

Department of Electroscience
Lund University
P.O. Box 118
SE-221 00 Lund, Sweden

This thesis is set in Computer Modern 10pt
with the $\LaTeX$ Documentation System

Series of licentiate and doctoral theses
No. 67
ISSN 1402-8662

Printed in Sweden by *Tryckeriet i E-huset*, Lund
February 2007

Abstract

Trellis decoding is a popular method to recover encoded information corrupted during transmission over a noisy channel. Prominent members of this class of decoding algorithms are the Viterbi algorithm, which provides maximum likelihood estimates, and the BCJR algorithm, which is a maximum *a posteriori* estimator commonly used in iterative decoding. In this thesis, the Viterbi algorithm is chosen since it provides a good trade-off between achievable coding gain and implementation complexity. This is the basis for considerations on simplified, hybrid, and, most importantly, flexible VLSI architectures.

Algorithm simplifications are necessary to reduce the computational burden laid on an implementation platform. In our work on trellis decoding blocks, a simplification that lowers the number of arithmetic operations is derived and evaluated. By using a complementary code property, the arithmetic complexity of the main part on the Viterbi algorithm is reduced by 17%. Synthesized blocks show varying savings for cell area and estimated power consumption. A comparison to a competing simplification shows the advantage in a hardware implementation of our work for the important class of rate 1/2 convolutional codes.

Hybrid architectures try to combine benefits of several approaches to lower the drawbacks of the individual contributors. For survivor path processing in Viterbi decoders, a new hybrid approach is proposed. A low-latency algorithm, whose implementation complexity quickly increases with the number of trellis states, is combined with a scalable RAM-based method. As a result, the developed hybrid architecture exhibits a better latency–complexity behavior compared to other hybrid approaches.

Flexible VLSI architectures to cover several communication standards become increasingly important as fabrication costs for microchips rise rapidly with every new process generation. In the context of flexible trellis decoding, earlier work mostly concentrated on varying encoder memory and thus the number of trellis states. This work studies the effects on hardware size and throughput introduced by flexibility if the code rate is varied. The investigation of a decoder for bandwidth-efficient codes, which was fabricated in a 0.13 μm digital

CMOS process and verified for functionality, distinguishes between task- and rate-flexibility. A comparison is carried out between flexible designs, which decode both convolutional and TCM codes and provide two or three transmission rates. It is concluded that the larger number of rates is more beneficial from a cost–flexibility viewpoint.

Two roads diverged in a wood, and I—

I took the one less traveled by

Robert Frost

Contents

Preface

This thesis summarizes my academic work in the Digital ASIC group at the Department of Electroscience for the Ph.D. degree in Circuit Design. The main contribution to the thesis is derived from the following publications:

- M. Kamuf, V. Öwall, and J. B. Anderson, "Optimization and implementation of a Viterbi decoder under flexibility constraints," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, submitted.
- M. Kamuf, V. Öwall, and J. B. Anderson, "Survivor path processing in Viterbi decoders using register-exchange and trace-forward," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, publication scheduled for June 2007.
- M. Kamuf, V. Öwall, and J. B. Anderson, "A hardware efficiency analysis for simplified trellis decoding blocks," in *Proceedings of IEEE Workshop on Signal Processing Systems*, Athens, Greece, Nov. 2005, pp. 128–132.
- M. Kamuf, V. Öwall, and J. B. Anderson, "Architectural considerations for rate-flexible trellis processing blocks," in *Proceedings of IEEE International Symposium on Personal, Indoor, and Mobile Radio Communication*, Berlin, Germany, Sept. 2005, pp. 1076–1080.
- M. Kamuf, J. B. Anderson, and V. Öwall, "Area and power efficient trellis computational blocks in 0.13 μm CMOS," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005, pp. 344–347.
- M. Kamuf, J. B. Anderson, and V. Öwall, "A simplified computational kernel for trellis-based decoding," *IEEE Communications Letters*, vol. 8, no. 3, pp. 156–158, Mar. 2004.

The following papers are published but not considered part of this thesis:

- D. Dasalukunte, A. Pálsson, M. Kamuf, P. Persson, R. Veljanovski, and V. Öwall, "Architectural optimization for low power in a reconfigurable UMTS filter," in *Proceedings of International Symposium on Wireless Personal Multimedia Communications*, San Diego, CA, Sept. 2006, pp. 264–267.

- J. N. Rodrigues, M. Kamuf, H. Hedberg, and V. Öwall, "A manual on ASIC front to back end design flow," in *Proceedings of International Conference on Microelectronic Systems Education*, Anaheim, CA, June 2005, pp. 75–76.

- H. Hedberg, J. N. Rodrigues, F. Kristensen, H. Svensson, M. Kamuf, and V. Öwall, "Teaching digital ASIC design to students with heterogeneous previous knowledge," in *Proceedings of International Conference on Microelectronic Systems Education*, Anaheim, CA, June 2005, pp. 15–16.

- M. Kamuf, J. B. Anderson, and V. Öwall, "Providing flexibility in a convolutional encoder," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Bangkok, Thailand, May 2003, pp. 272–275.

Acknowledgment

To summarize my gratitude for the people I met and who influenced me during my years as a Ph.D. student in Sweden is a difficult task.

First and foremost, I thank my dear principal supervisor Viktor Öwall for his enduring support, not only on a professional basis, but also for listening when needed. Also, the trips to various places in the world were great experiences for me and would not have been the same without him.

To my co-supervisor John B. Anderson, who has a great way to get people interested in communication (theory, too). His professional contributions could not be fully appreciated without all the anecdotes on the stories behind the stories—I think you should publish them.

Thanks to Joachim for bringing me to Sweden and for being a good friend. The things you did for me, I will not forget.

My colleagues in the DASIC corridor, both former and current, for making the days at work extra worthwhile. Fredrik for his constructive comments while reading parts of this thesis, Henrik for nice talks about everything and nothing, which of course also applies to Hugo, Thomas, Hongtu, Deepak, and Johan, who have accompanied me during these years. Also thanks to Martin for helping me to verify my chip just two days ago.

Not to forget, all the people that keep the department up and running in one way or another. Erik for his computer support, Stefan as CAD-responsible, Pia, Elsbieta, and Lars for fixing all the little things that could pop up during any single day.

Last but surely not least, to my family, my mother for her unconditional support throughout these years, and my brother for just being the good guy he is. Thanks for being there.

Lund, February 27, 2007

Matthias Kamuf

List of Acronyms

ACS	Add-Compare-Select
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	Bit Error Rate
BM	Branch Metric
CDMA	Code Division Multiple Access
CMOS	Complementary Metal Oxide Semiconductor
DMC	Discrete Memoryless Channel
DSP	Digital Signal Processor
FA	Full Adder
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPP	General-Purpose Processor
HDL	Hardware Description Language
LIFO	Last In, First Out
MAP	Maximum *A Posteriori*
ML	Maximum Likelihood

MSB	Most Significant Bit
PAM	Pulse Amplitude Modulation
PE	Processing Element
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RAM	Random-Access Memory
RE	Register-Exchange
RTL	Register-Transfer Level
SDR	Software-Defined Radio
SM	State Metric
SNR	Signal-to-Noise Ratio
SOVA	Soft-Output Viterbi Algorithm
SP	Survivor Path
TB	Trace-Back
TCM	Trellis-Coded Modulation
TF	Trace-Forward
UMC	United Microelectronics Company
VA	Viterbi Algorithm
VHDL	Very High Speed Integrated Circuit HDL
VLSI	Very Large Scale Integration
WPAN	Wireless Personal Area Network

Chapter 1

Introductory Remarks

This thesis discusses issues that arise at the transition from algorithm to architecture and hardware implementation. The algorithms concern channel coding and decoding used in wireless communication systems.

According to Shannon [84], "*the fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point*". In order to achieve this goal, channel coding tries to protect information from disturbances in a channel used as transportation medium. Due to these disturbances, transmitted and received signal are not the same. In order to restore the original information, the decoder exploits the received signal's redundancy, which was added by the channel code. There are several coding schemes and decoding algorithms that re-establish the original information with almost any desired accuracy. The achievable accuracy depends on the complexity of the coding scheme; the more complex, the more accurate the estimates. Clearly, the ultimate goal is to exactly reproduce the sent information, which is theoretically possible for communication systems that operate below the capacity bound defined by Shannon in [85]. More on that in Chapter 2.

To begin with, we specify three design domains: algorithm, architecture, and implementation. The links between these domains are design constraints, for example, throughput, energy efficiency, flexibility, latency, and so on. These constraints determine the choice of architecture and ultimately also the implementation platform. At last, the importance of mapping algorithms to efficient architectures is outlined. The main contributions of this thesis are simplified, hybrid, and flexible VLSI architectures for trellis decoding. These contributions are briefly presented in Section 1.2.

1.1 Algorithm–Architecture Trade-offs

An algorithm is an ordered list of (mathematical) instructions. It has neither connection to processing time, nor is it concerned with feasibility or required processing capacity. Take for instance the very simple algorithm $S \leftarrow A + B$. The algorithm consists of a single instruction. Storage of intermediate results is not necessary. Note that the instruction does not tell the way it is supposed to be carried out.

An architecture specifies how an algorithm is calculated. When designing an architecture, one must have in mind future implementation constraints set by timing, area, and power consumption. For example, the algorithm $S \leftarrow A + B$ can be carried out in many ways. The simplest architecture is a ripple-carry adder, which consists of a chain of full adders (FAs); see Figure 1.1.

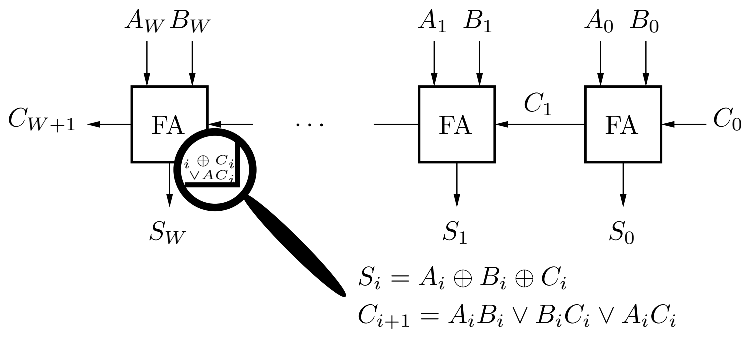


Figure 1.1: Ripple-carry architecture to carry out $S \leftarrow A + B$. The operands A, B are $W + 1$ bits wide. Also shown are the equations for a 1-bit addition (FA), which in turn can have several physical realizations.

Other adder architectures are carry-lookahead (CL), Brent–Kung, and so on [76]. Why to choose one over another depends on the constraints of the implementation, which is a physical realization of an architecture. The performance of different implementations can be characterized in the area–delay plane. Figure 1.2 shows different implementations of 16-bit adders synthesized towards a 0.13 μm digital CMOS standard cell library. The reciprocal of delay is a measure of throughput. Desirable is the lowest area–delay product, indicated by the thick shaded line in the figure. It spans different architectures, which are optimal in an area–delay sense in their respective delay regions.

The transition from architecture to implementation offers several platform choices depending on the overall performance goals. The goals are throughput, energy efficiency, flexibility, and so on. In the previous example, application-specific integrated circuits (ASICs) were the target platform. Other platforms

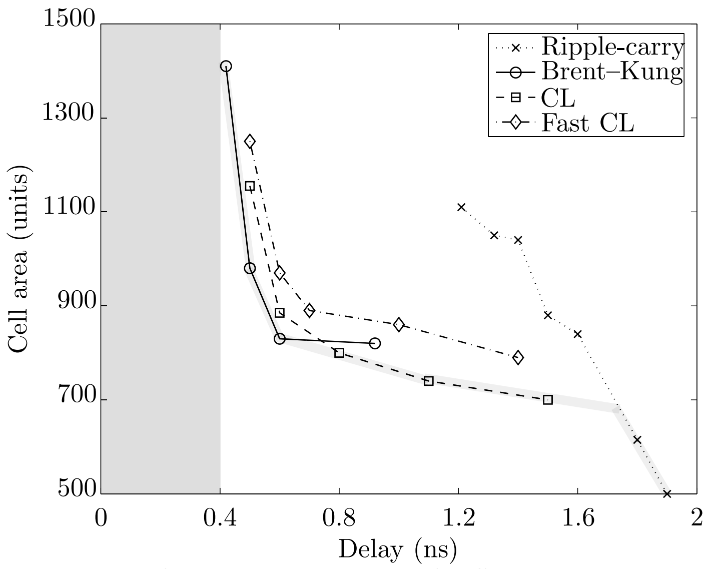


Figure 1.2: Performance comparison of different standard cell implementations of a 16-bit adder. The shaded area is the region where there is no physical realization of the algorithm for the chosen platform.

are general-purpose processors (GPPs), digital signal processors (DSPs), or field-programmable gate arrays (FPGAs) to name a few. Some of these platforms are discussed in Chapter 3.

In a throughput–flexibility plane, one can establish a graph shown in Figure 1.3. The different domains express the feasibility region of a certain platform. As one traverses through this plane, different requirements determine the "right" implementation platform. There is a trade-off between the order of throughput that can be provided with a certain grade of flexibility or reconfigurability. As flexibility is decreased, the performance measured in throughput or energy efficiency increases, and vice versa. The superiority of dedicated designs on the one hand is paid for by limited flexibility on the other hand. In Chapter 3, there are measures for these properties, exemplified by Viterbi decoders that run on dedicated platforms compared to general-purpose ones.

1.1.1 On Complexity Reduction

Algorithms are often developed and tested in floating-point environments on GPPs in order to show the achievable optimal performance. Besides shortest development time, there are no requirements on, for example, processing speed

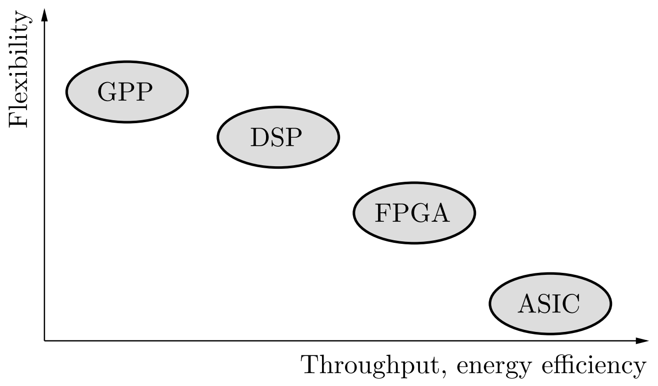


Figure 1.3: Different implementation platforms and their application domains.

or power consumption, and hence this platform is the best choice for the job. However, speed or power constraints might require an implementation in more or less specialized hardware. This transition usually causes many degradations, for example, reduced dynamic range caused by fixed-point arithmetic, which on the other hand provides tremendous reduction in implementation complexity.

Whether fixed- or floating-point arithmetic is used, the sheer amount of computation is often the biggest challenge when mapping an algorithm onto a feasible architecture and implementation. Therefore, simplifications are necessary that may or may not cause performance degradation. Although not covered in this thesis, consider as an example maximum likelihood (ML) decoding of space–time codes [1, 91] used in multiple-input-multiple-output communication systems [47]. To date, this decoding algorithm is much too complex in number of operations for a straightforward implementation to run at reasonable speed. Therefore, sphere decoding methods [42, 82] were applied, which limit the search space of the ML algorithm and thus decrease the number of operations to be carried out. Of course, the suboptimality of these algorithms degrade performance compared to optimal ML decoding.

To conclude, simplifications that cause little (ideally no) degradation are a necessity to satisfy ever increasing requirements on throughput and energy efficiency in VLSI implementations.

1.1.2 Hybrid Architectures

Nowadays, the word “hybrid” is often connected with cars that can run on several fuels, for example, petrol and natural gas or ethanol. With such cars, the fatal environmental impact of burning petrol is relaxed if drivers use the more environmental-friendly ("green") fuel in cases where driving distance is not the prime matter, for example, in urban traffic. That is, the benefits of two worlds, as well as drawbacks of either approach, are traded for each other to arrive at an overall favorable solution.

The hybrid principle applied to the topics in this thesis, different algorithms or architectures, with their benefits and drawbacks, can be combined to enhance the new approaches' overall feasibility. Recent examples of hybrid approaches in the coding field are [27, 28]. In [28], forward trellis processing from the suboptimal max-log-MAP algorithm is combined with the backward pass from optimal log-MAP decoding. This hybrid scheme can also be counted as simplification discussed in the previous subsection. Decoding performance is slightly degraded at the benefit of reduced storage requirement and power consumption. The authors of [27] put together two coding methods. One of the methods performs well in memoryless channels, the other is designed to cope with burst errors that often appear in channels with memory. The resulting hybrid coding scheme can therefore be applied to a greater variety of scenarios.

1.1.3 Why Flexibility?

Although the word flexibility is used extensively in almost every context, it is not easily defined as a free-standing concept. Linguistically [89], it is defined according to

Definition 1 *Susceptibility of modification or alteration; capacity for ready adaptation to various purposes or conditions; freedom from stiffness or rigidity.*

In other words, applied to the issues stated in the preceding paragraphs, flexibility expresses the grade of a design being able to adapt its parameters to various conditions and purposes. A flexible design not only serves one specific algorithm optimized in one special way, but provides an architecture with varying capabilities for, for example, throughput or power consumption. In particular, power consumption has become more and more important since many applications are or will be mobile and thus processing has to be energy-efficient. On top of all this, these objectives should be met together with high throughputs provided by ASICs.

For example, a common effort to merge different communication standards onto a single platform is culminating in what today is named software-defined radio (SDR). According to this consortium [90],

"*SDR is a collection of hardware and software technologies that enable reconfigurable system architectures for wireless networks and user terminals. SDR provides an efficient and comparatively inexpensive solution to the problem of building multi-mode, multi-band, multi-functional wireless devices that* *can be enhanced using software upgrades. As such, SDR can really be considered an enabling technology that is applicable across a wide range of areas within the wireless industry.*"

This excerpt certainly emphasizes today's need for adaptable platform design, considering the cost of hardware design is far larger than the cost of changing software. This cost factor becomes even more distinct as process feature size shrinks; see Figure 1.4. This graph shows the progression of development and fabrication cost of an ASIC. Note that the cost for possible re-spins are not included in the numbers.

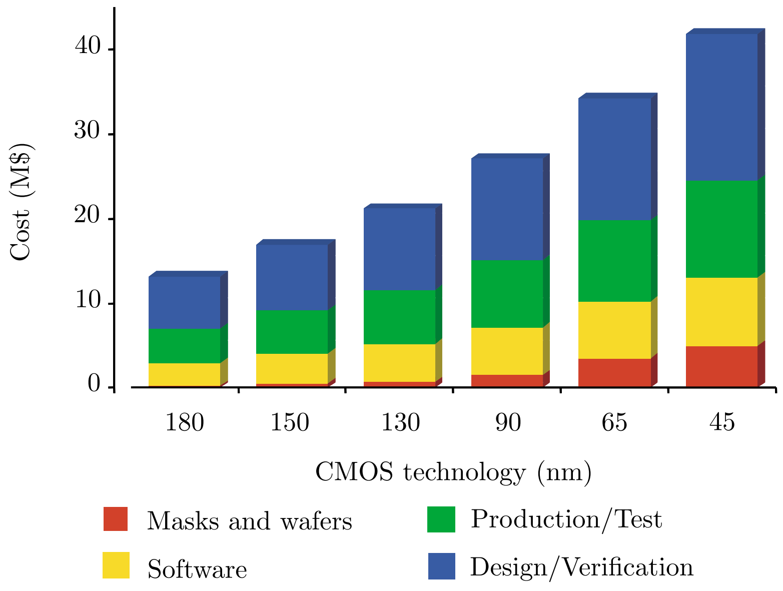


Figure 1.4: Standard cell ASIC development (large volumes) cost trend. Figure adapted from [13].

However, from Figure 1.3 it is seen that dedicated hardware has advantages in processing speed and is generally more efficient in energy per processed bit. That is, the gordic knot between required flexibility on the one hand and efficient implementation on the other hand must be solved. The solution is flexible architectures such as the one investigated and designed in this thesis. Note that this thesis concentrates on the decoding part of a communication system. Recent system-oriented approaches can be found in [97].

1.2 Thesis Contribution and Papers Published

In this thesis, algorithm–architecture issues discussed in the previous sections are addressed in the context of channel decoding. To begin with, necessary basics of channel coding and decoding are presented in Chapter 2. This theoretical chapter is followed by an overview of architectural concepts that enable an efficient implementation of algorithms. As an example, practical aspects for channel decoders (here, those based on the Viterbi algorithm) are discussed. Included is also a review of hardware implementation platforms with focus on ASICs. Having introduced these fundamental topics, the contributions of this thesis are found in Parts I–III. The content of these three parts is summarized now.

Part I: Simplified Trellis Computational Blocks

An algorithmic simplification to an important processing block in trellis decoders is derived. The block consists of add-compare-select (ACS) units that discard suboptimal branches in a code trellis. Based on these decisions the most likely state sequence can be reconstructed. Using the complementariness of code symbols along merging trellis branches, it is shown that for rate 1/2 codes, one addition can be saved in every other ACS unit. The effects in a hardware implementation are explored by synthesizing decoding blocks for different code constraint lengths. Thus, the predicted savings in cell area and power consumption compared to a conventional implementation can be verified. It is also discussed why a competing simplification, which also relies on the mentioned complementariness, cannot achieve the same reduction in a hardware implementation.

This part is an extended version of:

- M. Kamuf, V. Öwall, and J. B. Anderson, “A hardware efficiency analysis for simplified trellis decoding blocks,” in *Proceedings of IEEE Workshop on Signal Processing Systems*, Athens, Greece, Nov. 2005, pp. 128–132.

- M. Kamuf, V. Öwall, and J. B. Anderson, “Area and power efficient trellis computational blocks in 0.13 μm CMOS,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005, pp. 344–347.

- M. Kamuf, J. B. Anderson, and V. Öwall, “A simplified computational kernel for trellis-based decoding,” *IEEE Communications Letters*, vol. 8, no. 3, pp. 156–158, Mar. 2004.

Part II: Hybrid Survivor Path Architectures

Survivor path processing requires a large amount of computation in Viterbi decoders. Low latency and implementation efficiency are desirable properties and there are two basic algorithms, register-exchange (RE) and trace-back (TB), that fulfill either property. Usually, the number of trellis states is a good indicator for when to use which. In order to trade one property for another, hybrid approaches have partly lowered the drawbacks of the two algorithms. Combining the RE algorithm with a method that has been used to lower latency in TB architectures, we develop a new hybrid survivor path architecture. Its implementation efficiency extends to a larger number of states, with only small sacrifices in latency compared to pure RE processing.
This part is an extended version of:

- M. Kamuf, V. Öwall, and J. B. Anderson, "Survivor path processing in Viterbi decoders using register-exchange and trace-forward," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, publication scheduled for June 2007.

Part III: Designing a Flexible Trellis Decoder

Flexible channel decoding is important if one has to deal with varying channel conditions. If there is low channel signal-to-noise ratio (SNR), error-correcting capability is crucial to maintain a basic communication link. Or, when there is higher SNR, more information per channel use can be transmitted. Two candidate transmission schemes suit these requirements, convolutional coding with Gray-mapped QPSK, and trellis-coded modulation using larger constellations. The trellis diagram of the respective encoders consists of radix-2 and radix-4 butterflies, which requires two different processors in a straightforward implementation. By emulating the radix-4 butterfly with radix-2-based computation units, flexibility is provided while efficiently reusing hardware. It is shown that other processing parts benefit from this time-multiplexed approach, too. Furthermore, synthesized decoders indicate that providing more than two transmission rates is beneficial, considering that the initial price for rate flexibility has already been paid. The flexible Viterbi decoder was fabricated in a 0.13 μm digital CM0S process and its functionality was verified in a test environment.
This part is an extended version of:

- M. Kamuf, V. Öwall, and J. B. Anderson, "Optimization and implementation of a Viterbi decoder under flexibility constraints," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, submitted.

- M. Kamuf, V. Öwall, and J. B. Anderson, "Architectural considerations for rate-flexible trellis processing blocks," in *Proceedings of IEEE International Symposium on Personal, Indoor, and Mobile Radio Communication*, Berlin, Germany, Sept. 2005, pp. 1076–1080.

Chapter 2

Channel Coding and Decoding

This chapter deals with basics of channel coding and its decoding algorithms. It is by no means a survey of this vast area; instead, only topics relevant to this thesis are discussed such that the contributions in Parts I–III can be followed and appreciated. For a more general overview of the field, a good start are the review papers of [29, 46].

Following is a brief description of the simple communication model that is assumed in the sequel. This model also helps to understand the purpose of channel coding. Then, two popular coding approaches are discussed more thoroughly: convolutional coding together with Gray-mapped signal constellations and set-partition coding. Decoding algorithms are presented from their theoretical background along with a basic complexity comparison.

Consider the block diagram of the simplified communication system in Figure 2.1. It consists of an information source (not explicitly drawn) that emits data symbols $\{u_k\}$. A channel encoder adds some form of redundancy, possibly jointly optimized with the modulator, to these symbols to yield the code symbol sequence $\{c_k\}$, where c_k denotes an $\mathcal{M}$-ary transmission symbol. Linear modulation is assumed, that is, modulation is based on a linear superposition of (orthogonal) pulses. The signal sent over the channel is therefore

$$s(t) = \sum_k c_k \cdot w(t - kT_s),$$

where $w(\,)$ is the pulse waveform and T_s is the symbol time. The waveform channel adds uncorrelated noise $n(t)$ to the signal, which results in the waveform $r(t)$ at the receiver. For the remainder, the disturbance introduced by the channel is assumed to be additive white Gaussian noise (AWGN). That is,

$$\mathcal{E}\{n(t)\} = 0$$
$$\mathcal{E}\{|n(t)|^2\} = N_0/2.$$

The received waveform $r(t)$ is demodulated to yield a discrete sequence of (soft) values $\{y_k\}$. Based on these values, the channel decoder puts out an estimate $\{\hat{u}_k\}$ for the data symbols $\{u_k\}$.

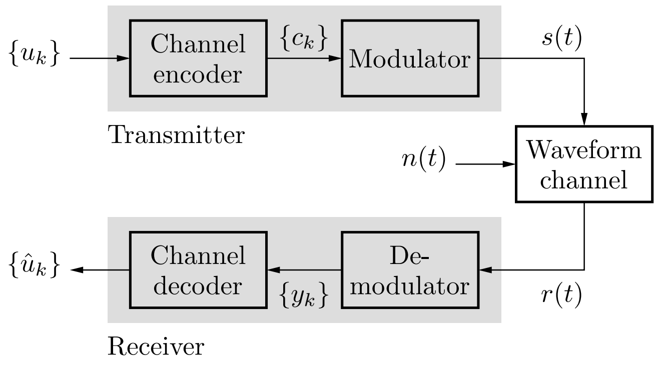


Figure 2.1: A simplified communication system.

According to Shannon [85], reliable communication with arbitrarily low bit error rate (BER) in the AWGN channel can be achieved for transmission rates below

$$C = \frac{1}{2}\log_2\left(1 + \frac{2E_s}{N_0}\right) \quad \text{(bits/dimension)}.$$

If there are J orthogonal signal dimensions per channel use, the transmission rate of a (coded) communication system is defined as

$$R_d = \frac{\log_2 \mathcal{M}}{J} \cdot R_c \;\; \text{(bits/dimension)}, \tag{2.1}$$

where $\mathcal{M}$ is the number of possible symbols per channel use and $R_c < 1$ denotes the code rate of the channel code in data bits/code bits. For example, a communication system with a channel code of rate $R_c = 1/2$ per channel use and a 16-QAM constellation, that is, $\mathcal{M} = 16$ and $J = 2$, has a transmission rate of $R_d = 1$ bit/dimension.

For equiprobable signaling, the energy devoted to a transmission symbol is expressed as

$$E_s = \frac{1}{\mathcal{M}} \sum_{i=1}^{\mathcal{M}} \|c_i\|^2,$$

or, alternatively, the energy per data bit is

$$E_b = \frac{E_s}{\log_2 \mathcal{M} \cdot R_c}. \tag{2.2}$$

2.1 Channel Coding

Plainly spoken, a good channel code reduces the necessary E_b to achieve the same BER over a noisy channel as an uncoded transmission system of equal transmission rate $R < C$. This reduction is referred to as coding gain.

The BER of many communication systems can be estimated in closed form based on the union bound [78]. Essentially, BER depends on the two-signal error probability, that is, the probability that one signal is mistaken for another upon decoding, and the minimum distance between signals. This probability resembles

$$p_e \sim \frac{2\mathcal{K}}{\mathcal{M}} Q\left(d_{\min}\sqrt{\frac{E_b}{N_0}}\right), \tag{2.3}$$

where $\mathcal{K}$ is the number of signal pairs that lie at distance $d_{\min}$ apart from each other and $Q(\,)$ is the complementary error function defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-u^2/2)\,\mathrm{d}u.$$

In practice, BER is estimated by computer simulations of the underlying communication model. From Equation 2.3 the task of the channel code (together with the modulator) becomes apparent: either increase $d_{\min}$, or decrease $2\mathcal{K}/\mathcal{M}$, or both. Then, E_b can be lowered for the same BER.

There are two major classes of binary channel codes: block codes and convolutional codes. In the context of this thesis, only the latter codes are considered since they are widely applied in today's communication systems. Nevertheless, the rediscovery of low-density parity-check codes [49] might reclaim some share from convolutional-based coding in these systems in the near future.

2.1.1 Convolutional Codes

Discovered by Elias [35] and extensively discussed in, for example, [44] or [61], convolutional codes operate on continuous (theoretically infinite) symbol streams. In practice, though, convolutional codes are terminated in some way to carry out the decoding on blocks of codewords.

Generation and Realization

A rate $R_c = b/c$, $b < c$ transfer function matrix for a convolutional code can be expressed as

$$G(D) = \begin{pmatrix} g_{11}(D) & \dots & g_{1c}(D) \\ \vdots & \ddots & \vdots \\ g_{b1}(D) & \dots & g_{bc}(D) \end{pmatrix}, \tag{2.4}$$

where

$$g_{ij}(D) = f_{ij}(D)/q_{ij}(D),\ \ i = 1, \dots, b,\ \ j = 1, \dots, c$$

are rational transfer functions and f, q are polynomial. D is the delay operator [44] which signifies a delay of one symbol position. For Equation 2.4 to be an encoding matrix, it must have full rank and be realizable, that is, $q_{ij}(D)$ must be delay free ($q_{ij}(0) = 1$).

From Equation 2.4, several classes of encoding matrices can be derived. For example, $G(D)$ is called polynomial if all entries are polynomials, that is, $q_{ij}(D) = 1$. Furthermore, if the b data bits appear unaltered among the c code bits, the encoding matrix is called systematic and has form

$$G(D) = \begin{pmatrix} I_b & \mathcal{R}(D) \end{pmatrix}, \tag{2.5}$$

where I_b is a $b \times b$ identity matrix and $\mathcal{R}(D)$ a $b \times (c - b)$ matrix whose entries are rational functions of D. As a convention, the systematic bits appear in the first b positions.

Let

$$\nu_i = \max_{1 \le j \le c} \{\deg g_{ij}(D)\} \tag{2.6}$$

denote the constraint length of the ith input of an encoding matrix. Then, the memory μ of this matrix is

$$\mu = \max_{1 \le i \le b} \{\nu_i\}$$

and the overall constraint length becomes

$$\nu = \sum_{i=1}^{b} \nu_i.$$

As an example of Equation 2.5, consider the systematic rate 2/3 encoding matrix

$$G(D) = \begin{pmatrix} 1 & 0 & \frac{D}{1+D^2+D^3} \\ 0 & 1 & \frac{D^2}{1+D^2+D^3} \end{pmatrix}. \tag{2.7}$$

Its memory is $\mu = 3$ and its overall constraint length is $\nu = 6$.

A rate R_c binary convolutional encoder is a realization of Equation 2.4 by a linear sequential circuit with tapped delay lines. It consumes b data bits and produces c code bits for every time step.

A convolutional encoder is realized either in controller or observer form. In controller form, each input (row of $G(D)$) is assigned to a separate shift register. The feedback encoder for Equation 2.7 in controller canonical form is depicted in Figure 2.2. Since g_{13} and g_{23} are rational, the encoder has feedback. Encoder tap sets are expressed as left-justified octals $(\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2) = (54, 20, 10)$, where $\mathbf{g}_0$ is the feedback tap set. According to Equation 2.6, $\nu_1 = 3$ and $\nu_2 = 3$. That is, the number of delay elements in the $b = 2$ shift registers is 3 in both cases. In total there are $\nu = \nu_1 + \nu_2 = 6$ delay elements needed in this realization. It is seen that for the controller form the number of delay elements is equal to ν.

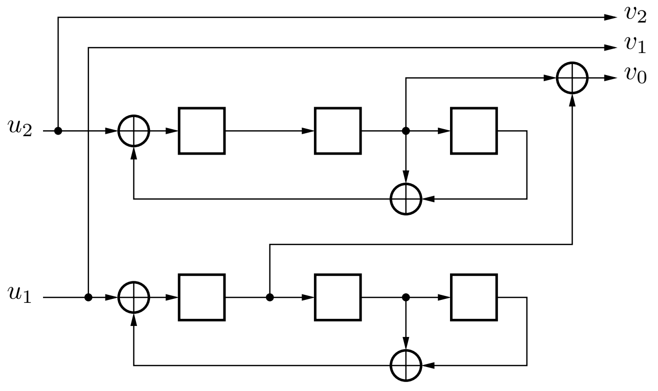


Figure 2.2: A systematic rate 2/3 code with encoding matrix from Equation 2.7 realized in controller canonical form. $\oplus$ denotes a modulo-2 addition.

Another equivalent realization of Equation 2.7 is the observer canonical form, and the respective encoder is shown in Figure 2.3. It has one shift register for every output (column of $G(D)$). Encoder tap sets are $(\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2) = (15, 02, 04)$ (right-justified octals), where $\mathbf{h}_0$ is the feedback tap.[†] Close examination reveals that this realization only has 3 delay elements, which is equal to the memory μ of the encoding matrix.

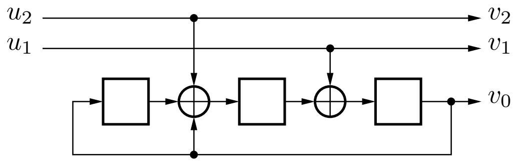


Figure 2.3: A systematic rate 2/3 code with encoding matrix from Equation 2.7 realized in observer canonical form.

Generally, the total number of delay elements depends on the encoder realization, not the encoding matrix. As occurred in the example, systematic rate $b/b+1$ codes in observer form are always minimal, that is, the number of delay elements is equal to μ. Minimal encoders are desirable since decoding complexity is proportional to the number of encoder states.

Code Representation

From now on, the total number of delay (memory) elements in an encoder realization is referred to as the encoder memory m. The contents of the memory elements (read from left to right) define the state of an encoder, and hence the behavior of this circuit can be described by a finite state machine (FSM). For example, Figure 2.4 shows the state diagram of a convolutional code with rate 1/2 polynomial encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ realized in controller form. The corresponding encoder tap sets are $(\mathbf{g}_1, \mathbf{g}_2) = (7, 5)$.

The number of states in such an FSM is $N = 2^m$ and there are 2^b edges entering and leaving each state. Drawn along these edges are the b data bits that caused the state transition along with the corresponding c-bit code symbol, separated by a slash, for example, "0/10".

In order to capture the evolution of the encoder states in time, Forney [43] introduced the trellis representation, see Figure 2.5. A trellis is one of the most convenient ways to visualize the principle of the decoding algorithms discussed in Section 2.2.

[†] To distinguish between realizations, controller form tap sets use left-justified octals and letter $\mathbf{g}_\ell$, observer form tap sets are right-justified and use $\mathbf{h}_\ell$.

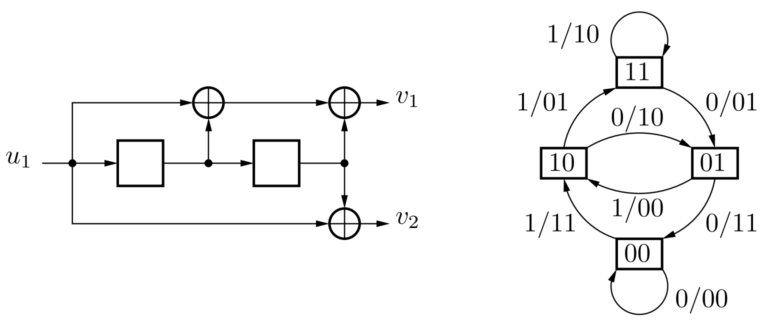


Figure 2.4: Nonsystematic feedforward encoder with tap sets (7,5) and its state diagram.

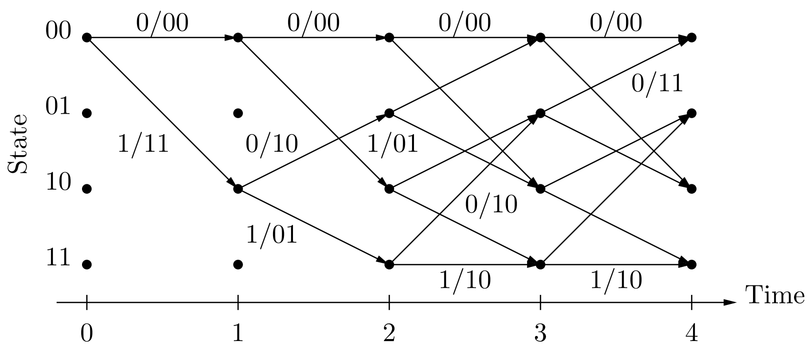


Figure 2.5: Trellis diagram of (7,5) encoder.

Nodes in the trellis represent states in the FSM. After m steps, the trellis is said to be fully extended and the branch pattern repeats indefinitely until the data stops. Usually, states are expressed as decimals, with the least significant bit equal to the oldest bit that entered the shift register. For example, binary state "10" equals state "2".

For any convolutional encoder, a trellis stage can be collapsed into $N/2^b$ disjoint butterflies of size (radix) 2^b. The most important butterflies are radix-2 and radix-4, which correspond to an encoder with $b = 1$ and $b = 2$ inputs, respectively. These butterflies are shown in Figure 2.6. State labels are denoted for encoders in controller canonical form, where $n \in [0, N/2^b - 1]$. Note that state transitions are independent of the encoder tap sets, that is, transitions solely depend on the input fed into the leftmost memory element. This is unlike the observer canonical form, where all memory elements can be fed in parallel by the input, depending on the tap sets; see Figure 2.3.

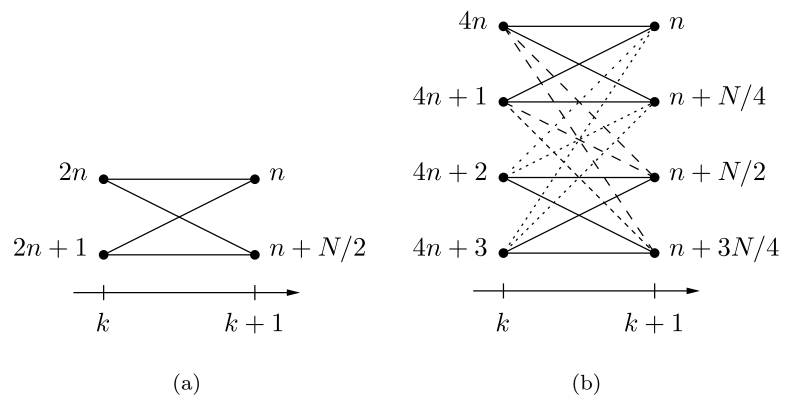


Figure 2.6: A radix-2 butterfly (a) and a radix-4 butterfly (b). The radix-4 butterfly can be decomposed into four radix-2 butterflies, indicated by the different line styles.

Error Performance

The Hamming weight of a code sequence $\mathbf{v}$ is the number of non-zero positions in $\mathbf{v}$. The Hamming distance d_{H} between two codewords $\mathbf{v}_1$ and $\mathbf{v}_2$ is the number of positions where the two sequences differ. Error performance of a convolutional code is determined by the minimum Hamming distance between any two $\mathbf{v}_1$ and $\mathbf{v}_2$ that differ in the first position. This expression, also called free distance of the code, becomes

$$d_{\mathrm{f}} \equiv \min_{\mathbf{v}_1 \neq \mathbf{v}_2} \{d_{\mathrm{H}}(\mathbf{v}_1, \mathbf{v}_2)\}.$$

Since convolutional codes are linear, d_{f} is equal to the minimum Hamming weight of any non-zero codeword. Visualized with the trellis diagram, this is the path with the lowest weight that merges with the all-zero path. This path covers at least ν stages.

As a first-order estimation, d_{f} grows linearly with the code's constraint length ν. Now reconsider Equation 2.3. If we assume binary signaling per dimension on the AWGN channel, the factor $2\mathcal{K}/\mathcal{M}$ becomes one and we can basically replace $d_{\min}$ by d_{f}. Calculations [61] in signal space for the bit error probability ultimately lead to

$$p_e \approx Q\left(\sqrt{2d_{\mathrm{f}}R_cE_b/N_0}\right).$$

That is, the error probability decreases with increasing d_{f}, This matches our intuition that increasing the code constraint length leads to lower BER.

There are many other parameters that ultimately influence the performance of a channel code. For example, it was found out in [3] that for the important class of rate 1/2 codes, systematic feedback encoders perform better than non-systematic feedforward ones under almost all circumstances.

Puncturing

Intuitively, there are two transmission scenarios to be distinguished: power-limited and bandwidth-limited scenarios. For the former class, low rate codes with large constraint length will do an excellent job [61]. However, the lower the code rate, the lower the amount of data transmitted per channel use; see Equation 2.1. To maintain the original data rate in bits/s, the symbol time has to be decreased, which in turn increases the bandwidth of the transmitted signal. If bandwidth were unlimited, this transmission is only bounded by the amount of energy per symbol that has to be provided for synchronization at the receiver [4]. From Equation 2.2, E_s becomes smaller the lower the code rate.

High-rate codes are crucial if one cannot tolerate bandwidth expansion. However, as b increases, so does the complexity of the butterflies (Figure 2.6) and thus the decoding. Puncturing [22, 104] is a convenient way to increase the code rate of a convolutional code without increasing trellis complexity.

The principle is best explained with the trellis diagram in Figure 2.7, which is based on the rate 1/2 (7,5) encoder from Figure 2.5. The punctured code is derived by deleting bits from the original code stream to be transmitted. In this example, there is one code bit omitted in every other trellis stage, which yields three code bits for two data bits, as indicated by the shaded box. The resulting code rate is thus 2/3.

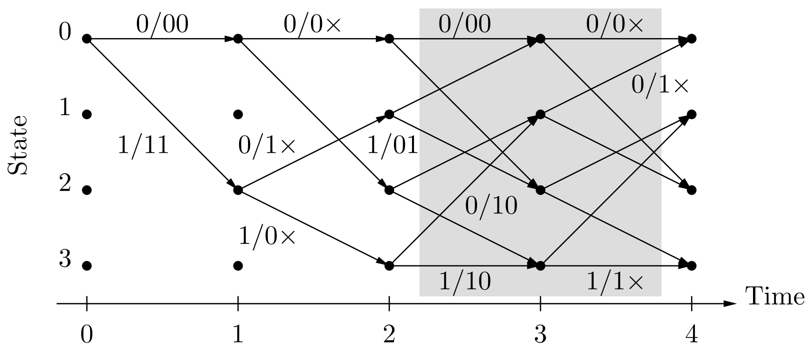


Figure 2.7: Trellis diagram of punctured rate 1/2 code to yield a 2/3 code. Bits denoted "×" are not transmitted.

Note that the butterfly structure has not been altered in a trellis stage. Therefore, the decoding architecture for the mother code can be used for punctured codes that are derived from it. Upon decoding, the punctured bit positions are simply neglected when calculating branch transition probabilities.

From the results in [22, 104], the free distance of these punctured codes is not significantly altered compared to the respective best non-punctured codes of equal rate. Thus, asymptotic error performance is only slightly degraded. If the punctured codes are derived from the same mother code, the codes are called rate-compatible [52]. This is a desirable feature if one wants to vary code rate depending on the channel conditions, since encoder and decoder can be easily reused.

2.1.2 Trellis-coded Modulation

Convolutional coding as discussed in the previous section is generally used together with Gray-mapped signal constellations. That is, code bits of signal points that are $d_{\min}$ apart differ in only one bit position. Apparently, coding and modulation are treated as separated entities in this approach. The redundancy introduced by channel coding causes bandwidth expansion.

Towards the end of the 1970's, Ungerböck [95, 96] addressed the issue of bandwidth expansion by combining coding and modulation. According to him, "redundancy" is now provided by using an expanded signal set and the coding is done directly on the signal sequences. What follows is a brief introduction into the concept of set-partition coding to familiarize the reader with the notation.

Apart from the original publications by Ungerböck, the textbook by Anderson and Svensson [6] treats the topic more thoroughly.

From now on, two-dimensional orthogonal ($\mathcal{I}/\mathcal{Q}$) signal constellations, for example, PSK or QAM, are assumed. Also, transmission rates R are now expressed in bits per two-dimensional channel use, which means omitting the dimensional factor J in Equation 2.1. Any such constellation has a certain $d_{\min}$, which essentially determines BER according to Equation 2.3. Recall that increasing $d_{\min}$ lowers the BER. In trellis-coded modulation (TCM), this increase is achieved by partitioning a (master) constellation into subsets. This split is done according to Ungerböck's rules such that the distance between signals belonging to the same subset is maximized. As is seen from Figure 2.8, every split of a uniform QAM constellation (here 16-QAM) increases $d_{\min}$ in the subset by a factor $\sqrt{2}$. If the lattice at the root has distance 1 between constellation points, the distance in the subsets at the lowest branch is $2\sqrt{2}$, denoted same-subset minimum distance d_{ss}. This distance is the ultimate limit on how well a TCM code can perform.

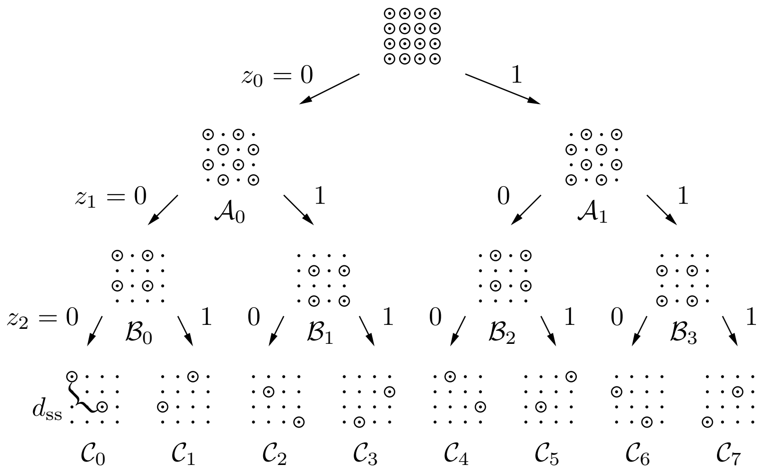


Figure 2.8: Set-partitioning according to Ungerböck. Picture adapted from [6].

In general, if $\mathcal{M}$ denotes the size of the master constellation, there are $\log_2 \mathcal{M} - 1$ possible splits to finally produce $\mathcal{M}/2$ subsets with 2 signals. Of course, one could split one more time to arrive at one signal per subset. This transforms the set-partition problem back to mapping code symbols efficiently to constellation points. However, it is not considered a set-partition code in the traditional sense, and will not be discussed further.

Figure 2.9 shows a 16-QAM constellation, which is divided into 8 subsets with 2 signal points per subset. This division stems from successive splits as in Figure 2.8, and the bit-to-constellation point mapping is derived from these splits. The subset number is the three rightmost bits ($z_2 z_1 z_0$) of a constellation point indicated by the underbrace. Then, the signal point in the subset is distinguished by the underlined bit. Apparently, the constellation mapping is not Gray anymore; only within a subset Gray-mapping is used.

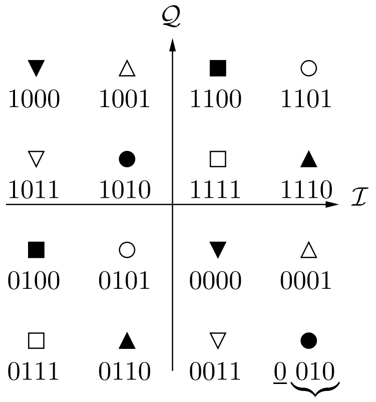


Figure 2.9: 16-QAM constellation divided into 8 subsets (marked by the different symbols) with 2 signal points each.

In order to achieve d_{ss}, only certain sequences of subsets are allowed. This sequencing is done by a tapped shift register, a convolutional encoder. In the context of TCM, this entity is called subset selector. Figure 2.10 shows a complete rate-R TCM encoder. It takes in R bits per symbol time. b bits are input to the subset selector that puts out $c > b$ bits z_i that determine the subset to be used for that symbol time. Note that branches in a trellis diagram now carry subsets, not code symbols as in the case of convolutional coding.

The remaining $R - b$ bits, sometimes called "uncoded" bits, are used to choose the signal sent in a subset. This implies that the trellis diagram has parallel transitions. As a consequence, decoding complexity is increased since the most likely of these subset signals has to be determined before one can calculate the branch transition probabilities. This process is called subset decoding.

The inter-subset minimum distance d_{is} is a measure similar to the Hamming distance between two code sequences, where the two outputs are in different subset sequences. In case of convolutional coding, the number of differing bit positions determines d_{f}. Now, the Euclidean distance between the signal

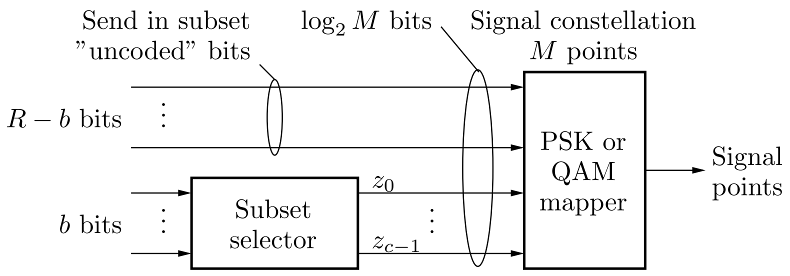


Figure 2.10: A generic rate-R TCM encoder.

sequences has to be evaluated. Since the performance of the TCM code is determined by $\min\{d_{\text{is}}, d_{\text{ss}}\}$, the goal is to design a clever enough subset selector that achieves $d_{\text{is}} \geq d_{\text{ss}}$.

Coding gain of a TCM code is measured compared to an uncoded (QAM) transmission of equal rate R. To achieve the same transmission rate as an uncoded system, and carrying out all subset splits, one needs a rate $b/b+1$ convolutional code for subset selection. Traditionally, the encoder is of systematic feedback type and often realized in observer canonical form (see Figure 2.3) since this leads to a minimal encoder.

To emphasize the importance of TCM in the high signal-to-noise ratio (SNR) region, consider Figure 2.11. It compares rate 1, 3, and 5 transmission schemes using QPSK, 16-QAM, and 64-QAM constellations, respectively. The TCM subset selectors are rate 1/2 for QPSK and 2/3 for the two multi-level constellations. The competing convolutional codes are rate 1/2, 3/4, and 5/6. They use the same constellations as the TCM schemes, albeit with Gray-mapping. Here, the higher rate codes are achieved by puncturing the rate 1/2 code. Corresponding puncturing patterns are found in [104]. In all cases, the encoders have 8 states. For the multi-level constellations, the gain at BER of 10^{-5} of TCM compared to the Gray-mapped system is around 1.3 dB. However, in the QPSK case with rate 1/2 coding, that is, for low transmission rates, the TCM code appears to be somewhat weaker.

2.2 Decoding Algorithms

From the considerations in Section 2.1.1, the trellis created by a convolutional encoder can be interpreted as finite-state discrete-time Markov source. Denote by $X_k \in [0, N-1]$, $k \in \mathbb{Z}$, a possible state of the encoder at time k. At the receiver side, the probability of a trellis transition from state X_k to X_{k+1} and

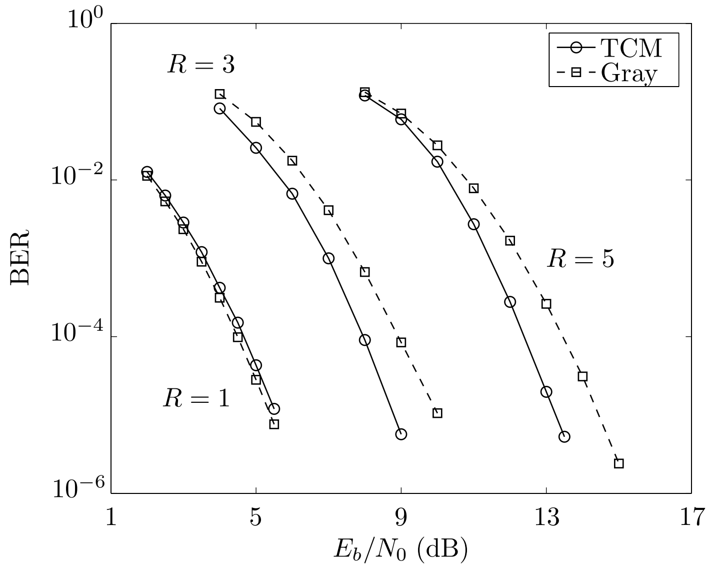


Figure 2.11: Performance comparison of rate-R transmission schemes using TCM or convolutional coding with Gray-mapped constellations.

the outcome $\mathbf{y}_k$ is given by

$$p(X_{k+1}, \mathbf{y}_k | X_k) = p(\mathbf{y}_k | X_k, X_{k+1}) \ \Pr(X_{k+1}|X_k). \tag{2.8}$$

Here $p(\mathbf{y}_k|X_k, X_{k+1})$ is the likelihood function of the received symbol $\mathbf{y}_k$ given the transition (X_k, X_{k+1}) and $\Pr(X_{k+1}|X_k)$ is the transition's *a priori* probability. For convolutional codes, there are c code symbols along a trellis branch and thus $\mathbf{y}_k = (y_{0,k} \cdots y_{c-1,k})$. Depending on the code rate R_c and the transmission scheme, these $y_{i,k}$ stem from one or several i.i.d. code symbols. For TCM codes, there are subsets along the branches. These subsets consist of two-dimensional signals and $\mathbf{y}_k$ is a two-dimensional signal.

When a demodulated noisy value $\mathbf{y}_k$ is received from an AWGN channel with variance $\sigma^2 = N_0/2$, the likelihood function becomes

$$p(\mathbf{y}_k|X_k, X_{k+1}) = \frac{1}{\sqrt{\pi N_0}} \exp\left(-\frac{|\mathbf{y}_k - c_k|^2}{N_0}\right).$$

One can take the logarithm of Equation 2.8 and scale with $-N_0$ to yield the branch metric (BM)

$$\begin{aligned}\lambda(X_k, X_{k+1}) &\equiv -N_0 \log p(X_{k+1}, \mathbf{y}_k | X_k) \\ &= |\mathbf{y}_k - c_k|^2 - N_0 \log \Pr(X_{k+1}|X_k) - \underbrace{N_0 \log \frac{1}{\sqrt{\pi N_0}}}_{\text{constant}} . \end{aligned} \tag{2.9}$$

The first term in Equation 2.9 corresponds to the squared Euclidean distance between the received symbol $\mathbf{y}_k$ and the expected symbol c_k along the branch (X_k, X_{k+1}). The second term is the weighted *a priori* probability of the branch. The constant can be neglected in the calculations since it contributes equally to all $\lambda(\,)$.

Based on the previous notations, consider a received symbol sequence $\mathbf{y} = \{\mathbf{y}_k\}$. Since the channel is memoryless, maximum likelihood (ML) and maximum *a posteriori* (MAP) sequence estimates can be expressed as finding the i that achieves

$$\min_i \|\mathbf{y} - \mathbf{c}_i\|^2 \tag{2.10}$$

and

$$\min_i \left\{ \|\mathbf{y} - \mathbf{c}_i\|^2 - N_0 \sum_i \log \Pr(X_{i+1}|X_i) \right\} , \tag{2.11}$$

respectively. Clearly, ML and MAP decoders would estimate the same symbol sequence if all symbols were equally likely, that is, the *a priori* probability is equal for all branches. Then, the second term in Equation 2.11 is the same for all branches (X_i, X_{i+1}), and can thus be removed in calculating the branch metrics. If there is *a priori* information about the transition, though, the decoding might give different results for ML and MAP. In any case, ML minimizes the sequence error probability, whereas MAP can be set up so as to minimize the bit error probability [8].

In the following, two prominent decoding algorithms, one ML and one MAP, are discussed.

2.2.1 Viterbi Algorithm

In 1967, Andrew J. Viterbi published a means of decoding convolutional codes with an optimum non-sequential algorithm that now carries his name [98], the Viterbi algorithm (VA). Until that time, decoding was mainly done sequentially [36,59,103] or with a method called threshold decoding [70]. Although the complexity of these algorithms is independent of the memory of the code, there are cases when they cease to work (above the computational cutoff rate) since the number of paths in a code tree diagram increases exponentially with the length of the sequence. This problem is avoided by the VA, whose computational effort only increases linearly with the length of the trellis.

Omura [73] observed that the VA is a solution to dynamic programming, for example, the shortest-route problem visually stated in [71]. Forney [45] showed later that the VA in fact is ML and he connected Viterbi's work to the concept of trellises, which illustrates very well how the algorithm works.

In its most general form, the VA is a MAP sequence estimator for a finite-state discrete-time Markov process observed in memoryless noise. The BMs to be used are the ones from Equation 2.9. If *a priori* information is not available, it is seen from Equation 2.10 and Equation 2.11 that ML and MAP estimates become the same because Equation 2.9 simplifies to

$$\lambda(X_k, X_{k+1}) = |\mathbf{y}_k - c_k|^2, \tag{2.12}$$

which is the squared Euclidean distance. Therefore, the VA does not need an estimate of the noise in its calculations, which simplifies receiver implementation. If one uses binary antipodal transmission per signaling dimension, it will be shown in Part I how the calculation of Equation 2.12 simplifies to additions only.

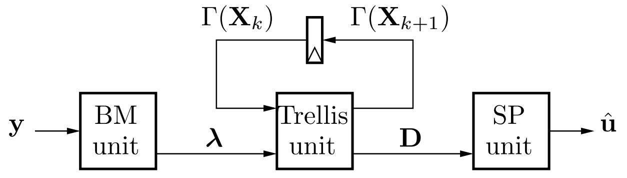


Figure 2.12: Block diagram of a Viterbi decoder.

From a computational view, the algorithm can be described by three parts; see Figure 2.12. The task of the BM unit was already laid out in the previous paragraphs. The BMs $\boldsymbol{\lambda} = \{\lambda(X_k, X_{k+1})\}$ are consumed by the trellis unit, which discards unlikely branches from the trellis diagram. This procedure is based on *The Principle of Optimality* [10]. Applied to the VA, the principle can be recast as follows. Whenever a number of paths merge into one state, only the most likely path (with the best metric, here the minimum accumulated metric) has to be retained. Obviously, for all extensions to these paths, the previous path which is currently better will always be better. The operation just described is called an add-compare-select (ACS) recursion. For every state X_{k+1}, such a recursion yields an updated state metric (SM) $\Gamma(X_{k+1})$ based on the previous SMs $\Gamma(X_k)$ connected to X_{k+1} and the current BMs $\lambda(X_k, X_{k+1})$, that is,

$$\Gamma(X_{k+1}) = \min_{(X_k, X_{k+1})} \{\Gamma(X_k) + \lambda(X_k, X_{k+1})\}. \tag{2.13}$$

From Equation 2.13 it becomes apparent that the processing complexity of the trellis unit increases both with the number of states $N = 2^m$ and branches 2^b per node that connect these states. Today's (parallel) hardware implementations of the VA are usually restricted to moderate m, that is, smaller than 9 [26]. The algorithm's complexity is also visualized by the butterflies in Figure 2.6. The higher the radix of the butterfly, the more transitions must be evaluated. Note, though, that there are suboptimal algorithms [2] with reduced complexity that only update a predefined number of states and survivor paths.

As an example, Figure 2.13 shows an ACS recursion for a trellis with two branches merging into one state. The originating states are labeled 0 and 1. The solid path is the survivor path and the decision $\mathcal{D}$ taken for state $X_{k+1} = 0$ is $\mathcal{D}_{(X_{k+1}=0)} = 0$. The decision bit indicates which state the surviving branch is connected to.

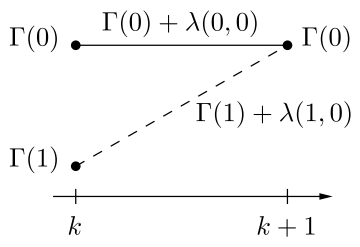


Figure 2.13: ACS recursion for two branches, that is, the number of data bits per trellis stage is $b = 1$.

At every trellis stage $N = 2^m$ paths need to be updated. Therefore, the trellis unit puts out a vector $\mathbf{D} = \{\mathcal{D}_{X_{k+1}}\}$ of N decision symbols about surviving branches. These symbols are stored in a memory that resides in the survivor path (SP) unit in order to reconstruct the data symbols that caused the state transitions.

Once the end of the trellis is reached, either because the data stream ended or the trellis is terminated into a pre-defined state, the decoding of the data symbols starts. One simply follows the survivor path linked to the (known) ending state backwards by means of the decision symbols. At every stage, these symbols point to the predecessor of the current state, and by this recursion one yields the ML state sequence. Linked to this sequence is a unique data bit sequence $\hat{\mathbf{u}}$, and the decoding is finished. Were it not for latency and/or storage requirements for the decision symbols, which both grow linearly with the length of the trellis, this is the most straightforward decoding method. Excessive latency and/or storage can be avoided by terminating the trellis into a defined state on the right. This procedure effectively transforms a convolutional code into a block code. However, a termination sequence that contains no actual information (zero-tail termination), causes an overall information rate loss, which may become critical for small block lengths. These practical limitations, may require the decoding to be carried out midstream.

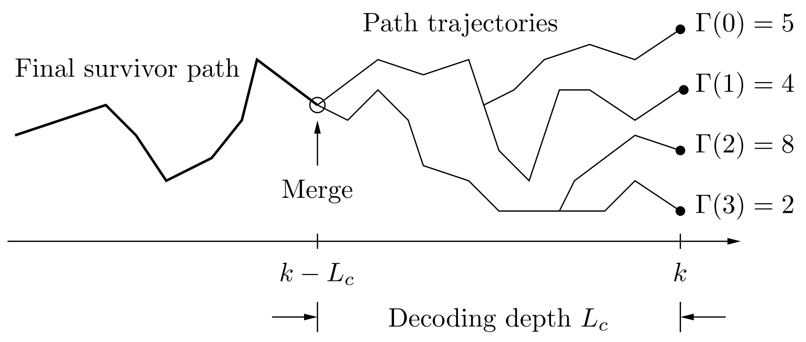


Figure 2.14: Evolution of N path trajectories that merge into final survivor path. For simplicity, only 4 trellis states are drawn.

Thankfully, the trellis of the described shift register process has a useful property. Consider Figure 2.14. At time k, the VA keeps track of N survivor paths. When traced back over time, these paths merge into a single path, indicated by the trajectories in the figure. The path found is the final survivor path for trellis steps smaller than $k - L_c$. Here, L_c is the necessary decoding depth for the underlying code. An estimation of L_c for convolutional codes is given in [5]. Asymptotically, the depth follows the rule [7]

$$L_c \approx \frac{d_{\mathrm{f}}}{c \cdot h_{\mathrm{B}}^{-1}(1 - R_c)}, \tag{2.14}$$

where $h_{\mathrm{B}}^{-1}(\,)$ is the inverse of the binary entropy function and there are c bits on a branch. At rate 1/2, L_c becomes approximately $4.54\,d_{\mathrm{f}}$. Note that higher rate codes, whether derived by puncturing or not, generally have a larger L_c. If the decoder traces back at least L_c steps, one achieves the optimum ML estimate given that $\lfloor \frac{d_{\mathrm{f}}-1}{2} \rfloor$ or fewer errors have occurred. Under this assumption, all survivor paths are guaranteed to have merged at this stage. In practice, the required depth L for the decoder to experience negligible BER degradation is about L_c and is found by computer simulation. With this L, the survivor paths have merged with sufficiently high probability. As a rule of thumb, rate 1/2 codes need to be observed for about five times the constraint length, which was experimentally shown in [57]. The decoding latency is at least L, and the memory depth for the decision symbols decreases at best to L.

Since the starting state is not explicitly known at time k, there are in principle two methods to decide which of the N survivor paths to follow. If fixed-state decoding is employed, the decoder always starts looking back from a predefined state, for example, state 0 in Figure 2.14. Best-state decoding, on the other hand, starts from the state that currently has the best (here smallest) SM, which would be state 3 in the figure. Since all path trajectories have merged in this example, the final survivor path found is the same in both cases. If there occurred more than $\lfloor \frac{d_{\mathrm{f}}-1}{2} \rfloor$ errors over the L_c stages shown in Figure 2.14 it is possible that not all (in the worst case none) of the N paths have merged with the actual ML path. In this case, the decoder produces erroneous bits. Generally, best-state decoding requires a smaller L than fixed-state decoding for the same performance; however, best-state decoding needs extra calculations (comparisons) to find the most likely starting state. Algorithms and architectures for survivor path processing are discussed in detail in Part II.

Finally, one should note that although the VA by definition provides a "hard" data sequence, that is, a decoded bit is either "0" or "1", the algorithm can be augmented with a soft-output unit. "Soft" information is characterized by real numbers, not bits, and is an essential part in the decoding of concatenated coding schemes. In these schemes, an outer decoding stage, for example, a Reed-Solomon decoder, uses soft information provided by the inner decoding stage to improve the overall BER of the communication system. The soft-output VA (SOVA) is described in [9, 53]. It produces bit reliabilities by calculating the difference of the SMs along the survivor path and the next best competing path. Note that for iterative decoding schemes [11, 12], where soft *a priori* information is exchanged in turns between two component decoders, SOVA causes BER degradation compared to true MAP decoding. For the setup described in [80], SOVA needs around 0.7 dB extra to achieve BER 10^{-4}. On the other hand, SOVA requires fewer computations, and architectural optimizations for the VA can be reused as is; only small modifications to the SP unit and some extra storage for the SM differences have to be provided [60].

2.2.2 BCJR Algorithm

Another algorithm named after its inventors, Bahl, Cocke, Jelinek, and Raviv, is the BCJR algorithm [8]. It can be set up to provide the most likely symbol at each time, equivalent to minimizing bit error probability. Contrary to the VA, the trellis is now processed in forward *and* backward direction. What is put out are "soft" bit reliabilities for every trellis stage, just as SOVA does. As mentioned earlier, soft outputs are required for iterative decoding. From the BCJR viewpoint, Z data symbols are encoded blockwise, and thus the trellis has Z stages.

The following brief discussion is restricted to the application of the BCJR algorithm in the logarithmic domain [80]. This modification (log-MAP, max-log-MAP) circumvents multiplications and divisions, both strong operations, which were used in the original BCJR paper. Also, binary coding is assumed in the notations; a non-binary log-MAP algorithm is found in [81].

The log-likelihood ratio (LLR), which is a measure of bit u_k being 0 or 1, is defined as

$$\mathrm{LLR}(u_k) \equiv \log \frac{\Pr(u_k = 1|\mathbf{y}_k)}{\Pr(u_k = 0|\mathbf{y}_k)}. \tag{2.15}$$

It encapsulates both hard and soft information. The sign of Equation 2.15 corresponds to the hard decision, and the magnitude is the reliability estimate. For notational convenience, we also define

$$\begin{aligned} \max{}^\star(x, z) &\equiv \log(e^x + e^z) \\ &= \max(x, z) + \log(1 + e^{-|x-z|}), \end{aligned} \tag{2.16}$$

which is the Jacobi logarithm. Note that the $\max^\star$ operation is merely an (A)CS operation with an added offset [21].

Suppose the encoding process is a Markov chain and the channel is memoryless. Then, numerator and denominator in Equation 2.15 can be divided into three terms each: forward and backward SMs, and the BM for the respective state transitions. Forward SMs α are recursively calculated for $k = 0, \ldots, Z-2$ according to

$$\alpha(X_{k+1}) = \max_{(X_k, X_{k+1})}{}^\star \{\alpha(X_k) + \lambda(X_k, X_{k+1})\}, \tag{2.17}$$

where $\alpha(\mathbf{X}_0) = [0, -\infty, -\infty, \ldots]$. That is, at time 0, the metric for state 0 is initialized to 0, the remaining $N-1$ SMs to $-\infty$. Equivalently, the backward SMs for $k = Z-1, \ldots, 1$ are

$$\beta(X_k) = \max_{(X_k, X_{k+1})}{}^\star \{\beta(X_{k+1}) + \lambda(X_k, X_{k+1})\} \tag{2.18}$$

and $\beta(\mathbf{X}_Z) = [0, -\infty, -\infty, \ldots]$. Together with the BM $\lambda(X_k, X_{k+1})$ from Equation 2.9, we get the LLR as difference between soft estimates for branches with $u_k = 1$ and $u_k = 0$, respectively:

$$\begin{aligned}\mathrm{LLR}(u_k) = &\max^\star_{\substack{(X_k, X_{k+1})\\ u_k=1}} \{\alpha(X_k) + \lambda(X_k, X_{k+1}) + \beta(X_{k+1})\}\\ &- \max^\star_{\substack{(X_k, X_{k+1})\\ u_k=0}} \{\alpha(X_k) + \lambda(X_k, X_{k+1}) + \beta(X_{k+1})\}.\end{aligned} \tag{2.19}$$

An intuitive graphical illustration of these steps is found in Figure 3.13 in [6]. Note that if $\max^\star$ is exchanged with max in Equations 2.17–2.19, that is, the additional offset in Equation 2.16 is omitted, log-MAP turns into the suboptimal max-log-MAP algorithm.

The algorithm needs to store two complete sets of SMs, in total $2NZ$ values and the complete symbol (or BM) sequence. However, there are architectural optimizations [21,31,83] that also use the decoding depth property of the code such that storage requirement and latency in an implementation are greatly reduced.

2.2.3 Complexity Estimation

From a computational perspective, the "soft" bit estimates [80] of the MAP algorithm compared to the VA come at an increased cost. The number of operations per trellis stage of the discussed decoding algorithms are compared in Table 2.1.

Assume a radix-2 trellis with N states. For the VA, there are N ACS operations per trellis stage. One ACS operation requires two additions, one comparison, and one selection. A comparison is implemented as a subtraction (addition); see also Part I. A trace-back operation in the VA is counted as a selection. The log-MAP algorithm has an extra addition per ACS ($\max^\star$) operation for the offset in Equation 2.16. The number of $\max^\star$ operations is derived by observing that $\max^\star$ over N elements requires $N-1$ elementary $\max^\star$ operations according to Equation 2.16. Hence, for the log-MAP there are $4N-2$ such operations in total.

Table 2.1: Complexity comparison between the discussed decoding algorithms. Operations are counted per trellis stage.

	VA	max-log-MAP	log-MAP
Addition	$3N$	$12N+1$	$17N-1$
Selection	$N+1$	$4N-2$	$4N-2$
Look-up	—	—	$4N-2$

The numbers for (max-)log-MAP in Table 2.1 are based on the considerations in [80] without the normalization of forward and backward SMs. Therefore, $2N$ additions have vanished compared to their published numbers. Also, BMs are precalculated and are not part of this analysis, thus lowering the number of additions by 8 since there are 4 BMs with 2 additions each.

We can conclude that (max-)log-MAP decoding requires about 4–5 times more basic operations such as additions and selections than the VA. If the decoding does not rely on soft values, whose quality is especially crucial in iterative decoding, there is no reason to prefer the MAP-based decoding over the VA.

Chapter 3

Architecture to Implementation

This chapter focuses on two design domains from Section 1.1: architecture and implementation. Platforms for hardware design, that is, FPGAs and ASICs, are discussed since they leave an engineer the greatest freedom to trade design constraints for each other. Moreover, the achievable performance of these platforms enable today's high-speed mobile communication, whose underlying signal processing algorithms are computation-intensive.

Usually, the choice of platform, including the programmable ones (GPP, DSP), is made very early in the design process, that is, as soon as estimates on required throughput, processing capacity, or energy efficiency are available. Also, production cost and time-to-market are issues to be taken into account in this decision. Depending on the algorithm or size of the overall system, the platform will often be hybrid, which in turn requires hardware–software partitioning. Then, critical parts are mapped onto dedicated hardware (accelerators), while remaining parts are executed in software on DSPs or GPPs.

Once a hardware platform is decided upon, there are several architectural mapping methods, namely direct-mapping, time-multiplexing, and parallelization, to meet other design constraints such as silicon area or fine-tuned throughput. In the following, these methods are explained with Viterbi decoder architectures as the vehicle.

3.1 Implementation Platforms

A basic design flow, which was applied for the designs in Parts I and III, is introduced. This flow gives a notion of the steps that have to be carried out, and also of the complexity of designing FPGAs and ASICs.

3.1.1 The Prime Question: ASIC or FPGA?

To understand the difference between ASIC and FPGA, it is good to consider the design steps that are taken for either platform. A hardware design flow can be roughly divided into two domains, front-end and back-end design. Front-end design is concerned with high-level algorithm evaluation to arrive at trade-offs that lead to an efficient implementation. Architecture design according to the methods in Section 3.2 is followed by coding in a hardware description language (HDL) and functional simulation of the written HDL description.

So far, ASIC and FPGA design flows do not differ. At this stage in the front-end design, the choice of platform splits the design flow into two directions. As far as ASICs are concerned, the HDL code is translated (synthesized) to a netlist based on standard cells. A standard cell library is a collection of several versions of basic logic functions such as AND, OR, and storage elements such as latches or flip-flops. Generally, a logic cell of a certain type comes in several versions, that is, with different number of inputs, driving strengths, and so on. Based on this wide variety of possible combinations, the synthesis tool searches the design space for an efficient solution that fulfills the constraints set by the designer. The tool can also choose from pre-defined optimized building blocks such as arithmetic functions or memories. As an example, the different adders in Figure 1.2 were derived with these pre-defined implementations. The total control over the design process is the key to an ASIC's superiority to achieve tailored architectures with high throughput and energy efficiency. ASIC back-end design involves the actual physical placement and connections (Place & Route) of the standard cells, which are described by their layout based on geometrical shapes that are abstract views of the underlying transistors. In addition, design rule checks, circuit extraction, and post-layout simulation have to be carried out in order to guarantee correct electrical and functional behavior of the circuit.

Contrary to an ASIC, an FPGA has predefined logic and routing resources that can be configured to execute the desired functionality. Clearly, the architecture's reconfigurability inherently satisfies the need for flexibility. On the other hand, delays due to signal routing that also lowers an FPGA's logic density degrades its efficiency measures compared to an ASIC. In particular, there is an overhead in power consumption because of the routing wires' capacitance and the memory-based programming cells. Back-end design for an FPGA only involves the configurable logic blocks to be properly connected by configuring the interconnection network.

The number of ASIC back-end steps is certainly larger than for FPGAs, which is one reason for the overall longer time-to-market of an ASIC. The most important reason, though, is the time for fabrication and possible re-spins.

What has not been mentioned in the previous parts is the time for verification. When going from one design phase to another, the functional equivalence of the current design should be verified against test vectors that were generated from a high-level reference model. This verification approach is sufficient for prototype designs as the one presented in Part III. However, ASIC design for large volumes requires more advanced verification techniques, for example, formal verification or automatic test pattern generation. With these steps in mind, it is not surprising that the time for verification is longer than the actual design phase. This trend gets even more distinct for every new process technology. On the other hand, verification for an FPGA design is not as time-consuming since the underlying physical hardware is already tested and its behavior specified.

Although design time is an important economical factor, the volume cost for an ASIC is lower if large volumes are to be fabricated. However, in the prototyping context of this thesis, the efficiency reasons elaborated on in the next subsection justify the use of an ASIC.

3.1.2 Comparison of Implementation Efficiencies

To support Figure 1.3, which showed the application domains of different platforms, efficiency measures (area, throughput, and energy) have to be evaluated. To get a good estimation of these measures, appropriate cost functions must be established. Based on these, a fair comparison can be carried out. However, due to architectural differences in the platforms, this is not an easy task. Also, CMOS technology scaling effects [79] have to be considered to take into account different process generations.

There are investigations of the efficiency of Viterbi decoders implemented on several platforms. In [18, 37], the difference of an area–delay–energy cost function between DSP implementation and a physically optimized ASIC [51] is about 7 orders of magnitude! Even with a standard cell implementation, this difference is still 5 orders of magnitude. According to the authors, an FPGA lags the ASIC implementations by 2–3 orders of magnitude.

Zhang [106] carries out a similar study. His survey includes DSP architectures, datapath reconfigurable processors, FPGAs, and ASICs. The energy efficiency and computation density of his flexible Viterbi decoder implementation is about 2–3 orders of magnitude better than commercial flexible platforms. Thus, the required flexibility is efficiently provided by this so-called function-specific hardware (= flexible ASIC). From this viewpoint, it is feasible to strive for flexibility at the function level, where an algorithm is run in different configurations (see also Part III).

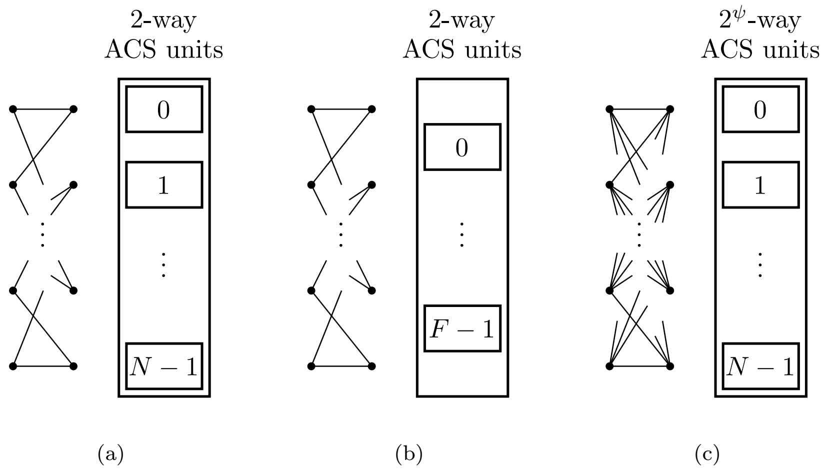


Figure 3.1: Block view of the various mapping methods, (a) direct-mapping, (b) time-multiplexing, and (c) parallelization, applied to the VA.

3.2 Mapping Methods

In the following, the main design constraints area and throughput. There are basically three mapping alternatives to satisfy either constraint without violating the other: direct-mapping, time-multiplexing, and parallelization; see Figure 3.1. Direct-mapping of N trellis states is shown in Figure 3.1(a), whereas the time-multiplexed architecture in Figure 3.1(b) only uses $F < N$ ACS units. Parallelization in Figure 3.1(c) is based on a collapsed trellis, which has 2^ψ branches leaving and entering each node [15]. The discussion of these methods in the following sections is supported by some examples taken from published Viterbi decoder architectures. Note that processing speed is related to a synchronous clocking strategy [79]. The delay of the longest (combinatorial) path between two registers is called the critical path T_{crit}. That is, the clock frequency for the registers in a design is bound to $f_{\text{clk}} \leq 1/T_{\text{crit}}$. This frequency is the main indicator of an implementation's achievable throughput.

3.2.1 Direct-mapping

An obvious choice is to assign one ACS unit per trellis state as in Figure 3.1(a), and there are numerous examples, too many to list, of this straightforward approach in the literature. Work on these state-parallel architectures is mostly concerned with optimizations for the implementation platform at hand or specialized parts of the decoder, such as the arithmetic in the ACS units or survivor path processing schemes.

Since the number of trellis states grows exponentially with the encoder memory m, a direct-mapped approach soon becomes area-inefficient for larger m, partly due to the arithmetic requirement of the ACS recursion, and partly due to the feedback connections of the ACS units. Usually, the feedback loop of the ACS recursion determines throughput. However, since a multiple-pointer trace-back architecture [30, 41] has to be employed if the number of states becomes large, the trace-back recursion can also become the throughput bottleneck. To date, however, the majority of designs rely on the de-facto standard rate 1/2, 64-state Viterbi decoder [100] which still allows an efficient direct-mapped implementation [67].

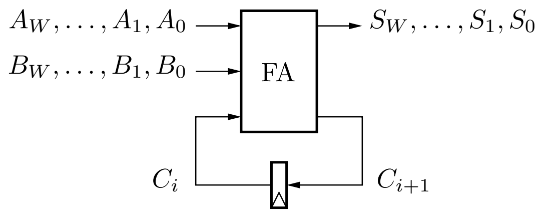


Figure 3.2: Architecture for bit-serial addition to carry out $S \leftarrow A+B$ with wordlength $W+1$. The bit-parallel equivalent is shown in Figure 1.1.

The largest (state-wise) published direct-mapped design in recent times is a 256-state Viterbi decoder [26]. The authors satisfy the arithmetic requirement of the ACS units by applying a bit-serial addition scheme [76] to reduce silicon area. Whereas one trellis stage in state-parallel architectures is usually iterated in one clock cycle, a bit-serial approach† requires as many clock cycles as the wordlength W to be processed; see Figure 3.2. Nevertheless, throughput of this design is improved compared to a competing time-multiplexed decoder implementation [32] that processes only 32 states, since the critical path of a bit-serial addition is shorter. A cluster-based ACS placement that minimizes the interconnection area between ACS units relaxes the tight area constraints even further. With these techniques, the chip still performs in accordance with the requirements of wideband code division multiple access (CDMA) systems.

†In fact, bit-serial processing is a time-multiplexed approach.

3.2.2 Time-multiplexing

Time-multiplexing, also called folding [77], is a method to minimize the silicon area of an integrated circuit by reducing the number of functional units. For example, several arithmetic operations such as additions or multipliers are assigned to one functional unit. If $\mathcal{G}$ operations are executed on a single unit, a new output sample is available after $\mathcal{G}$ clock cycles. That is, the processing time per sample is now $\mathcal{G}T_{\text{crit}}$ compared to T_{crit} in the direct-mapped case. $\mathcal{G}$ is called the folding factor. Consequently, in an area–delay design space as in Figure 1.2, folding moves an implementation to the lower right part.

Not to be forgotten in the design of folded architectures is a controller which sequences data at the right time to their respective functional units. Depending on the application, the controller design is far from a trivial task [77]. Thus, the incurred complexity has to be carefully evaluated keeping in mind the desired area savings. Another obstacle of folded architectures is the possible increase in number of registers, which is addressed by applying register minimization techniques [77]. Also, from an overall system viewpoint, there might be limitations on the maximum clock frequency of the time-multiplexed architecture, which sets an upper limit to throughput.

There are methodologies for systematic folding of the VA in order to freely trade throughput for area. Shung *et al.* [86, 87] developed a method to efficiently partition, schedule, and map N trellis states to an architecture with $F < N$ ACS units; see Figure 3.1(b). The internal parallelism created by this approach allows pipelining of the ACS units, which increases throughput, and thus yields an overall favorable area–delay product. The methodology is applicable to generic trellises that are described by the number of node-connecting edges. However, once the mapping is decided, especially if trellises with different numbers of edges (= radices) are to be processed on the same architecture, a solution for a small radix cannot emulate higher radices. Boo *et al.* [19] are mainly concerned with trellises based on radix-2 butterflies. Their approach optimizes specific ACS processing elements (PEs) and the communication between them. Based on a mathematical model, data flow and the processor mapping is derived. Contrary to the work of Shung, not only are the ACS computations embedded in the PEs; the survivor path processing is also adapted to the ratio of PEs and trellis states.

An actual state-sequential implementation is presented in [64]. This approach uses only two butterfly units, that is, 4 ACS units in total, to process 256 states. Hence, the folding factor $\mathcal{G}$ is 64. Special design techniques are applied in order to fulfill throughput and tight power requirements. For example, routing resources are minimized by applying a bit-flipping property of a butterfly. The ACS unit is based on the LSB-first approach, which lowers the critical path compared to the MSB-first method. The achieved data rate is 14.4 kbit/s, which was sufficient for use in CDMA mobile terminals at that time.

3.2.3 Parallelization

This method originally aimed at increasing the throughput of an architecture. In times when dynamic power consumption was the dominant factor in digital CMOS designs [25], parallelization was even applied to lower power consumption. This was achieved by lowering the supply voltage, which is a quadratic contributor to power, while maintaining throughput. Nowadays, as CMOS feature sizes shrink, static power consumption claims an ever-increasing share of the power budget [79]. As a first order measure, static power consumption increases with the amount of hardware. Since parallelization basically duplicates hardware, it is expected that this power source surpasses dynamic power consumption.

For forward processing algorithms, parallelization is achieved as follows. For every level of parallelism, simply provide another set of hardware that deals with the additional input. Algorithms with feedback, such as the VA, are not as easily parallelized. They require lookahead techniques [77], originally applied to infinite impulse response filters.

Ever since the VA's appearance, the vast majority of work in trellis decoding architectures has been devoted to the class of rate $1/c$ codes since these yield the simplest trellis diagram. Two branches are entering and leaving each trellis node; see Figure 2.5. Even with clever addition schemes, the ACS recursion has long been the bottleneck in Viterbi architectures. To overcome this throughput limitation, one can apply ψ levels of lookahead [38,93]. Essentially, the trellis is collapsed into one stage, which changes the radix of a trellis segment, as shown in Figure 3.3. It is concluded in [15] that collapsing a radix-2 into a radix-4 trellis is the most area-efficient transformation compared to even higher radices considering the exponential cost of carrying out 2^{ψ}-way ACS operations. In order to achieve a speed-up compared to radix-2 processing, the 4-way ACS recursion is the critical factor. The designed full-custom 4-way ACS datapath in [15] has a 17% longer delay compared to its 2-way equivalent . However, since the collapsed structure provides two bits per trellis iteration, the effective speed-up is $2/1.17 = 1.7$. Lookahead is also applied to survivor path processing by employing a technique called pre-trace-back [16]. Several implementations have later followed the trellis-collapsing technique [51, 105].

In order to achieve unlimited concurrency and thus higher throughput, minimized-method [40] and sliding-block Viterbi decoders [17] were proposed. By reformulation of the VA, these block-based processing approaches run sev-

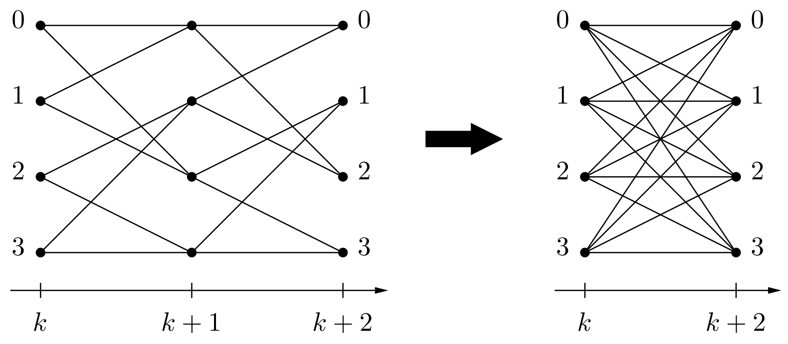


Figure 3.3: Two-stage radix-2 trellis collapsed to one-stage radix-4 trellis.

eral decoders in parallel. Due to duplication of ACS units to compute several trellis stages in forward and backward manner, the area requirement increases significantly. In practice, these concepts were only applied to a 4-state trellis [17,33], with a throughput of 1 Gbit/s.

The preceding sections gave a glimpse of the various methods that were applied to tailor architectures and implementations of Viterbi decoders to specific application needs. However, what is missing in the presented designs is variable-radix processing, which is required if an implementation has to cope with decoding applications that are based on different butterfly structures. This gap is filled by our work presented in Part III, which is best described as a state-parallel, time-multiplexed radix-flexible trellis decoding architecture.

Part I

Simplified Trellis Computational Blocks

Abstract

Simplified branch metric and add-compare-select units are presented for use in trellis-based decoding architectures. The simplification is based on a complementary property of best feedforward and some systematic feedback convolutional encoders. As a result, one adder is saved in every other add-compare-select unit and only half the branch metrics have to be calculated. For a 0.13 μm digital CMOS process, synthesized trellis computational blocks show up to 17% savings in both cell area and power consumption. A competing simplification is analyzed in terms of hardware efficiency. While the reduction can be calculated straightforwardly for our approach, the competing method relies on modified computational operations and hence this reduction is not as evident. From synthesis results, we conclude that for rate 1/2 codes, our approach is preferable for hardware implementation.

Based on: M. Kamuf, V. Öwall, and J. B. Anderson, "A hardware efficiency analysis for simplified trellis decoding blocks," in *Proceedings of IEEE Workshop on Signal Processing Systems*, Athens, Greece, Nov. 2005, pp. 128–132.
and: ——, "Area and power efficient trellis computational blocks in 0.13 μm CMOS," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005, pp. 344–347.
and: M. Kamuf, J. B. Anderson, and V. Öwall, "A simplified computational kernel for trellis-based decoding," *IEEE Communications Letters*, vol. 8, no. 3, pp. 156–158, Mar. 2004.

1 Introduction

Trellis-based decoding is a popular method to recover encoded information corrupted during transmission over a noisy channel. For example, the VA and the BCJR algorithm are two schemes that work on an underlying trellis description of the encoded sequence.

Basic computations in either algorithm involve BM calculations and ACS operations. In case of the VA, an ACS operation successively discards branches that cannot be part of the survivor path. In case of the BCJR in the logarithmic domain (log-MAP algorithm), this operation corresponds to an add-max* operation [21] to recursively calculate forward and backward SMs. This is basically an ACS operation with an added offset (ACSO). Hence, the presented considerations for the ACS hold for the ACSO as well considering the *a priori* probabilities of the branches are equal.

In this part, rate $1/c$ feedforward codes and some systematic feedback codes are considered. The best feedforward encoders of memory m are defined by c shift register tap sets which are delay free [61]. Some best systematic feedback encoders of rate 1/2 can be found in [3]. The trellis diagrams of these encoders have one thing in common: code symbols along both merging and diverging branches are always complementary. This property gives the largest growth in Hamming distance between competing paths, which ultimately determines d_f and thus BER performance. In Figure I–1, the complementary operation on $\mathbf{c} = \{c_i\}$ is defined as the complementation of its elements; that is, $c_i + \overline{c}_i = 0$.

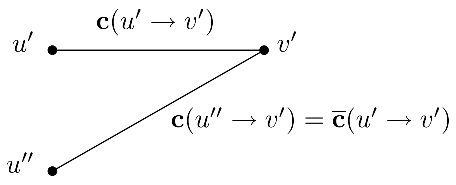


Figure I–1: Complementary code symbols along merging branches.

Based on this property, simplified architectures with reduced complexity for BM and ACS units can be derived. Two simplifications are considered, the *Offset Approach* discussed in [62, 63] and a competitor called *Differential Trellis Decoding* (DTD) [48]. For the former method, it is investigated how achieved arithmetic savings translate into area and power savings in a silicon implementation. The savings for DTD, on the other hand, are originally only described by (modified) arithmetic operations, and hence the predicted reduction is not as evident. In this work, a DTD architecture is developed based on the theoretical considerations and the two schemes are compared.

The following section describes how channel symbols are mapped to BMs. Based on these considerations, the simplified architectures are presented in Section 3. A variation of the offset approach, which trades area for speed, is also discussed. As a case study, a computational kernel for Viterbi decoding is synthesized. The synthesis results in Section 4 confirm the benefits of the offset approaches compared to a traditional setup. Also, power savings are estimated at gate level. In Section 5 hardware efficiency is compared by means of synthesized blocks for different encoder memories that show the progression of area requirement for the simplified approaches.

2 Branch Metric Computations

We take up the thread starting at Equation 2.12 in Section 2.2.1. This was the definition of the squared Euclidean distance, which is the optimal distance measure in the AWGN channel. In the following, a trellis branch (X_k, X_{k+1}) is indicated by small letters, for example, $(u' \to v')$. Where needed, time steps appear explicitly in the argument list of a variable.

The c expected coded bits along a trellis branch are denoted $b_i(u' \to v') \in \{0, 1\}$ for $i = 0, \ldots, c-1$. These bits are mapped to antipodal transmission symbols such that

$$c_i(u' \to v') = 1 - 2b_i(u' \to v').$$

The received real-numbered noisy channel values are y_i.

In accordance with this notation, one can expand Equation 2.12 to yield the BM as a superposition of the Euclidean distances in c "dimensions" as in

$$\lambda(u' \to v') = \sum_{i=0}^{c-1} y_i^2 - 2y_i c_i(u' \to v') + c_i^2(u' \to v'). \tag{I–1}$$

In Equation I–1, y_i^2 and c_i^2 contribute equally to all BMs and can thus be neglected. Also, the factor 2 can be taken out without altering future comparisons [68], and the previous equation collapses to

$$\lambda(u' \to v') = -\sum_{i=0}^{c-1} y_i c_i(u' \to v').$$

Since $c_i \in \{+1, -1\}$ the BM becomes a superposition of the received channel values as to

$$\lambda(u' \to v') = -\sum_{i=0}^{c-1} \pm y_i. \tag{I–2}$$

According to Equation I–2, $\lambda(\,)$ is a signed number. In order to choose between signed or unsigned representations, one can introduce a constant Λ in Equation I–2. Since this constant is the same for all BMs, the comparison result of an ACS operation is not altered. That is,

$$\lambda(u' \rightarrow v') = \Lambda - \sum_{i=0}^{c-1} \pm y_i. \tag{I–3}$$

Since the summation in Equation I–3 is a linear operation, complementary code symbols along the branches translate into complementary BMs that are used in the same butterfly. Here, $\overline{\lambda}(\,)$ denotes the complementary BM to $\lambda(\,)$. As an example, consider a rate 1/2 code and $c_0 = c_1 = +1$. Then,

$$\lambda(u' \rightarrow v') = \Lambda - y_0 - y_1. \tag{I–4}$$

The complementary code symbols are $\overline{c}_0 = \overline{c}_1 = -1$ and, therefore, the complementary BM becomes

$$\overline{\lambda}(u' \rightarrow v') = \Lambda + y_0 + y_1. \tag{I–5}$$

Substituting Equation I–4 into Equation I–5, we can write

$$\begin{aligned} \overline{\lambda}(u' \rightarrow v') &= 2\,\Lambda - \lambda(u' \rightarrow v') \\ &= \lambda(u' \rightarrow v') + 2\,[\Lambda - \lambda(u' \rightarrow v')] \\ &= \lambda(u' \rightarrow v') + \lambda^*(u' \rightarrow v'), \end{aligned} \tag{I–6}$$

where

$$\lambda^*(u' \rightarrow v') \equiv 2\,[\Lambda - \lambda(u' \rightarrow v')], \tag{I–7}$$

which is a signed number, is called a modified BM. Although here derived for $c = 2$, Equation I–6 holds for any c.

The quantization scheme of the soft-output demodulator, together with Λ, determines how the complementariness of code symbols translates into complementary BMs. We assume demodulator outputs Q that are equidistantly spaced and quantized with q bits, that is, in 2's complement integer notation $Q \in [-2^{q-1}, 2^{q-1} - 1]$. According to Equation I–3 the minimum and maximum BMs based on quantized channel values $[y_i] = Q + \delta$ are

$$\min \lambda(u' \rightarrow v') = \Lambda + c\,(\min Q + \delta)$$

and

$$\max \lambda(u' \rightarrow v') = \Lambda + c\,(\max Q + \delta),$$

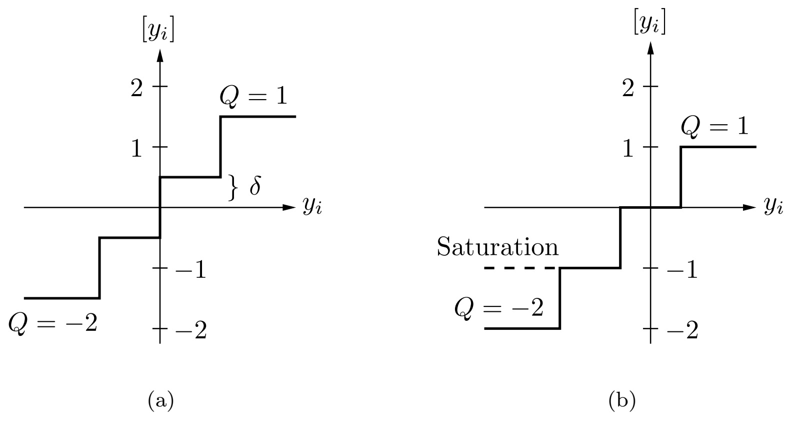


Figure I–2: Truncation with offset δ is shown in (a), rounding in (b). Both quantizers have $q = 2$ bits.

where δ depends on the used quantization scheme, see Figure I–2. In the following, two commonly used schemes are introduced.

The scheme from Figure I–2(a) interprets a 2's complement number in a symmetric way. The demodulator output $[y_i]$ is actually the 2's complement number Q plus $\delta = 0.5$ to achieve a decision threshold at 0. In other words, this corresponds to truncation plus 0.5. Choosing

$$\Lambda = -c\,(\min Q + 0.5) = \frac{c}{2}\,(2^q - 1)$$

results in nonnegative BMs in the range of $[0, c\,(2^q - 1)]$ and unsigned arithmetic can be used. Thus,

$$\overline{\lambda}(u' \to v') = c\,(2^q - 1) - \lambda(u' \to v').$$

Rounding as in Figure I–2(b) takes a 2's complement value without any offset ($\delta = 0$). Choose $\Lambda = 0$ and hence

$$\overline{\lambda}(u' \to v') = -\lambda(u' \to v').$$

By making the quantizer output range symmetrical, that is, limiting $[y_i]$ to $[-2^{q-1} + 1, 2^{q-1} - 1]$, the BM range covers $[-\frac{c}{2}(2^q - 2), \frac{c}{2}(2^q - 2)]$, which can be represented with $\lfloor \log_2 \frac{c}{2} \rfloor + q + 1$ bits as in the truncation case.

3 Simplified Architectures

In this section the two simplifications are presented. Since both methods simply reformulate the underlying ACS operation, there is no BER degradation compared to the original algorithm.

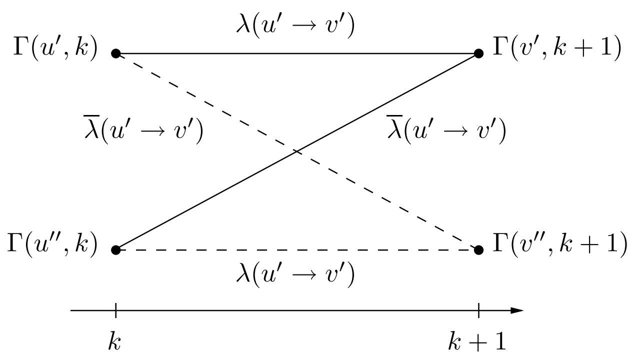


Figure I–3: A radix-2 butterfly. For good rate $1/c$ codes, there are two (complementary) BMs that belong to such a butterfly.

From Equation 2.13, the ACS operation for rate $1/c$ codes is derived by noting that there are two merging branches, as shown in Figure 2.13. A butterfly according to Figure 2.6(a) consists of two such ACS operations. Taking into account the complementariness of the considered codes, the corresponding butterfly is depicted in Figure I–3. To update the SMs $\Gamma(\,)$ at time $k + 1$, we can write

$$\Gamma(v', k+1) = \min\{\Gamma(u', k) + \lambda(u' \to v'), \Gamma(u'', k) + \overline{\lambda}(u' \to v')\} \tag{I–8}$$

and

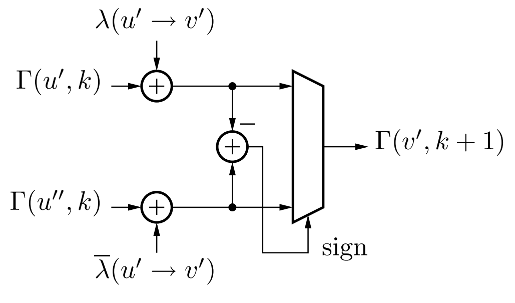
$$\Gamma(v'', k+1) = \min\{\Gamma(u'', k) + \lambda(u' \to v'), \Gamma(u', k) + \overline{\lambda}(u' \to v')\}. \tag{I–9}$$

The processing units for Equation I–8 and Equation I–9 are shown in Figure I–4. Substituting Equation I–6 into these two equations, and taking $\lambda(u' \to v')$ out of the comparison, we get

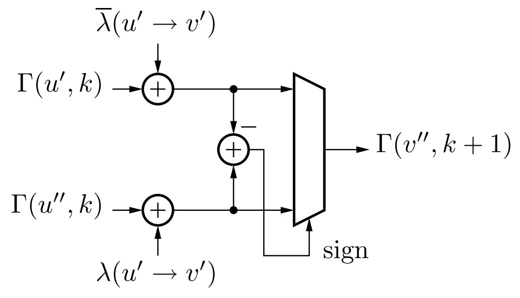
$$\begin{aligned}\Gamma(v', k+1) = {} & \lambda(u' \to v') + \\ & \min\{\Gamma(u', k), \Gamma(u'', k) + \lambda^*(u' \to v')\}\end{aligned} \tag{I–10}$$

and

$$\begin{aligned}\Gamma(v'', k+1) = {} & \lambda(u' \to v') + \\ & \min\{\Gamma(u'', k), \Gamma(u', k) + \lambda^*(u' \to v')\}.\end{aligned} \tag{I–11}$$



(a)



(b)

Figure I–4: ACS units for a radix-2 butterfly. The unit in (a) belongs to the solid branches in Figure I–3 and the one in (b) to the dashed branches.

3.1 Offset Approach

For convenience, we introduce $\tilde{\Gamma}(\,)$ as the new outcome of the min operation; thus, Equation I–10 and Equation I–11 become

$$\Gamma(v', k+1) = \lambda(u' \to v') + \tilde{\Gamma}(v', k+1)$$

and

$$\Gamma(v'', k+1) = \lambda(u' \to v') + \tilde{\Gamma}(v'', k+1).$$

Compared to Equation I–8 and Equation I–9 there is one addition less needed to determine the outcome of each $\tilde{\Gamma}(\,)$. In order to retain the numerical relation between interconnected SMs in a trellis with different $\lambda(u' \to v')$, this term has to be added after having determined $\tilde{\Gamma}(\,)$; see Figure I–5. So far, the number of additions to carry out a complete ACS operation has not been lowered.

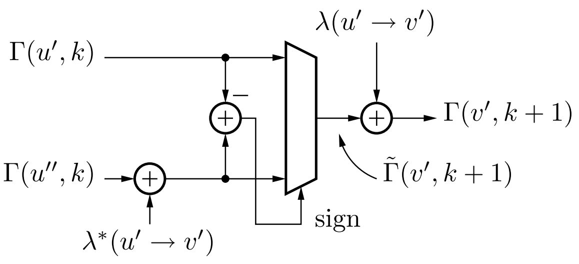


Figure I–5: Transformed ACS unit derived from Figure I–4(a). Both units have the same number of adders but the transformed one needs only two adders to determine the outcome of the comparison $\tilde{\Gamma}(v', k+1)$.

There are 2^c different combinations for the c code (or transmission) symbols along a trellis branch. According to the complementary property, there are two BMs per butterfly, and one can be expressed by the other. Since one of these BMs becomes the update term $\lambda(u' \to v')$ of the respective butterfly, there exist 2^{c-1} distinct update terms. The total number of additions to update $\tilde{\Gamma}(\,)$ to achieve $\Gamma(\,)$ equals the number of ACS operations 2^m. By subtracting any one distinct update term from the others,

$$\max\{2, 2^{m-(c-1)}\} \tag{I–12}$$

updates (additions) are removed. The $a = 2^{\min\{c,m\}-1} - 1$ new offset terms $\Delta\lambda_j$ for $j = 1, \ldots, a$ yield by the subtractions are generated in the BM unit as to

$$\Delta\lambda_j = \lambda_j(u' \to v') - \lambda_0(u' \to v'), \tag{I–13}$$

where by definition $\Delta\lambda_0 = 0$.

Equation I–12 is maximized for $c = 2$, which also gives only one offset term, namely $\Delta\lambda_1$, in Equation I–13. Hence, the offset approach is most beneficial for rate 1/2 codes. This rate plays the largest role in today's communication systems since it is a good compromise between achievable coding gain, bandwidth efficiency, and implementation complexity. In practice, high-rate convolutional codes are obtained by puncturing a rate 1/2 code.

The BM $\lambda(u' \to v')$ can take four different values for a rate 1/2 code, namely $\lambda[c_0\, c_1]$ for every possible combination of transmission symbols c_i. The complementary metrics to $\lambda[+1+1]$ and $\lambda[-1+1]$ that are needed in a conventional

ACS unit are $\lambda[-1-1]$ and $\lambda[+1-1]$, respectively. Note that only two BMs are needed since the other two can be calculated according to Equation I–6, that is, $\lambda[-1-1]$ is expressed in terms of $\lambda[+1+1]$ and $\lambda[+1-1]$ by $\lambda[-1+1]$. The term $\lambda(u' \to v')$ in Figure I–5 to be added in an ACS unit is therefore either $\lambda[+1+1]$ or $\lambda[-1+1]$. However, one can subtract either term from all SMs and in that case half the ACS units do not need this correction, that is, $\Gamma(\,) = \tilde{\Gamma}(\,)$.

From the preceding considerations the hardware savings become apparent by looking at an example, an ACS unit setup for decoding a (7,5) code in Figure I–6. In this figure, the SM corrections in the two ACS units in (a) become obsolete since $\lambda[+1+1]$ is subtracted from all updated SMs. Using Equation I–3 and Equation I–13, the offset term to be used in the two ACS units (b) becomes

$$\Delta\lambda_1 = \lambda[-1+1] - \lambda[+1+1] = 2\,y_0.$$

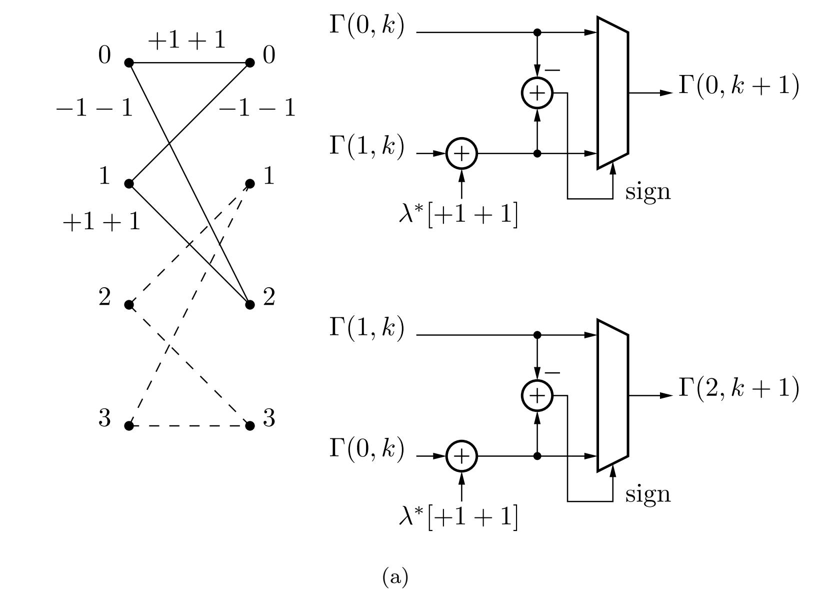
In the following, assume the quantizer from Figure I–2(a) and that the BMs $\lambda[\,]$ are nonnegative. If there are 2^c distinct code sequences, a conventional BM unit requires $\sum_{i=2}^{c} 2^i$ additions and c negations to calculate 2^c BMs. Hence, for a rate 1/2 code we need four adders and two negations to calculate four BMs as in Figure I–7(a). The proposed BM unit shown in Figure I–7(b) requires only two additions, one negation of a channel value, and negations to calculate two modified BMs $\lambda^*[\,]$ and the required offset term $\Delta\lambda_1$. Note that a bit-shift operation comes at negligible cost in a hardware implementation.

To conclude, for rate 1/2 codes, Table I–1 shows the number of additions for a BM/ACS unit setup and encoder memory m. The proposed scheme halves the additions in the BM unit and reduces the number of additions for the ACS units by 17%.

Note that the critical path in half the ACS units in Figure I–6 is increased by the delay of an addition. If speed is an issue, this problem is solved by delaying the correction into the next computation cycle and the original critical path of the ACS unit is maintained. In this case, the additions for the correction that are after the multiplexers in Figure I–6(b) move into the comparison path of different ACS units instead; see the retimed ACS units in Figure I–8.

Table I–1: Number of additions for BM/ACS unit setup of a rate 1/2 code.

	ACS	BM
conventional	$3 \cdot 2^m$	4
offset	$2.5 \cdot 2^m$	2



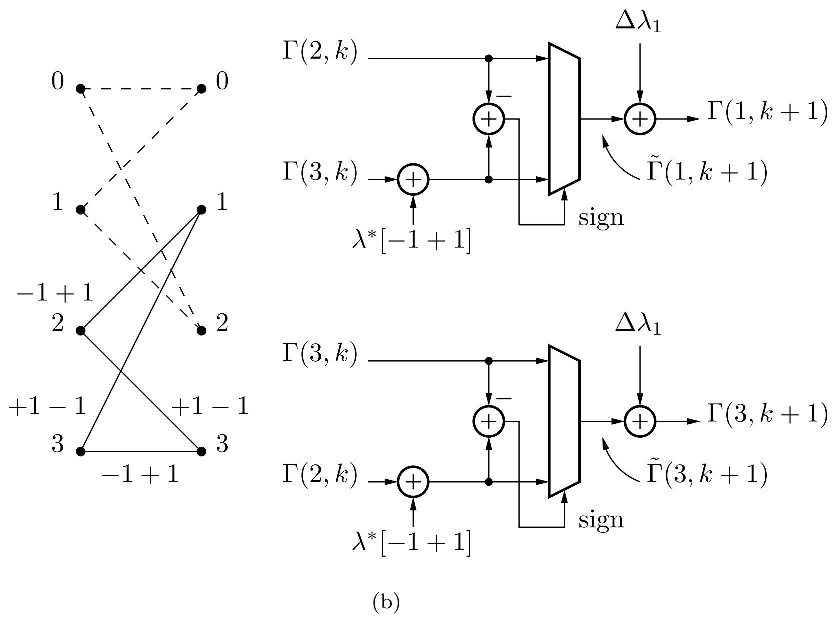
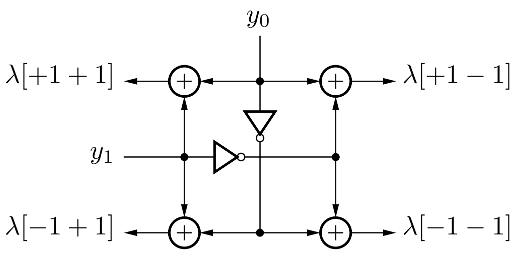
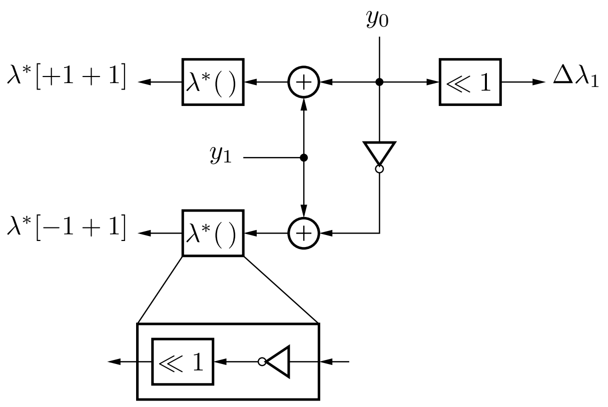


Figure I–6: ACS setup for a (7,5) code for the offset approach. The respective trellis nodes are shown on either side together with the expected transmission symbol labels $[c_0\, c_1]$ along the branches. $\lambda[+1+1]$ is subtracted from all updated SMs.

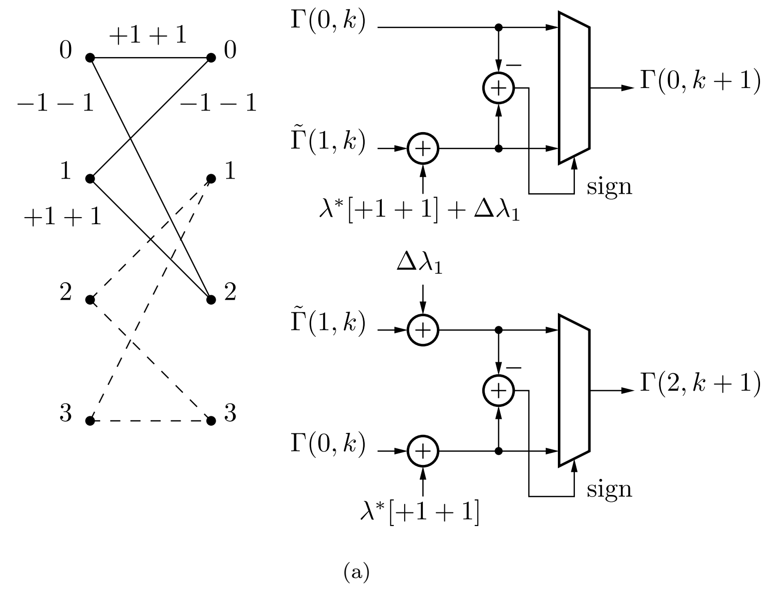


(a)



(b)

Figure I–7: Conventional (a) and modified (b) BM unit for a rate 1/2 code. $\ll 1$ denotes a left shift by one bit.



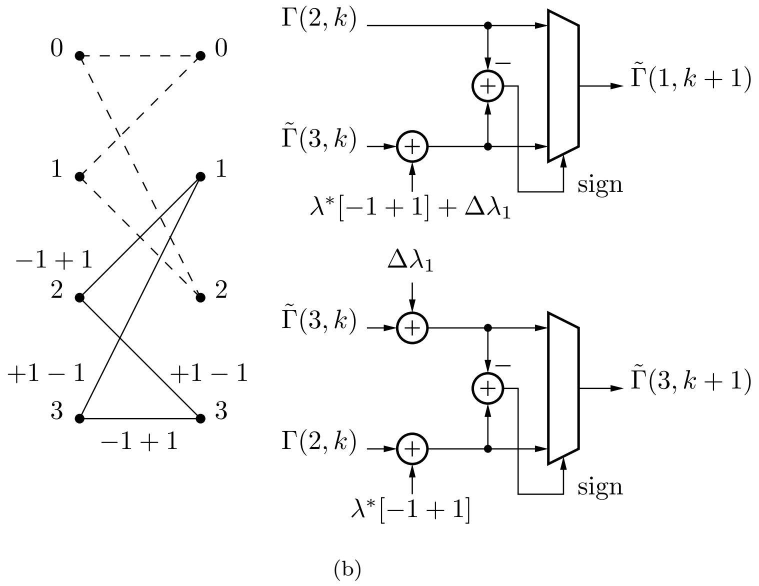


Figure I–8: Retimed ACS setup for a (7,5) code for the offset approach. These ACS units preserve the critical path of the conventional ACS units from Figure I–4. Note that $\Delta\lambda_1$ is delayed by one clock cycle in the BM unit.

Besides storing $\Delta\lambda_1$ in the BM unit, two new correction terms $\lambda^*[+1+1]+\Delta\lambda_1$ and $\lambda^*[-1+1]+\Delta\lambda_1$ are needed for this architecture. These additions are not carried out in the ACS units since this again would increase the critical path. They are instead precalculated in the BM unit; the complexity is moved from the ACS units to the BM unit which is instantiated only once instead of 2^m times. Thus, the BM unit for the retimed setup is the one from Figure I–7(b) appended with two additions and a register. If the extra delay introduced by these additions cannot be tolerated, the datapath can always be pipelined since it is purely feedforward.

3.2 Differential Trellis Decoding

Following [48], DTD can be described as follows. By extracting $\Gamma(u'',k)$ and $\Gamma(u',k)$ from the second argument in the comparison in Equation I–10 and Equation I–11 respectively, we get

$$\begin{aligned}\Gamma(v',k+1) = \lambda(u' \to v') + \Gamma(u'',k) + \\ \min\{\Gamma(u',k) - \Gamma(u'',k), \lambda^*(u' \to v')\}\end{aligned} \tag{I–14}$$

and

$$\begin{aligned}\Gamma(v'',k+1) = \lambda(u' \to v') + \Gamma(u',k) + \\ \min\{-[\Gamma(u',k) - \Gamma(u'',k)], \lambda^*(u' \to v')\}.\end{aligned} \tag{I–15}$$

The comparisons share the common terms $\Delta\Gamma = \Gamma(u',k) - \Gamma(u'',k)$ and $\lambda^*(u' \to v')$ and are therefore carried out jointly by evaluating the differences and signs of these terms.

Figure I–9 depicts the principal branch selection process based on these common terms. This process is shaded in gray in Figure I–10, which shows a computational unit developed by us for the DTD method from the preceding considerations. This unit carries out the two ACS operations for the butterfly in Figure I–3. Based on the binary result x of the absolute comparison $|\Delta\Gamma| > |\lambda^*(u' \to v')|$ (Abs Cmp) and the signs of these two comparison terms, the sign mapper (Sgn Map) decides which of the two SMs are chosen as survivors. Its truth table is found in Table I–2. The resulting control signals M_0 and M_1 are also used to steer which update arguments are to be chosen.

Considering a hardware implementation, 2's complement arithmetic is often used for a Viterbi decoder. In this case, normalization of the otherwise unbounded wordlength increase of SMs in the trellis datapath can be done on-the-fly using the modulo normalization technique published by Hekstra [56]. Here, we restate the result for rate $1/c$ codes. The VA bounds the maximum dynamic range between two SMs to

$$|\Gamma(u'',k) - \Gamma(u',k)| \le m \cdot \lambda_{\max}, \tag{I–16}$$

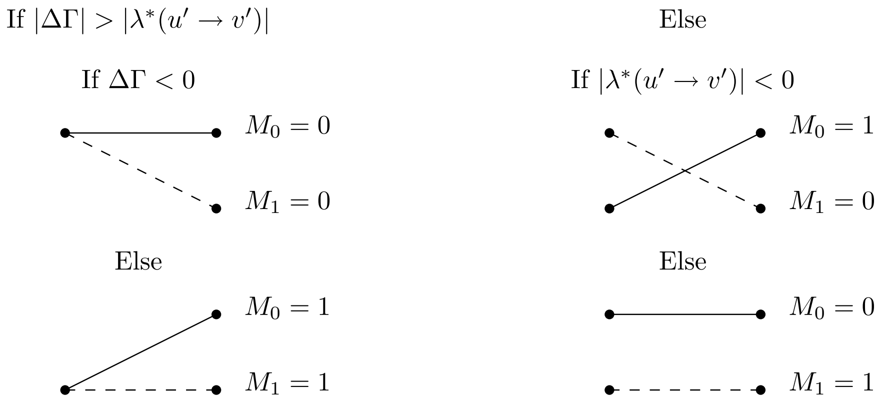


Figure I–9: Branch selections for the butterfly from Figure I–3. Figure adapted from [48].

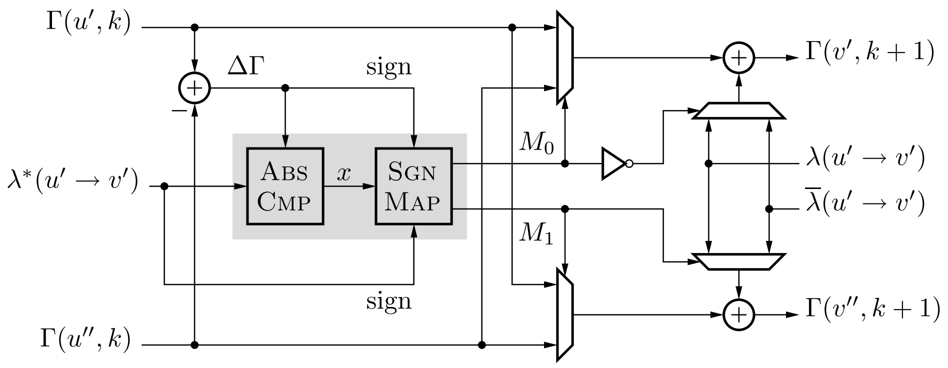


Figure I–10: A computational unit for a butterfly using the DTD method.

Table I–2: Truth table of SGN MAP.

x	sign $\lambda^*(u' \to v')$	sign $\Delta\Gamma$	M_0	M_1
0	0	—	0	1
0	1	—	1	0
1	—	0	1	1
1	—	1	0	0

where $\lambda_{\max}$ is the maximum absolute value of a BM. In the modulo approach, the comparison of two cumulative SMs in an ACS operation is done with a subtraction[†]

$$\Gamma(u'', k) + \lambda(u'' \to v') - [\Gamma(u', k) + \lambda(u' \to v')]. \tag{I–17}$$

If the result of this subtraction lies in the range of a W-bit 2's complement integer, that is, in $[-2^{W-1}, 2^{W-1} - 1]$, a modulo-2^W operation on Equation I–17 does not change the result. Substitute Equation I–16 into Equation I–17, and the range requirement is expressed as

$$(m + 2) \cdot \lambda_{\max} \leq 2^{W-1} - 1.$$

Modulo arithmetic is simply implemented by ignoring the SM overflow if the wordlength is chosen to represent twice the dynamic range of the cumulated SMs before the compare stage. The wordlength of the SMs is thus

$$W = \lceil \log_2\{(m + 2) \cdot \lambda_{\max}\} \rceil + 1 \text{ bits}. \tag{I–18}$$

Note that Equation I–18 assumes positive and negative BMs. Otherwise, the factor in front of $\lambda_{\max}$ is $(m + 1)$.

In Fossorier's paper [48], the joint comparison for Equation I–14 and Equation I–15 is done according to

$$|\Delta\Gamma| > |\lambda^*(u' \to v')|. \tag{I–19}$$

However, a comparison based on absolute values is not efficient in 2's complement arithmetic. To derive an operand's absolute value basically requires half-additions in hardware. Although Equation I–19 would be somewhat simpler to implement in sign-magnitude representation, SM normalization becomes more complex [88], adding overhead to the design. Therefore, we use a conditional addition/subtraction based on 2's complement arithmetic to carry out Equation I–19. The chosen operation depends on the sign bits of the operands $A = \lambda^*(u' \to v')$ and $B = \Delta\Gamma$. If both have the same sign, one is subtracted from the other $(A - B)$. An exclusive-or operation on this difference with the sign of the minuend (A) yields the binary result x, which is 0 for $|B| \leq |A|$. Otherwise, both numbers are added and again the sign of A determines the outcome of Equation I–19. The corresponding unit is shown in Figure I–11.

Retiming as in the offset approach is not applicable in DTD since this method explicitly avoids the evaluation of cumulative SMs and rather works

[†]Here, we do not use the complementary property for the BMs.

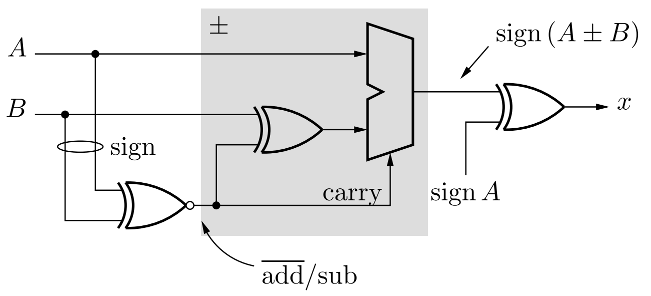


Figure I–11: The Abs Cmp unit for comparison of absolute values (Equation I–19). The basic building block is a conditional addition/subtraction unit (shaded in gray). For DTD, $A = \lambda^*(u' \to v')$ and $B = \Delta\Gamma$.

directly on the numerical relationship between state and branch metrics. Therefore, delaying the SM updates into the next clock cycle does not decrease the critical path, which basically consists of three additions and additional logic by Sgn Map. Compared to the retimed offset approach, this is an increase by at least one addition.

The reduction in arithmetic complexity compared to the original architecture from Figure I–4 cannot be expressed as straightforwardly as in the offset approach since modified operations, for example, conditional additions and subtractions, are introduced. Therefore, synthesis results of the two simplifications are compared in Section 5.

4 Implementation and Synthesis Results

In the next case study, computational blocks for the VA using the offset approach are implemented. Best feedforward rate 1/2 convolutional codes up to memory $m = 7$ are considered. The output of an ACS unit is the decision bit for the surviving path of the respective state and the updated SM, see Figure 2.12. Survivor path processing is neglected since this part of the decoder does not differ between the conventional and improved architectures. The BM and ACS units are described in a VHDL model at register-transfer level based on generic parameters. The well-known SM normalization techniques are still valid since differences among the SMs remain the same. Nonnegative BMs together with Equation I–16 and Equation I–18 yield the following expression for the wordlength of the SMs:

$$\lceil \log_2\{(m+1) \cdot 2\,(2^q - 1)\} \rceil + 1 \ \text{bits.} \tag{I–20}$$

The comparison in the ACS unit is implemented with the modified comparison rule [88]. Channel symbol wordlength is $q = 3$ since this gives small degradation in decoding performance compared to infinite precision as stated in [57].

We used a design kit from Virtual Silicon for the United Microelectronics Company (UMC) 0.13 μm digital CMOS process. Power figures were obtained by Synopsys Power Compiler using toggle information from a gate level simulation run with state- and path-dependent cell information, and random input stimuli. Both dynamic (switching and short-circuit power) and leakage power are included in the results. However, the contribution from leakage power is negligible in this study. At this design stage, it is assumed that the contribution from clock tree and interconnection, which is relevant for absolute area and power numbers, is the same in both architectures since we are only interested in the relative savings between architectures.

The improved versions are compared to their respective conventional setup with regard to their application area. For applications with relaxed timing requirements, area and power comparisons are done for the architecture in Figure I–6 together with the respective BM units. Synthesis tests showed that the area–delay product curve is flat down to a delay of about 3.5 ns, which is set as a constraint for the critical path. The power simulation is carried out at a clock frequency of $f_{\text{clk}} = 250$ MHz. For the retimed architecture of Figure I–8 this delay reaches further down to about 2 ns. Here, we only synthesized the ACS units in order to investigate the impact of the saved adder in every unit. For the power simulation $f_{\text{clk}} = 400$ MHz is assumed. In both cases, $V_{\text{dd}} = 1.2$ V.

Table I–3 lists the synthesis results of the cell area for a conventional ACS setup with and without the BM unit from Figure I–7(a). In comparison with it, Figure I–12 shows the possible savings in cell area and power consumption when the improved architecture from Figure I–6 is employed. As mentioned earlier, the arithmetic complexity is reduced by 17%, which is true for $m = 2$. However, with increasing m the percent savings decrease since both area and power overhead introduced by the registers gets bigger. At $m = 4$, the SM

Table I–3: Cell area in μm^2 for a conventional BM/ACS setup. Timing constraints for row BM & ACS (non-retimed) and ACS (retimed) are 3.5 ns and 2 ns, respectively.

m	2	3	4	5	6	7
BM & ACS	3697	6876	15120	29723	58930	115974
ACS	2923	5846	13328	26657	53315	106601

register wordlength is increased by one bit; refer to Equation I–20. Thereafter the combinational power savings catch up with this initial penalty and again reach 12% at $m = 6$.

Figure I–12 also shows the comparison results when the retimed setup from Figure I–8 is used. Note that compared to the non-retimed setup, the power figures were obtained at a higher clock frequency due to the shorter critical path. The adders incorporating the correction terms are one bit wider than the ones in the conventional architecture. Again, the improved setup saves both area and power by 7% and 10%, respectively.

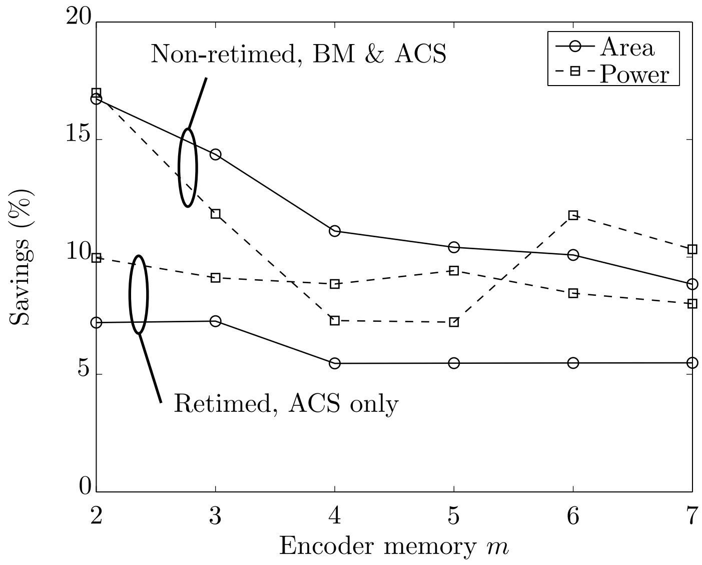


Figure I–12: Area and power comparison between conventional setups from Table I–3 and the improved setups from Figure I–6 and I–8.

If speed requirements allow use of the computational kernel in a time-multiplexed fashion, the savings increase compared to a parallel implementation; for example, for $m = 6$, there are achievable savings of 10% in cell area; however, a time-multiplexed architecture using a $m = 2$ kernel could gain an extra 7%.

5 Comparison of Hardware Efficiency

In this section, a comparison between the offset approach and DTD is carried out based on hardware savings when 2's complement arithmetic is used. Again, the setup for the case study is the one described in Section 4. Tests with the different architectures showed that the area–delay product curve is flat down to a delay of about 4 ns in case of DTD, which is set as a constraint to the critical path. BM units are not included in the synthesis since their contribution is assumed to be approximately equal for a certain m. Considering throughput, a pipeline is assumed between BM and ACS units and thus the critical path is determined by the ACS units.

Figure I–13 shows the synthesis results for decoding blocks for different encoder memories m. It is seen that the offset approaches (both retimed and non-retimed) are favorable across the whole range of m. On a percent basis, these approaches are 24% ($m = 2$) to 30% ($m = 6$) smaller than the DTD architecture in this case study.

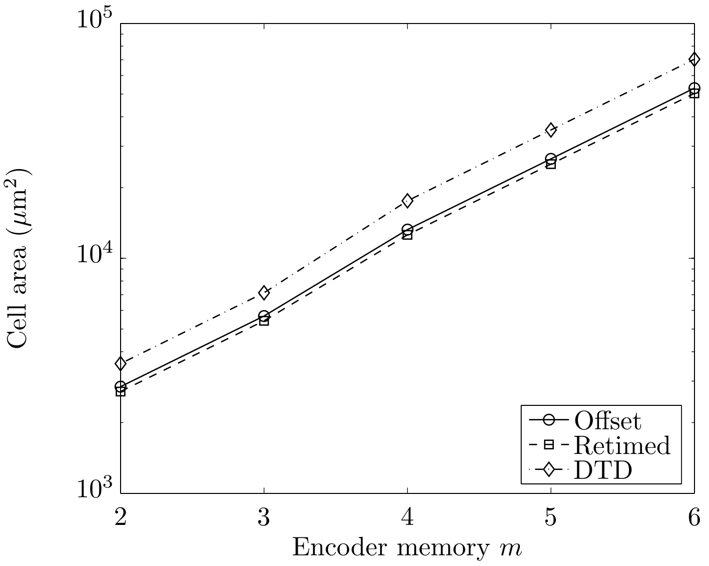


Figure I–13: Cell area versus encoder memory m for the simplified decoding blocks.

Although the number of arithmetic operations appears comparable at first glance, the sizes of the synthesized architectures differ remarkably. Generally, conditional addition/subtraction adds an overhead compared to a conventional addition, which is determined by the implementation style of the arithmetic unit. Considering the wordlengths in DTD, the initial subtraction that yields $\Delta\Gamma$ requires sign extension, which increases the wordlength for the conditional addition/subtraction in Abs Cmp by one bit. Apparently, the increased wordlength and the selection logic turn out to be the drawback of this approach. On the other hand, in the offset approach the subtraction of the cumulative metrics to determine the surviving metric is usually done with an unsigned comparison, see [88], which lowers the computational cost by a full-adder stage.

The critical path in the offset approach, which is characterized by three additions and a multiplexer, is shorter than the one in the DTD method, where it is determined by the delay through the initial subtraction, Abs Cmp, Sgn Map, a multiplexer, and the final addition. This roughly translates to three additions, a multiplexer, and the logic delays inside Sgn Map. Furthermore, the offset approach can be retimed as indicated in Section 3.1.

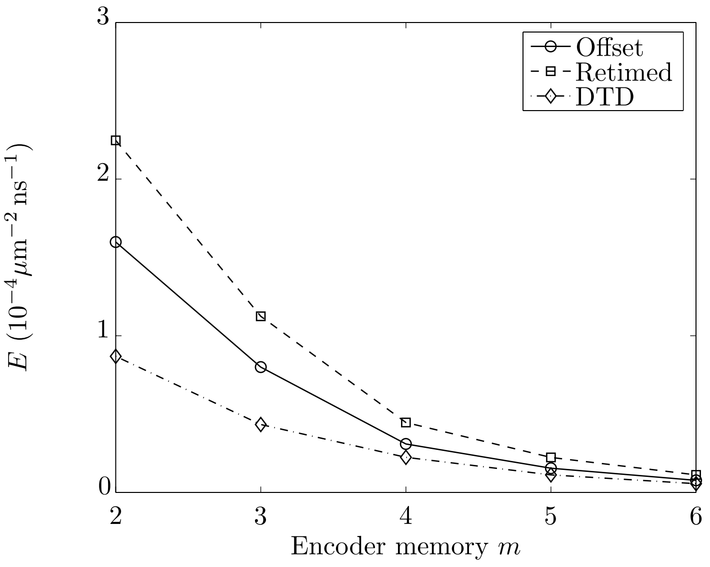


Figure I–14: Hardware efficiency versus encoder memory m for the simplified decoding blocks.

Let the hardware efficiency E be expressed as

$$E = \frac{1}{A \cdot T},$$

where A stands for area and T is processing time per sample, that is, $1/T$ denotes the achievable throughput. Based on this definition, Figure I–14 compares the two simplifications in terms of E, where a higher value denotes a better design for a certain m. Again, it is seen that the offset approaches are preferable throughout the considered design space. Note that with increasing m, the efficiency as defined here decreases since the hardware is duplicated, whereas T stays approximately the same. Although less obvious from the figure, the percent loss in efficiency for DTD compared to the other two methods is slightly larger for $m = 2, 3$. This is due to the larger deviation of the critical paths in these cases.

6 Conclusion

We showed that the implementation of BM and ACS units in trellis-based decoding architectures can be simplified for a certain class of convolutional codes. By making use of the codes' complementary symbol property, both area requirement and power consumption of trellis computational blocks are reduced. The offset approach applied to a rate 1/2 code saves one adder in half the ACS units compared to a conventional implementation. Furthermore, only two BMs have to be calculated instead of four. In a case study, area savings vary between 17% and 9% and power savings from 17% to 7% are reported in a 0.13 μm digital CMOS process.

The offset approach and a competing simplification named DTD, which also uses the code symbols' complementariness, are analyzed in terms of their hardware efficiency. Although promising at first glance, the drawback of DTD turns out to be the $|\cdot|$ calculation. Also, the "sign checks" to determine the surviving SMs degrade the performance compared to the offset approach, which solely relies on common arithmetic operations. Hence, for rate 1/2 codes the offset approach is preferable to the DTD method for hardware implementation.

Part II

Hybrid Survivor Path Architectures

Abstract

A new class of hybrid VLSI architectures for survivor path processing to be used in Viterbi decoders is proposed. The architecture combines the benefits of register-exchange and trace-forward algorithms, that is, low memory requirement and latency versus implementation efficiency. Based on a structural comparison, it becomes evident that the architecture can be efficiently applied to codes with a larger number of states where trace-back-based architectures, which increase latency, are usually dominant.

Based on: M. Kamuf, V. Öwall, and J. B. Anderson, "Survivor path processing in Viterbi decoders using register-exchange and trace-forward," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, publication scheduled for June 2007.

1 Introduction

The VA is a maximum-likelihood algorithm that can be applied to decoding of convolutional codes. In this part, we consider convolutional codes of rate $1/c$ and high-rate punctured codes that are derived from them. Their trellises have $N = 2^m$ states, where m is the encoder memory.

A Viterbi decoder typically consists of three building blocks, as in Figure 2.12. In this setup, the ACS units (ACSUs) inside the trellis unit and the survivor path unit (SPU) are known to be critical parts for a hardware implementation. Whereas the feedback loop of the ACS operation often determines the throughput of the decoder, the algorithm used for the SPU affects the overall storage requirement and latency, two important aspects in today's communication systems. Recall from Section 2.2.1 that SPU algorithms rely on the fact that the survivor paths are expected to have merged with sufficiently high probability after a certain depth L.

Traditional approaches for the SPU are register-exchange (RE) and trace-back (TB) [41] algorithms, and they are discussed in Section 2. After a brief review of existing hybrid SPU architectures in Section 3, a new hybrid approach based on RE and trace-forward (TF) [16] is proposed in Section 4. Storage requirement and latency can be traded for implementation complexity. Therefore, this architecture can be applied to a larger number of states, which is justified by a comparison to existing hybrid approaches in Section 5.

2 Basic Algorithms

In this section, two basic algorithms for the SPU are presented. They cover a wide range of applications. RE has the lowest memory requirement (NL bits) and latency (L) among all SPU algorithms. As will be seen, it is limited to a rather small number of states. TB is considered applicable to an almost arbitrary number of states at the cost of an increase in both storage and latency.

For clarity, we want to stress that an information bit enters the encoder and causes a state transition, and a decision bit is put out from an ACSU upon decoding. The latter thus indicates a surviving branch in the trellis diagram. Note that information and decision bits for a state are not the same for feedforward codes but coincide for feedback codes.

2.1 Register-exchange

This method is the most straightforward way of survivor path processing since the trellis structure is directly incorporated in the algorithm. Every trellis state is linked to a register that contains the survivor path leading to that state.

The entire information sequences of the survivor paths are then continuously updated based on the decisions provided by the ACSUs.

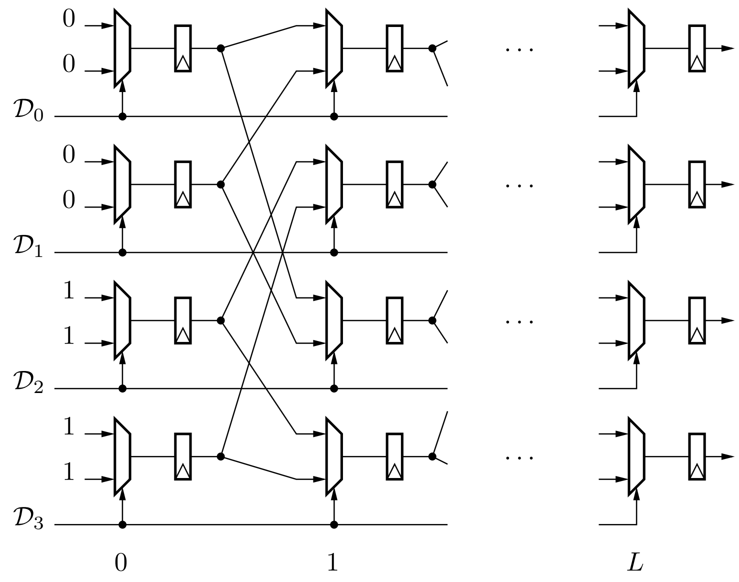


Figure II–1: An REU for a 4-state trellis. $\mathcal{D}_X$ denotes the decision bit for trellis state X. Initialization values at stage 0 apply to feedforward codes. In this case, the first $m = 2$ stages become obsolete since the start sequences are always the same due to the given trellis topology. For feedback codes, the decision bits are fed directly to the first register stage.

After L steps, either one reads the decoded bit from a fixed state or one takes a majority decision among all states. This is equivalent to the principle of fixed-state or best-state decoding discussed in Section 2.2.1. The latency of this algorithm is simply L. In a parallel implementation, NL bits are required which must be read and written every cycle. This high access bandwidth makes an implementation in high-density random-access memory (RAM) impractical. Instead, the algorithm is preferably realized by a network of multiplexers and registers that are connected according to the trellis topology; see Figure II–1 for the RE unit (REU). For a larger number of states, though, the low integration density of the multiplexer-register network and the high memory bandwidth of NL bits per cycle become the major drawbacks of this algorithm. It is used, though, in applications where the survivor path has to be updated in one clock cycle, or the overall latency and/or storage requirement is crucial.

2.2 Trace-back

The TB method [30, 41] is a backward processing algorithm and requires the decisions from the ACSUs to be stored in a memory. An L-step backward search through a segment where all survivor paths merge into one delivers the starting state of a decoding segment. For encoders realized in controller form, the principle is as follows. The binary state number at time k is shifted b bits to the left and appended on the right with the b decision bits of that state to yield the originating state at time $k-1$. Once the starting state is established, the final surviving state sequence is reconstructed step-by-step in the described backward fashion. The corresponding information symbols are put out time-reversed and, therefore, a last-in-first-out (LIFO) buffer has to be introduced to reverse the decoded bitstream.

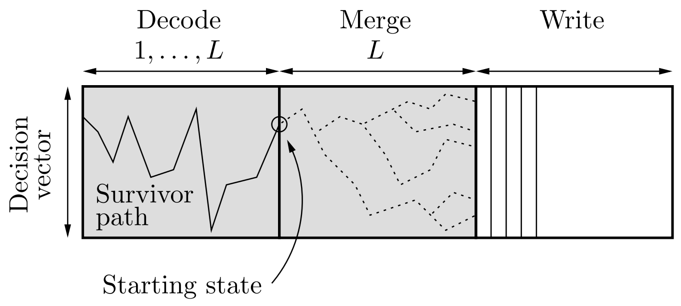


Figure II–2: Organization of decision memory in a TB architecture with two read (in gray) and one write region.

Consider an architecture with two read regions, decode and merge. The memory partitions and their tasks are shown in Figure II–2. Tasks shift circularly among the partitions, that is, the merge segment becomes the decode segment of the next TB cycle, which starts every Lth step. Then write segment becomes merge segment, decode becomes write, and so on. Decode TB and merge TB are performed concurrently, while at the same time, a new decision history is built in the write region. Assuming that decode, merge, and write segments are each of length L (access rates are matched), we find the storage requirement is $3LN$ bits for the decision memory plus L bits for the LIFO, and latency is $4L$.

The advantage compared to RE is the storage of the decisions in a much denser memory, typically RAMs. Only N decision bits are written every cycle, thus the write access bandwidth is greatly reduced. Since TB operations (decode and merge) are started every Lth step, only $N(L + L)/L = 2N$ decision bits have to be read per decoded output bit. It is assumed that only standard memories are used, that is, a read access provides a complete N-bit word, and the $N : 1$ column multiplexer is separated from the memory. Clearly, storage requirement and latency become higher. This method is suitable for applications where more than one clock cycle is available per trellis step.

Generally, for an n-pointer-odd architecture [41], where $n > 1$ is the number of read pointers, the RAM requirement is $NL(2 + 1/(n-1))$ and the LIFO buffer is of size $L/(n-1)$. The overall latency becomes $L(2 + 2/(n-1))$. To reduce the increased memory size and latency, TB is mainly used in conjunction with the TF [16] procedure, which is discussed in Section 3.2.

3 Existing Hybrid Approaches

Several attempts have been made to increase the implementation efficiency of the SPU by combining different algorithms. Two prominent members are discussed in the following since these are the ones our architecture is derived from and competes with.

3.1 Register-exchange and Trace-back

A hybrid architecture combining RE and TB was first published in [75]. The idea was also discovered in [16] and [15], and later generalized in [20] to derive a class of so-called pre-compiled SPUs. This class also includes the TF approach, which in [20] is called ER(ν, L) precompilation. However, except for the TF method, these approaches require specific memories which have to be accessible row- and columnwise, thus increasing the implementation complexity.

To reduce both latency and read accesses during the merge phase, the hybrid architecture, denoted RE/TB, carries out TB operations over κ bits at a time instead of one. Let the decoding depth L be divided into blocks of size κ, that is, $L = \ell\kappa$, ℓ an integer. An REU of size $N\kappa$ is used to continuously build segments of the survivor path for each state. These segments are then stored every step as N-bit *column* vectors in a RAM. A κ-bit segment of a *row* vector for a certain state contains the starting state of its survivor path κ bits earlier; that is, this so-called block TB covers κ bits per RAM read access, instead of one bit as in the traditional TB method. Hence, the number of TB operations to find the starting state of a decoding segment is lowered from L to ℓ. Since the survivors are preprocessed in the REU, the final decoding can be carried out in one step. Note, however, that the RAM has to be accessible both row- and columnwise, which requires a more complex specialized memory implementation. The overall storage requirement, listed by different implementation complexity, becomes

$$\underbrace{N\kappa}_{\text{REU}} \text{ and } \underbrace{N(L+(2p-1)\kappa)}_{\text{RAM}}, \tag{II–1}$$

where p is the number of κ-bit segments that are finally decoded once a starting state is found [75].

3.2 Trace-forward and Trace-back

In agreement with its first appearance in the literature [16], we adopt the name TF for the following procedure. An algebraic generalization is found in [39] and real hardware effects of this approach have been recently published in [50, 55].

TF is a forward-processing algorithm that estimates the starting state for a decode TB on-the-fly such that TB operations during a merge phase, which do not contribute to the actual decoding, can be omitted. The TF method is applied to lower both storage requirement and latency in TB-based architectures.

Every survivor path at time $i+\Delta$, i an integer, is connected to some state at time i, called a tail state. According to the considerations about the decoding depth L_c in Section 2.2.1, all survivor paths should stem from the same state for $\Delta > L_c$, that is, their tail states should be identical. TF is basically a selection operation, similar to the RE algorithm without the shift.

Figure II–3 shows the TF unit (TFU) for a 4-state rate $1/c$ convolutional code. At time iL, each m-bit register block is initialized with the state label it represents; that is, current states and tail states are identical, and the survivor paths are of zero length. The decision $\mathcal{D}_X$ for state $X = 0, \ldots, N-1$ selects the tail state of its predecessor state to update the current tail state. At time $(i+1)L$, when all survivor paths should have merged, all registers contain the starting state for the decoding segment at time iL. For illustration, state 0 is chosen to be read from in Figure II–3. Furthermore, it is also indicated in [16] that a TFU can be optimized, where area-efficient ACSU topologies [19] are applied due to the structural equivalence of TFU and ACSU.

The extension to TB architectures with ℓ TFUs that estimate starting states at times L/ℓ to further reduce latency is discussed in [39]. In total the storage requirement is

$$\underbrace{N \times L(1+1/\ell)}_{\text{RAM}}, \underbrace{\ell N m}_{\text{TFU}}, \text{ and } \underbrace{L/\ell}_{\text{LIFO}} \tag{II–2}$$

bits in this approach.

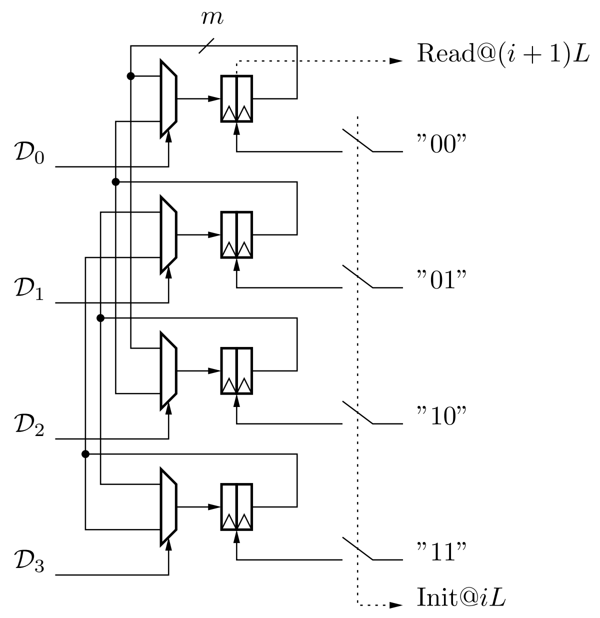


Figure II–3: A TFU for a 4-state trellis, that is, $m = 2$. Figure adapted from [16].

4 New Approach: Register-exchange and Trace-forward

As stated in [39], the starting state of decoding segments can be found by means of so-called multiplier feedback loops, which are equivalent to TFUs. According to this observation, we estimate the starting states of length-κ segments with TFUs in intervals of κ; see Figure II–4. Every κth step, a TFU is initialized, and a total of ℓ TFUs are needed to cover the complete decoding depth L; that is, TFU_j, for $j = 1, \dots, \ell$, is initialized at time $(i-1)L + j\kappa$. Then, at time $iL + j\kappa$, TFU_j contains the estimated starting state of this segment.

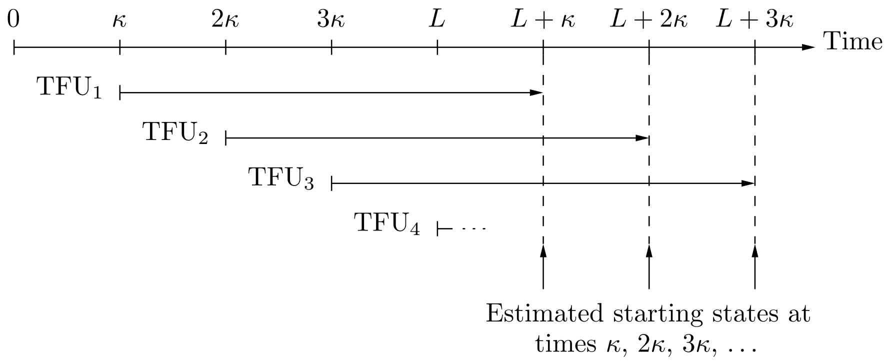


Figure II–4: Picture of TF and decode flow. Note that $\ell = L/\kappa$ is an integer; in this example $\ell = 4$.

Contrary to the previous hybrid approaches, the sequences in the REU are not used for initializing a block TB operation. Instead, these partial survivor sequences are stored every κth step in a RAM with first-in first-out (FIFO) access that can be implemented in a much denser fashion than the original RE network of length L. Once an estimated starting state is established, the respective partial information sequence is directly read from the FIFO. Therefore, time reversal upon decoding as in hybrid TB-based architectures becomes unnecessary and the latency is not increased.

The proposed SPU architecture is depicted in Figure II–5. It consists of three parts: an REU to continuously update the partial survivor sequences for each state, a FIFO to store ℓ sets of N sequences, and a bank of ℓ TFUs that provide the starting states of the length-κ segments.

The following considerations focus on feedforward codes, where an estimated starting state is equivalent to the last m information bits that entered the encoder. In a straightforward implementation, an REU of length κ is needed. We note, though, that for feedforward codes the start of all partial

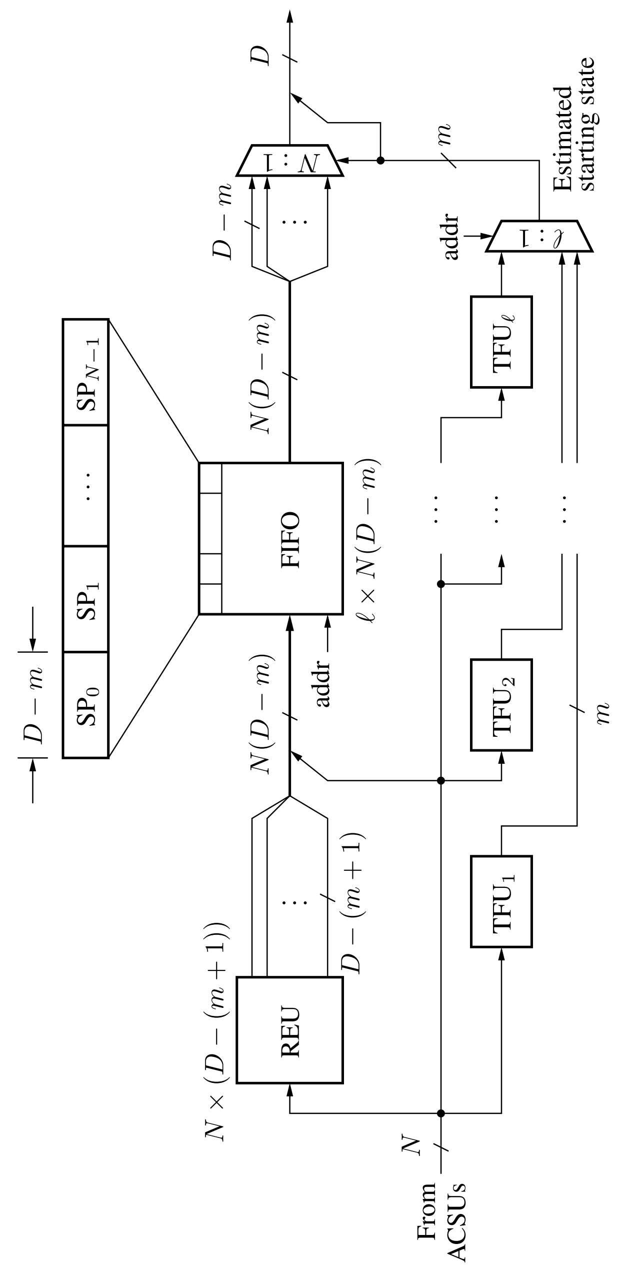


Figure II–5: The proposed hybrid SPU for feedforward codes. Shown above the FIFO is the address pattern for a partial survivor sequence word. The word consists of N sequences SP_X of length $\kappa - m$. The FIFO could be organized for $(\kappa - m)$-bit read accesses, that is, multiplexer $N:1$ is incorporated in such a specialized memory.

survivor sequences is always the same until the trellis is fully extended, that is, after m steps. Thus, only $\kappa - m$ stages are required. Additionally, due to the trellis topology, the last column of decision bits can be directly transferred to the FIFO without storing them in the REU. The REU's length is thus $\kappa - (m + 1)$. Note that there is a constraint on the minimum feasible block length, $\kappa \geq m$.

At times $i\kappa$, for each state, a partial survivor sequence from the REU is stored in the FIFO which is disabled otherwise. The storage scheme of these sequences is shown above the FIFO. Here, SP_X denotes an information stream of length $\kappa - m$ associated with state X. It is seen that the sequence SP_X resides at address X of the memory word. To find the part that is linked to the actual survivor path, the estimated starting state from a TFU is used. For example, at time $L + \kappa$, TFU_1 contains the starting state of the surviving path at time κ and the FIFO subword at this address is selected. These bits represent the information sequence from time 0 to $\kappa - m - 1$. The remaining m bits are included in the estimated starting state since it is identical to the information sequence that entered the encoding shift register. Hence, the overall latency of this approach is $L + \kappa$. For feedback codes, these remaining bits are delivered by the REU, which has to be extended to $\kappa - 1$ stages.

Both REU and TFUs are controlled by the ACSU decisions and run continuously at data rate, whereas the FIFO only runs at $1/\kappa$ times the data rate. The FIFO and the multiplexer $\ell : 1$ both use the same address counter; compared to TB architectures with multiple pointers that require independent address counters, control is much simpler. The estimated starting state selects the subword of length $\kappa - m$ by accessing the $N : 1$ multiplexer. No reversal of the output sequence is required since only forward processing algorithms are used. This preserves low latency.

In summary, the total storage requirement becomes

$$\underbrace{N(\kappa - (m + 1))}_{\text{REU}}, \ \underbrace{\ell \times N(\kappa - m)}_{\text{RAM}}, \text{ and } \underbrace{\ell N m}_{\text{TFU}}. \tag{II–3}$$

The architecture is scalable by varying κ, thus trading storage requirement for implementation complexity. Different L require different partitions between the processing blocks (FIFO, TFU_j, REU) to optimize the implementation. Moreover, an optimal partition depends on the implementation platform. Two special cases can be pointed out for feedforward codes, namely $\kappa = m$ and $\kappa = m + 1$. In both cases, the REU becomes redundant. In the former case, the FIFO also vanishes and the architecture solely consists of TFUs.

5 Comparison and Discussion

Table II–1 lists SPU architectures and their key performance features to allow for comparison between different methods. These comparisons are concerned with the hybrid approaches only; RE and TB algorithms are mentioned for completeness.

Considering the RE/TB method from [75], it is seen that it lags our approach when it comes to RAM requirements. More specifically, given the same latency ($p = 1$) their RAM size is larger by

$$N(\ell m + \kappa) \text{ bits.}$$

Comparing the number of register bits, their REU has $m + 1$ extra stages. However, due to ℓ TFUs in our approach, there are now an additional $N((\ell - 1)m - 1)$ register bits compared to [75]. Note that this is not necessarily the only measure for RE complexity. For example, a TF operation can be executed sequentially in $l \leq m$ steps, which lowers the numbers of multiplexers and interconnections by a factor l. This observation concurs with [39], where the complexity of a multiplier feedback loop (= TFU) is that of one stage of RE since they both operate sequentially on one single decision matrix at a time. Therefore, the complexity of additional ℓ TFUs are comparable to ℓ stages of RE network. That is, the κ RE stages in [75] can be set into relation to $\kappa - (m + 1) + \ell$ equivalent RE stages in our approach. The RE complexity in our architecture is reduced if

$$\kappa > \kappa - (m + 1) + \ell. \tag{II–4}$$

According to Equation 2.14, the decoding depth L_c of a convolutional code is a function of the codes' free distance and the code rate. Choose, for simplicity, $L = \rho(m + 1) \geq L_c$, that is, the required depth is expressed as a multiple ρ of the encoder constraint length $m + 1$. Then, Equation II–4 holds if $\kappa > \rho$.

Apart from that, the REU from [75] has to employ a so-called "zone limit", which distinguishes between RE mode and shift register mode. This increases implementation complexity due to a multiplexer in front of every register in the REU, which thus requires $N\kappa$ additional multiplexers.

Another drawback in [75], which also applies to almost all of the precompiled approaches from [20], is that the RAMs have to be accessible row- and columnwise, which requires a specialized memory implementation and increases complexity. Decision bits are written on a per-state basis (columns) and are read on a per-time-instance basis (rows). This requirement could possibly be dropped by means of a pre-buffering scheme to do the required transposition, which on the other hand increases register complexity.

Table II–1: Key performance features of different SPU architectures. Memory requirement from Equations II–1–II–3 was reformulated to allow for easier comparison. The number of read pointers for TB is $n = \ell + 1$. For RE/TB, the REU requires $N\kappa$ additional multiplexers. Also, RAM in RE/TB must be accessible row- and columnwise and should be organized for κ-bit read accesses.

	REU	RAM	TFU	LIFO	Latency
RE	NL	—	—	—	L
TB	—	$NL(2+1/(n-1))$	—	$L/(n-1)$	$L(2+2/(n-1))$
RE/TB [75]	$N\kappa$	$NL(1+(2p-1)/\ell)$	—	—	$L(1+p/\ell)$
TB/TF [16, 50, 55]	—	$NL(1+1/\ell)$	ℓNm	L/ℓ	$L(1+2/\ell)$
Proposed (RE/TF)	$N(\kappa-(m+1))$	$NL(1-m/\kappa)$	ℓNm	—	$L(1+1/\ell)$

The comparison to TB/TF is carried out on the basis of same latency. From Table II–1 it is seen that there are twice as many pointers ℓ needed in the TB/TF approach compared to our method. Let ℓ_1 and ℓ_2 denote the number of TFUs in our approach and the TB/TF method from Section 3.2, respectively, and hence $\ell_2 = 2\ell_1$. Now the different units can be compared in terms of complexity. To start with the RAM, ℓ_2 partitions are necessary in TB/TF, which increases peripheral overhead, for example, independent address counters. This overhead is not considered in the following calculations. On the contrary, our method needs only one single RAM block, independent of ℓ_1. Simplifying the difference of the RAM sizes, it is seen that TB/TF requires an additional

$$N\left(\ell_1 m + \frac{1}{\ell_2}\right) \text{ bits.}$$

Since there are twice as many TFUs, there are $\ell_1 N m$ additional bits in TB/TF. Since the number of bits is comparable to the ones in the REU, we can directly subtract this overhead from the size of our REU. Furthermore, because of the small size of the LIFO ($\kappa_2 = L/\ell_2$ bits), an implementation with registers is favorable compared to RAM cells. Based on these observations, the register overhead becomes

$$N(\kappa_1 - [1 + m(\ell_1 + 1)]) - \kappa_2 \text{ bits} \tag{II–5}$$

in our approach. Equation II–5 grows with $\mathcal{O}(N)$ and depending on the sign of the expression in the parenthesis this overhead is in or against our favor. If $1 + m(\ell_1 + 1) > \kappa_1$, the register complexity in our approach is smaller. Modifying this inequality gives

$$(\ell_1 - \rho)(m + 1) + m\ell_1^2 > 0,$$

which holds for many parameter choices apart from the obvious $\ell_1 \geq \rho$. It is clear that ℓ and N are critical parameters when comparing implementation efficiency of RE/TF and TB/TF. One should also keep in mind that the complexity of a TFU compared to an REU can be adjusted by means of area-efficient approaches as mentioned in [19].

Generally, the different architectures' feasibility depend on the choice of implementation parameters. That is, a factor sets the cost of register and RAM bits into relation. Such a factor would depend on the number of RAM bits, partitions, RE interconnects, folding of TF operations, and so on.

The proposed architecture is seen as means to lower the RE complexity by employing denser storage cells for the survivor sequences. Thus, the architecture can be applied to codes with larger number of states. At the same time, the desirable high-speed low-latency feature of RE is preserved.

Throughout the preceding considerations we assumed a two-port memory for the FIFO that allows a read-before-write access on the same address, so the old value is present at the output while the new value is written into the chosen memory location. However, if a single-port memory is employed, the read access has to be carried out one cycle prior to the write access to the same address, and hence an additional RAM word is needed to temporarily store the old value. Since the two-port constraint was also assumed in the competing hybrid architectures, the effect of an additional storage word is cancelled out.

6 Conclusion

We presented a new class of hybrid survivor path architecture based on register-exchange and trace-forward concepts. Latency and memory requirement can be traded for implementation complexity. To be specific, the register-exchange complexity is lowered by employing denser storage cells. No partitioning is necessary for this memory, independent of the number of decoding blocks, contrary to combined trace-back and trace-forward architectures. Therefore, our approach can be seen as means to extend the desirable high-speed low-latency feature of pure register-exchange implementations even for a larger number of states. Furthermore, contrary to some other existing hybrid architectures, this new architecture is not bound to a specialized memory implementation and can thus be optimized for different platforms.

Part III

Designing a Flexible Trellis Decoder

Abstract

This part discusses the impact of flexibility when designing a Viterbi decoder for both convolutional and TCM codes. Different trade-offs have to be considered in choosing the right architecture for the processing blocks and the resulting hardware penalty is evaluated. We study the impact of symbol quantization that degrades performance and affects the wordlength of the rate-flexible trellis datapath. A radix-2-based architecture for this datapath relaxes the hardware requirements on the branch metric and survivor path blocks substantially. The cost of flexibility in terms of cell area and power consumption is explored by an investigation of synthesized designs that provide different transmission rates. Two designs were fabricated in a 0.13 μm digital CMOS process and verified for functionality. Based on post-layout simulations, a symbol baud rate of 168 Mbaud/s is achieved in TCM mode, equivalent to a maximum throughput of 840 Mbit/s using a 64-QAM constellation.

Based on: M. Kamuf, V. Öwall, and J. B. Anderson, "Optimization and implementation of a Viterbi decoder under flexibility constraints," *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, submitted.
and: ——, "Architectural considerations for rate-flexible trellis processing blocks," in *Proceedings of IEEE International Symposium on Personal, Indoor, and Mobile Radio Communication*, Berlin, Germany, Sept. 2005, pp. 1076–1080.

1 Introduction

With growing application diversity in mobile communication, the need for flexible processing hardware has increased. Consider, for instance, high-rate wireless personal area networks (WPANs) [101], which provide short-range (<10 m) ad-hoc connectivity for mobile consumer electronics and communication devices. In this environment, different transmission schemes and code rates are required in order to adjust to varying channel conditions [34, 65]. A flexible channel decoding platform should be able to provide at least two decoding modes, one when good error-correcting capability is required at low SNR, and one supporting high data throughput if the channel is good. According to this requirement, IEEE 802.15.3 [101] suggests coded modulation schemes to gradually adjust data throughput. These schemes range from QPSK to 64-QAM and are based on a symbol rate of 11 Mbaud/s.

As part of the mentioned standard, TCM [95, 96] enables transmitting information at high rates per Hertz of bandwidth. It is most efficient for higher (quadrature) constellations beyond QPSK, which carry more than two data symbols per two-dimensional channel use. The subset selectors of the TCM codes are rate 1/2 for QPSK and rate 2/3 for 16-QAM, 32-CR, and 64-QAM constellations. For QPSK, one data bit is transmitted per channel use, and the data rate becomes 11 Mbit/s. The higher modulations cover data rates in multiples of the symbol rate. Trellis-coded 64-QAM is the highest constellation considered in the standard and carries five data bits per channel use. Thus, the maximum data rate is 55 Mbit/s.

Shown in Figure III–1 are the codes' BER in the AWGN channel. For comparison, the Shannon limit for equivalent rate-R (bits/dimension) systems is also drawn. The transmission rate R is derived according to Equation 2.1. For example, since there is one coded bit per two-dimensional constellation point, the transmission rate for 16-QAM TCM is 3 bits per two dimensions, equivalent to $R = 1.5$ bits/dimension. The target BER in the standard is around 10^{-5}; here the transmission schemes using higher constellations are roughly 5 dB from the Shannon limit.

The QPSK scheme is considered as a dropback mode for low SNR. In such a scenario, Gray-mapped rate 1/2 convolutional codes are usually preferred since they can achieve the same transmission rate R as TCM with better BER performance. Simulations show that at the target BER, the best 8-state rate 1/2 convolutional code together with Gray-mapped QPSK is about 0.3 dB better than the TCM QPSK scheme; see Figure 2.11. Therefore, BPSK or QPSK are generally used together with rate $1/c$ convolutional codes, or punctured codes derived therefrom. The trellis diagram of these codes can be decomposed into a radix-2 (R2) butterfly state interconnect structure as in Figure 2.6(a).

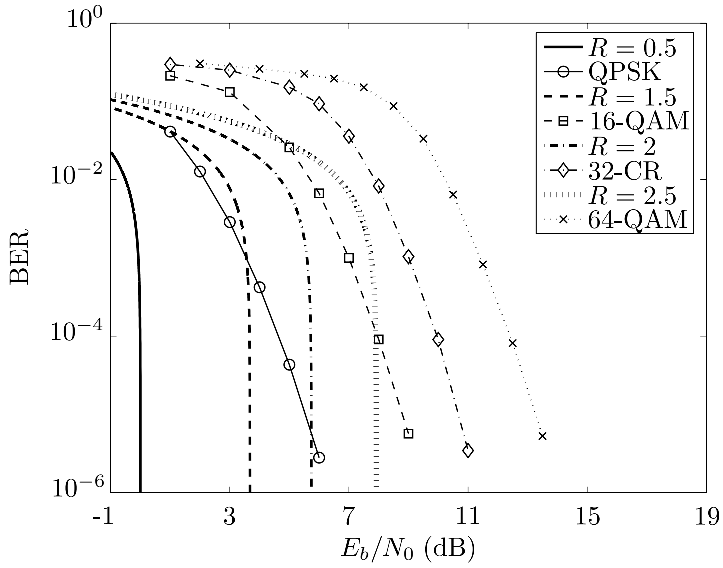


Figure III–1: BER performance in AWGN of TCM codes from [101] together with the Shannon limit of equivalent rate-R (bits/dimension) systems.

Recall from Figure 2.11 that coding in the higher-energy region, where larger constellations are used for transmission, is a domain of TCM. To date, the most practical codes for TCM used together with two-dimensional modulation schemes appear for $b = 2$. Puncturing, however, is not applicable if code performance is to be fully maintained. This degradation stems from altering the minimum inter-subset distance [6]. Thus, the trellis of the TCM subset selectors consists of radix-4 (R4) butterflies; see Figure 2.6(b). Note, however, that for PSK-constellations some pragmatic codes based on punctured subset selectors were found in [99, 102]. Since PSK is a weak master constellation because its signal points are sparsely packed, it is not further considered.

To summarize these considerations, the flexible channel decoding architecture has to be tailored to efficiently process both R2 and R4 butterflies. Both modes should use the same computational kernel to limit overhead in both area and power consumption. A more general design objective could be stated as follows: increase flexibility with as little sacrifice as possible in area, throughput, and power consumption.

ML decoding is provided by the VA [45, 98], which was described in Chapter 2. We denote by m the number of memory elements in the encoder which is excited by b bits per time step. The number of trellis states is $N = 2^m$ and

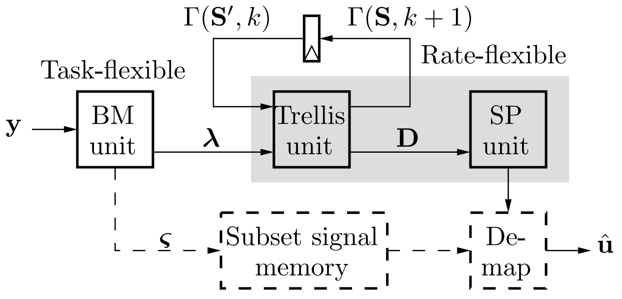


Figure III–2: Block diagram of a flexible Viterbi decoder. Additional parts needed for decoding of TCM codes are dashed.

there are 2^b branches per node that connect these states. Shown in Figure III–2 is a principal architecture for a flexible Viterbi decoder. Let us revisit the three main processing blocks of this decoder, that is, BM, trellis, and SP units, from a flexibility perspective.

Based on the demodulated channel values $\mathbf{y}$, the BM unit provides measures of likelihood $\boldsymbol{\lambda}$ for the transitions in a trellis stage. This unit is strongly related to the task the Viterbi processor is intended for. For example, apart from calculating distances between received and expected symbols as in the case of binary convolutional codes, TCM codes require an additional subset decoder as discussed in Chapter 2. Extension of this architecture to cope with, for example, Viterbi equalization would require another processing part for finding the necessary BMs.

These BMs are consumed by the trellis unit, where ACS operations on the SMs $\Gamma(\mathbf{S}', k)$ at instant k form a new vector of SMs $\Gamma(\mathbf{S}, k+1)$ at instant $k+1$. This operation is equivalent to discarding suboptimal branches in the trellis diagram. Here, $\mathbf{S}'$ denotes the vector of states in a trellis and $\mathbf{S}$ is its permutation according to the given state interconnection, which is determined by the encoder. The architecture of this unit depends on the code rate and number of states in a trellis diagram.

The trellis unit produces an $N \times b$ matrix $\mathbf{D}$ of decision bits about surviving branches. These bits are processed by the SP unit to reconstruct the data bits that caused the transitions. Depending on the algorithm used for the SP unit, the architecture becomes more or less related to the number of bits b and states N per trellis stage. For example, as discussed in Part II, the register-exchange algorithm requires the trellis to be directly mapped onto hardware, which gives a stronger connection to b and N.

Additionally, in case of TCM, the most likely transmitted signals $\boldsymbol{\varsigma}$ for all subsets have to be stored in the subset signal memory. These signals, together with the reconstructed subset sequence from the SP unit, lead to the final decoded data sequence $\hat{\mathbf{u}}$.

After this outline of the general technical parts, there is one major question left. How does flexibility constrain the design process, that is, what trade-offs arise when striving for flexibility? What is the overall hardware cost? These are issues often overlooked in the work on other flexible trellis decoders named in Section 2. No attempt is made to discuss design trade-offs and investigate the cost of the provided flexibility. Therefore, after studying the main building blocks of our flexible Viterbi decoder in Sections 3–5, a quantitative investigation on the cost of flexibility is performed in Section 6. This investigation is supported by a silicon implementation. A cost estimation of an alternative architecture is also discussed.

2 Classification of Flexible Trellis Decoders

Several approaches have been made to incorporate flexibility in the design of trellis decoders. We divide these attempts into the following categories: the first two (m- and algorithm-flexible) are expected to operate in the low energy region and therefore employ small constellations such as BPSK or QPSK. Coding is based on rate $1/c$ convolutional codes, including punctured or concatenated versions therefrom. The last category (bandwidth-flexible) inherently supports larger constellations and different coding schemes, thus facing other design challenges, as mentioned in the introduction for the case of TCM.

2.1 m-flexible Solutions

These approaches use one decoding algorithm and provide flexible error correction by varying the encoder memory m. For example, Chadha [24], Zhu [107], and Hocevar [58] designed flexible VA-based architectures.

Chadha's implementation provides a fully parallel solution (up to $m_{\max} = 6$) and shuts down unnecessary parts when processing trellises with fewer states. The extra hardware spent in the flexible designs is compared to a fixed design with the same $m_{\max}$. This overhead is at most 2.9%, which is not surprising since it mainly accounts for shut-down logic and routing resources. What is missing is an evaluation of the provided flexibility compared to fixed designs with $m < m_{\max}$. In this case, an increasing relative overhead should be encountered as m decreases. Supported code rates are $1/2$ and $1/3$. Code rate is not a critical design parameter when used with antipodal constellations such as BPSK and QPSK since the calculation of distances to the 2^c code sequences is very simple, as shown in Part I. However, the design does not explicitly provide puncturing resources.

Zhu's reconfigurable decoder ($R_c = 1/2$, $m = 6, \ldots, 9$) works in a folded manner; that is, starting from a trellis unit with 8 ACS units working in parallel, the processing is carried out time-multiplexed. According to the sequential access scheme of the SMs, 5-level pipelining can be introduced in the feedback loop of the ACS units. Reconfigurability is achieved by 4×4 switches that shuffle the SMs between the ACS units and the global SM memory. These switches are steered by a controller that provides the necessary schedule for a given m. This controller is probably the most complex part of the implementation since access patterns change in every iteration and for every encoder memory.

Hocevar's design is a DSP coprocessor, which supports a variety of code rates that are achieved by puncturing the basic 1/2, 1/3, and 1/4 convolutional codes. m is variable from 4 to 8, and 16 states can be processed in parallel. Generally, a processor's flexibility is inherently larger than a tailored design such as Chadha's. As discussed in Chapter 3, the price for this flexibility is paid by throughput degradation.

A different class of reconfigurability is the dynamically adaptive Viterbi decoder investigated by Tessier [92]. It is an FPGA-based approach that can be reconfigured externally to cope with varying channel SNR. The designs that can be loaded range from $m = 3, \ldots, 13$. Power is saved compared to a static decoder by adaptively updating only a predefined portion of the trellis with paths that have the least cumulative distance.

A flexible max-log-MAP decoder with $m_{\max} = 4$ is presented in [54]. The evaluation is carried out similar to [24], and hence the information about the introduced overhead does not cover aspects missing in Chadha's investigation.

2.2 Algorithm-flexible Solutions

A natural combination is the (soft-output) VA together with the (max-)log-MAP algorithm since they share the main processing engine, the ACS operation. For example, Bickerstaff *et al.* [14] provide 256-state Viterbi and 8-state log-MAP decoding. The trellis block processes 8 states in parallel, that is, the Viterbi decoding is carried out time-multiplexed. Since this design is to be (commercially) used in third generation mobile services, it includes several communication interfaces and an evaluation of the cost of flexibility was not of main interest.

In Cavalloro's work [23] the combination of VA with an augmented soft-output unit is investigated. Encoder memory is variable up to $m_{\max} = 8$. The approach is in principle based on the work of Chadha [24], that is, the contribution of flexibility lies mainly in shut-down logic and routing resources.

2.3 Bandwidth-flexible Solutions

A Viterbi processor for TCM codes is presented by Lou in [69]. It is widely programmable and executes the main decoding operations sequentially. Flexibility is achieved by look-up tables and some dedicated computational blocks. 32 states and both R2 and R4 processing are supported. Codes with a maximum of 8 subsets are allowed and two- or four-dimensional symbols can be processed.

Miyauchi *et al.* [72] describe a fully integrated dedicated soft-input soft-output processor, whose focus is on iterative decoding. It has many features such as arbitrary coding polynomials, interleaving patterns, and constellation configurations. Different classes of coding approaches are supported, parallel/serial concatenated convolutional codes, turbo TCM [81], and serial concatenated TCM. The highest constellation considered is 8-PSK. Design challenges in this approach are mainly concerned with the incorporation of R4-processing to log-MAP decoding.

Our flexible Viterbi decoder also belongs to the bandwidth-flexible class. It covers a wider range of transmission rates in order to adapt to both low- and high-energy scenarios.

2.4 Performance Evaluation

In this section, the different approaches are compared in an energy–bandwidth sense. Figure III–3 shows some implemented flexible trellis decoders and their energy–bandwidth performance to achieve a BER of 10^{-5} in the AWGN channel. Transmission rates from Equation 2.1 are now related to bandwidth. It is assumed that an AWGN channel use with $J = 2$ dimensions occurs every symbol time $T_s = T_b \cdot R$. Here T_b is time per data bit. The number of data bits is $R = \log_2 \mathcal{M} \cdot R_c$ for convolutional and $R = \log_2 \mathcal{M} - 1$ for TCM codes. RF bandwidth $W_{\mathrm{RF}} = 1.3/T_s$ is normalized to T_b and includes excess bandwidth introduced by 30% root-raised-cosine pulses. The Shannon AWGN capacity C is also shown for comparison. Flexible decoders that principally support larger constellations beyond QPSK but do not provide a (soft-output) demapper [94] belong in a bandwidth sense to QPSK-systems.

From Figure III–3 note that m- and algorithm-flexible designs experience throughput limitations as the channel SNR improves. They just trade required energy for bandwidth, which is indicated by vertical lines. Throughput can only be varied on a small scale. One solution is to employ higher constellations together with (punctured) convolutional codes to gradually increase data rates. However, compared to TCM, which was intended for higher constellations, these systems are not as energy-efficient for the same BER, throughput, and complexity. The BER curves in Figure 2.11 confirm this statement.

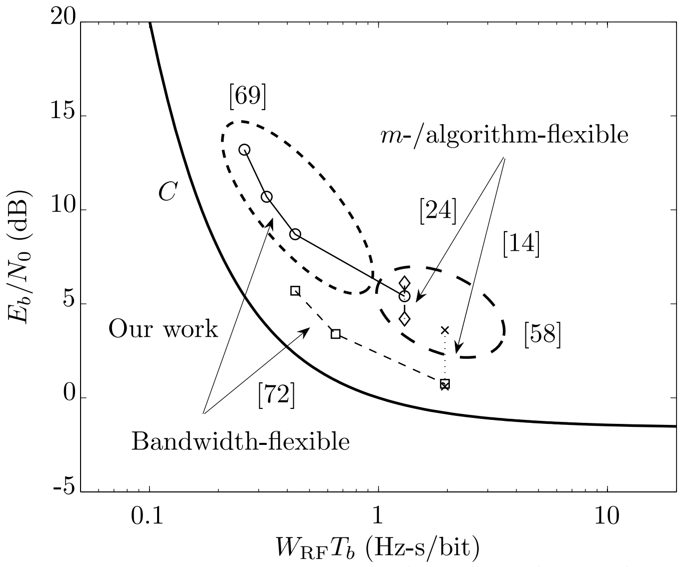


Figure III–3: Energy–bandwidth performance of some flexible trellis decoders for BER of 10^{-5}. Shannon AWGN capacity C is drawn bold for comparison.

Not surprisingly, [58] and [69] provide the highest flexibility since these are programmable trellis processors. Realizable systems are visually bound by a sphere. Note again that this flexibility degrades processing speed and power consumption by several orders of magnitude compared to dedicated solutions [37].

It is also noteworthy that systems become more complex to implement the closer to capacity they get. Consider Miyauchi's design [72], which apparently provides a good energy–bandwidth trade-off with help of iterative decoding. One must bear in mind, however, that iterative decoding schemes run multiple times over a trellis, increasing latency and raw computational cost per decoded bit. That is, for low-power low-cost applications as in WPAN, this design is certainly overdesigned. The flexible Viterbi decoder described in our work is a lower-complexity solution that adjusts to varying channel conditions by providing several transmission rates using different constellations.

There could be many more dimensions added in Figure III–3, for example, latency, computational complexity, energy consumption per decoded bit, and so on. Such measures ultimately give an overall cost per decoded bit and depending on what is crucial for a certain application, a design choice becomes evident.

3 Branch Metric Unit

The architectural discussion of the flexible Viterbi decoder starts by introducing an applicable distance measure in the context of TCM. An investigation of symbol quantization follows. Its impact on cutoff rate as well as error performance is evaluated, which ultimately leads to the required BM wordlength that in turn determines the wordlength of the trellis datapath.

Throughout this section, consider two-dimensional quadrature modulations that consist of two independent pulse amplitude modulations (PAMs) generated from orthogonal pulses. Dimensions $i = 0, 1$ relate to in-phase ($\mathcal{I}$) and quadrature-phase ($\mathcal{Q}$) signal components, respectively.

To provide measures of likelihood for the state transitions, different metrics are appropriate for different channel models. In the AWGN channel, the optimal distance measure is the squared Euclidean distance between received channel value $\mathbf{y}$ and constellation symbol $\mathbf{c}$,

$$\sum_{i=0}^{1} (y_i - c_i)^2, \tag{III–1}$$

where c_i is from the alphabet of M-PAM. In case of binary signaling per dimension where $c_i \in \{-1, +1\}$, recall from Section 2 in Part I that Equation III–1 simplifies to $-\sum_i y_i c_i$. For larger constellations, different c_i^2 have to be taken into account in the distance calculation. Here, possible simplifications use the constellation's bit mapping, commonly Gray, to determine soft bit values [94] that can be used by the ML decoder. Due to TCM's subset partitioning, assigning such soft bit values is not meaningful since Gray mapping is only applied to points in the same subset, not to the constellation itself. Furthermore, only the distance to the nearest constellation point in a specific subset is considered as BM for that subset in TCM decoding. This point has to be determined for each subset before the BM calculations. Figure III–4 shows part of the lattice $\mathbb{Z}^2$, where $D_{\min}$ denotes the minimum distance between lattice points. The subset distribution is represented by circles, squares, and triangles. The largest distance appears if the $\mathbf{y}$ matches a lattice point and the resulting maximum attainable BM $\lambda_{\max}$ is bound by a circle of radius $2D_{\min}$.

For a given finite constellation (here part of 64-QAM), the situation is slightly different. Assume that the received values are limited by the dynamic range of the analog-to-digital (A/D) conversion. Receiving a value equal to a corner point gives the largest distance ($\sqrt{10}D_{\min}$) to any point in the ■-subset. This is indicated by the solid quarter circle in Figure III–4. Depending on the dynamic range, which ultimately determines the mapping of constellation points with respect to the maximum number range, one might have to consider

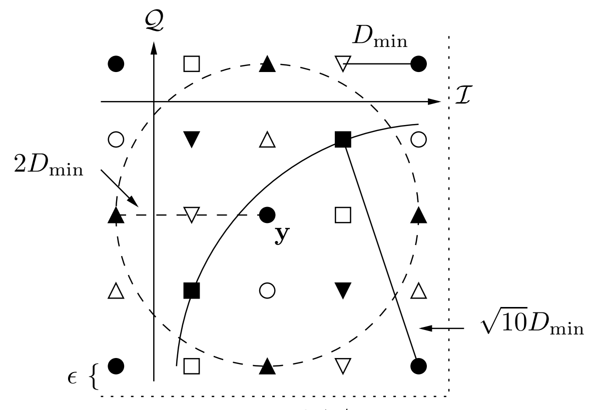


Figure III–4: Maximum distance between a received point and constellation point belonging to one of 8 subsets, which are represented by circles, squares, and triangles.

an additional margin ϵ per dimension. This term will be included in the BM calculations in the following subsection.

3.1 Quantization Issues

One method of wordlength design [74] is based on the cutoff rate R_0, which is a lower bound to capacity for any specific signaling constellation. Consider a communication system using M-PAM at the transmitter and a receiver that puts out noisy channel symbols quantized with q bits. This arrangement constitutes a discrete memoryless channel (DMC) with M inputs and 2^q outputs. Let the M symbols be equiprobable (symmetric cutoff rate [66]). Thus

$$R_0 = \log_2 M - \log_2 \left\{ \frac{1}{M} \sum_{i=0}^{2^q-1} \left[\sum_{j=0}^{M-1} \sqrt{P(i|j)} \right]^2 \right\}, \tag{III–2}$$

where the transition probabilities for AWGN are

$$P(i|j) = \int_{r_i} \frac{1}{\sqrt{\pi N_0}}\, e^{-(y-s_j)^2/N_0}\, dy, \tag{III–3}$$

and $\{s_j\}$ is the set of M equally spaced constellation points over the interval $[-\sqrt{E_s}, +\sqrt{E_s}]$. A quantization scheme for this DMC is considered optimal for a specific choice of q and N_0 if it maximizes Equation III–2. For simplicity, assume a uniform quantization scheme with precision Δ. The thresholds that bound the integration areas r_i in Equation III–3 are located at either

$$0, \pm\Delta, \dots, \pm(2^{q-1} - 1)\Delta \text{ (truncation)},$$

or

$$\pm\Delta/2, \dots, \pm(2^{q-1})\Delta/2 \text{ (rounding)}.$$

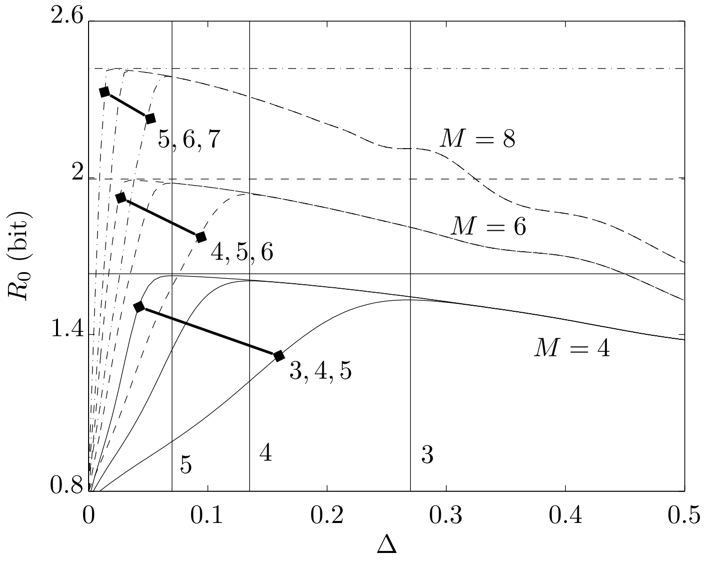


Figure III–5: Cutoff rates for a DMC with M equally spaced inputs ($\sqrt{E_s} = 1$) and 2^q uniformly quantized outputs. E_b/N_0 is chosen for desired quadrature BER of 10^{-5}, see Figure III–1. Connected curves are in increasing order of q from right to left. Vertical lines denote the optimal Δ for a chosen q found beside the line. The respective maximum R_0 for unquantized channel outputs is indicated by the horizontal lines.

An information theoretical approach could involve an evaluation of $\partial R_0/\partial\Delta$ to determine the Δ that maximizes R_0 given M, q, and N_0. However, since there is no analytical solution to the integral in Equation III–3 for $r_i < \infty$, numerical calculations have to be carried out to find the optimal threshold spacing Δ, which in turn determines the dynamic range of the A/D conversion. Figure III–5 shows the cutoff rates R_0 for different M-PAM schemes. E_b/N_0 is chosen to achieve a quadrature BER of 10^{-5}. The corresponding M^2-QAM schemes are found in Figure III–1. Here, 36-QAM is reduced to 32-CR by removing the four corner points. This gives a lower energy constellation, whose number of points is a power of two. The vertical lines denote the optimal Δ given q. The upper limit on R_0 for unquantized channel outputs is indicated by the horizontal lines. For example, given $M = 4$ and $q = 4$, the optimal Δ is around 0.135, which yields a dynamic range of $16 \times 0.135 = 2.16$. Δ is based on the estimated noise. Considering the precision of the estimation, it is already seen from Figure III–5 that a deviation towards a slightly higher Δ than the optimal one is more tolerable. That is, the slope of a cutoff rate curve is rather flat after its maximum, whereas it is comparably steep before it.

Having found a range of feasible Δ, one can evaluate the degradation in BER due to this channel quantization. Furthermore, to lower the complexity of the distance calculation in Equation III–1, which involves squaring, a suboptimal metric is used for the higher constellations. It considers only the absolute distances in either dimension and Equation III–1 is replaced by

$$|y_0 - c_0| + |y_1 - c_1|. \tag{III–4}$$

Table III–1 summarizes the expected loss in E_b/N_0 compared to Equation III–1 and unquantized channel outputs. As expected, for a certain constellation the loss becomes smaller as q increases, and larger constellations generally require finer quantization. QPSK together with binary convolutional coding is not listed in Table III–1 since it is well known that 3 bits symbol quantization is usually sufficient for good performance [57]. Following Equation I–2, a rate 1/2 convolutional code with two 3-bit code symbols therefore requires 4 bits for the BM.

In order to determine $\lambda_{\max}$, one can consider two approaches. First, to guarantee negligible performance degradation for all schemes, one could choose the q that leads to the largest tolerable degradation for the largest constellation, in this case 64-QAM. Then, the overall loss in E_b/N_0 is no worse in all transmission modes. Or secondly, one could vary the wordlength of the A/D-samples to achieve the required tolerable degradation for each mode.

Pursuing the first approach, we assume $q = 7$ bits to perform close to optimal for 64-QAM; see Table III–1. This choice provides negligible degradation for the other two constellations. In case the largest q-bit number exceeds the number assigned to the largest constellation point in either $\mathcal{I}$ or $\mathcal{Q}$, an additional factor ϵ has to be added per dimension. From Figure III–4 and Equation III–4, the largest BM becomes

$$\lambda_{\max} = 4D_{\min} + 2\epsilon. \tag{III–5}$$

Choosing Δ according to Figure III–5 both to maximize R_0 and achieve equidistant spacing between constellations points yields $D_{\min} = 18$. Then $\lambda_{\max}$ re-

Table III–1: Loss in E_b/N_0 for BER of 10^{-5} for uniform symbol quantization with q bits and optimum choice of Δ. Absolute distances are used according to Equation III–4. As a reference, row $q = \infty$ shows the required E_b/N_0 with unquantized inputs and Euclidean distance as in Equation III–1.

q	16-QAM	32-CR	64-QAM
∞	8.7	10.7	13.2
7	≈ 0	≈ 0	0.05
6	≈ 0	0.05	0.3
5	0.15	0.31	1.15
4	0.4	0.85	n/a
3	1.4	n/a	n/a

quires at least $\lceil \log_2 72 \rceil = 7$ bits. If the A/D-conversion uses its dynamic range efficiently, ϵ is small compared to the first term in Equation III–5 and the number of bits will be sufficient. Using the same Δ also for 16-QAM and 32-CR increases the number of quantization levels between two constellation points so that 8 and 7 bits are now required for the BMs. That is, the lowest constellation needs the largest number of bits, which means that the architecture is overdesigned.

Considering the second approach, that is, adjusting the wordlength of the A/D-samples, assume that 16-QAM and 32-CR employ $q = 5$ and $q = 6$ bits, respectively. This yields a $D_{\min}$ of 9 and 13, and the largest BM becomes at least 36 and 52. $\lambda_{\max}$ can now be represented by 6 bits, and the largest BM range applies to the highest constellation. The candidate q are shaded in Table III–1.

3.2 Subset Decoding and Signal Memory

In contrast to binary convolutional codes, which carry code symbols along trellis branches, TCM codes carry subsets that themselves consist of signals. Before BM calculations can be done one has to determine the most likely transmitted signal for each subset. This process is called subset decoding. For example, the decision boundaries for subset $\mathcal{C}_0$ are depicted in Figure III–6 for the different constellations. The $\mathcal{I}$ and $\mathcal{Q}$ channel values are y_0 and y_1. In order to find the most likely subset point, comparisons of $y_1 - y_0$ and $y_1 + y_0$ to boundaries are needed. Furthermore, if there are more than two points per subset, additional comparisons are required to resolve ambiguities. Points that lie in the shaded area in Figure III–6(b) yield the same comparison result if one only considers diagonal boundaries. Hence, for 32-CR and 64-QAM, comparisons with y_1 and y_0 describe the necessary horizontal and vertical boundaries. To determine the most likely points for the other subsets, one can either translate the input symbols relative to $\mathcal{C}_0$ or simply adjust the comparison values for the boundaries.

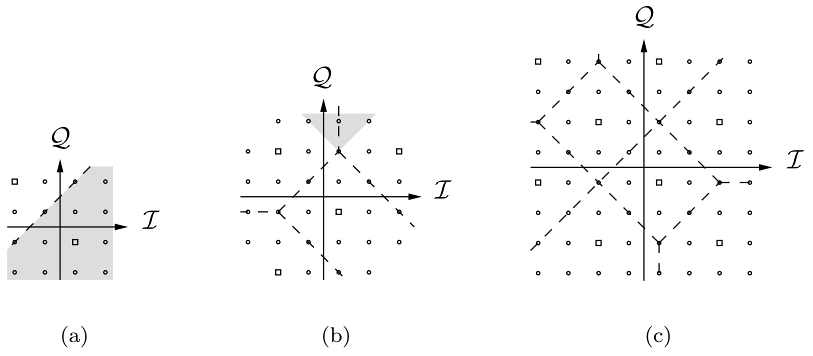


Figure III–6: Decision boundaries for subset $\mathcal{C}_0$ (squares) for (a) 16-QAM, (b) 32-CR, and (c) 64-QAM constellations. For example, the shaded part in (a) shows the region where $y_1 - y_0 < 2/3$.

To evaluate the computational effort, consider the lattice $\mathbb{Z}^2$ with an M^2-point constellation that is divided into $1 < P \leq M^2$ subsets with M^2/P points per subset. With $\omega = \log_2 M^2/P$, this setup requires

$$\Omega = \begin{cases} (2\omega - 1) + 2\min\{\omega - 1, 2\}, & \text{for } P = 2^{2\ell - 1} \\ 0 + 2\,(\omega - 1), & \text{for } P = 2^{2\ell} \end{cases} \tag{III–6}$$

slicing operations (comparisons with a constant) along the decision boundaries. The number of comparisons in Equation III–6 is split into two terms: the first relates to diagonal boundaries and the second to horizontal/vertical boundaries. For $P = 2^{2\ell} = 4, 16, 64, \ldots$ subsets, there are only horizontal/vertical boundaries. The boolean comparison results are demapped to a unique point in the subset.

The complete architecture for the BM unit is depicted in Figure III–7. In total there are ΩP slicing operations and P BM calculations according to Equation III–1 or Equation III–4. These BMs appear as $\boldsymbol{\lambda} = \{\lambda_j\}$ for $j = 0, \ldots, P-1$ in Figure III–2. The additional hardware due to TCM decoding is indicated in

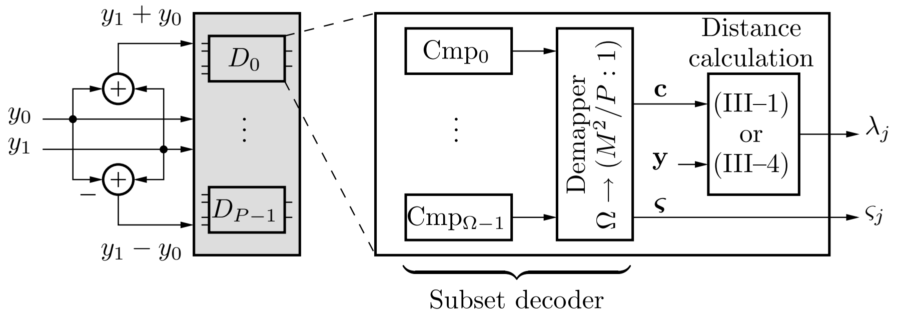


Figure III–7: Architecture of the BM unit for the flexible decoder. The gray part is the additional hardware required due to the use of higher constellations and TCM.

gray in Figure III–7. A decoder for subset $\mathcal{P}_j$ consists of Ω comparisons and a demapper (look-up table) that chooses from these comparison bits one out of M^2/P constellation points. This point $\mathbf{c}$ is used for distance calculation to yield the BM λ_j. The calculations needed for subset decoding, $y_1 - y_0$ and $y_1 + y_0$, can be reused in case of rate 1/2 convolutional coding. These results are equivalent to the BMs for code symbols $\{+1 -1\}$ and $\{-1 -1\}$, respectively. The remaining two metrics are derived from these by negation.

As already mentioned in Section 1, TCM introduces overhead in this flexible architecture: the unit that stores the candidate surviving signals. These "uncoded" bits represent subset points at each trellis stage and together with the reconstructed survivor path form the decoded output sequence. The unit comprises a memory that stores the $\log_2 M^2/P$ bits ς_j from the subset decoder for all P subsets. The length of this first-in first-out (FIFO) buffer equals the latency of the SP unit. For the implementation to be power-efficient, this part has to be shut down when TCM is not employed.

4 Trellis Unit

The trellis unit consists of ACS units that are arranged and connected with each other in a butterfly fashion. These units deliver updated SMs $\Gamma(\mathbf{S}, k+1)$ and decisions $\mathbf{D}$ based on previous SMs $\Gamma(\mathbf{S}', k)$ and present BMs $\boldsymbol{\lambda}$. According to the considerations in the introduction, this unit has to cope with two different code rates, 1/2 and 2/3, and hence both R2 and R4 butterflies are to be processed.

The design objective is the following: given a fixed R2 feedback network for the SMs, how can R4 butterflies be efficiently mapped onto this architecture, so the existing interconnections can be reused? In this section, the issue of rate-flexibility in the trellis unit is discussed and a framework is derived for emulating R4 butterflies by means of R2 butterflies. Note that almost all practical trellis codes are based on either R2 or R4 butterflies. Therefore, the investigated architecture also extends to other codes.

4.1 Considered Trellises

The base architecture for the flexible trellis unit is an R2 architecture for the systematic 8-state rate 1/2 convolutional code with tap set $(\mathbf{g}_0, \mathbf{g}_1) = (54, 74)$ realized in controller form, where $\mathbf{g}_0$ are the feedback taps. The trellis consists of butterflies with state transitions according to Figure 2.6(a) in Chapter 2.

The systematic encoding matrix of the TCM code is the one from Equation 2.7. The TCM subset selector is realized in observer form (see Figure 2.3), since this maintains the number of memory elements of the underlying systematic code if $b > 1$. However, state transitions in the observer form depend on the tap set. That is, the feedback network has to be flexible if one wants to process both R2- and R4-based codes on a single R2 architecture. This can be done by introducing additional routing resources that modify the permutation in R4 mode to feed back the state metrics in correct order in both processing modes. In this design example, though, the state transitions of the TCM subset selector allow the reuse of the trellis feedback connections of the binary code. Figure III–8 shows the interconnection structure of a trellis stage for the considered encoders. It is seen that the feedback connections for both R2 and R4 architectures are the same, for example, $\Gamma(2, k+1)$ in R4 mode is fed back along the same connection as $\Gamma(1, k+1)$ in R2 mode.

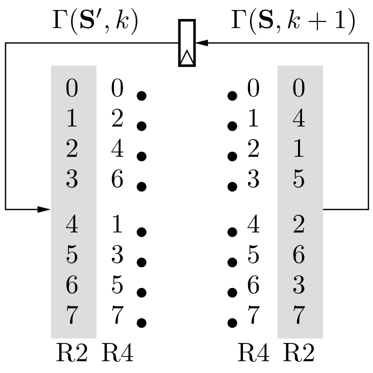


Figure III–8: Feedback connections for R2 and R4 trellis processing.

4.2 Processing Framework

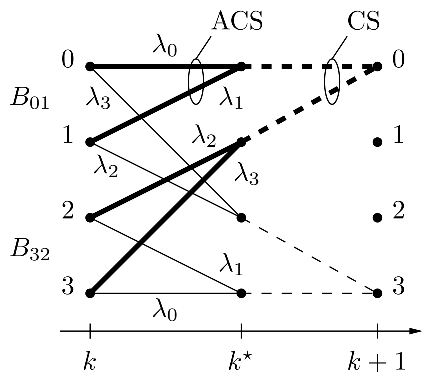
In the following, an R4 butterfly is considered which uses BMs λ_i for $i = 0, \dots, 3$. As in Figure 2.6, there are $N/2^b$ butterflies in a trellis, and thus, for $N = 4$ and $b = 2$, there is only one R4 butterfly. Since $n \in [0, N/2^b - 1]$, we have $n = 0$ and the state labels become $0, \dots, 3$.

To update one state in an R4 butterfly, one can carry out all six possible partial comparisons in parallel [15]. Four such operations are needed to calculate a complete R4 butterfly as in Figure 2.6(b). However, this leads to inefficient hardware reuse in a rate-flexible system due to the arithmetic in the 4-way ACS units. In [37] area–delay complexity of R2- and R4-based butterfly processing is evaluated. Two cases are considered. One where an R2-based trellis is processed with R2 processing elements (PEs), and one where two R2 trellis stages are collapsed into one R4 stage [15], which is processed with the 6-comparator approach. For a standard cell design flow (this includes FPGA implementations), R2 PEs are found to be the more cost-efficient, whereas a full-custom datapath as in [15] benefits the R4 6-comparator method. Since platform independence is an important issue, we use an architecture that consists of R2 butterfly units. In Section 6.3, there is an architectural evaluation of the R4-based approach and its implications.

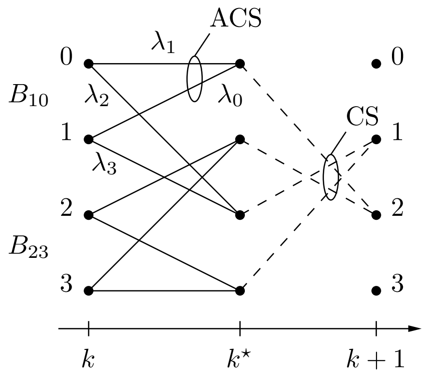
Generally, a 4-way ACS can be carried out in two successive steps: in the first step a pair of cumulative metrics (ACS) is evaluated and discarded; then in the second step, one of the surviving metrics is discarded, which corresponds to a compare-select (CS) operation in Figure III–9. This procedure is a decomposition into R2 operations that are separated in both state and time. For step $(k \rightarrow k^\star)$, the split into four R2 butterflies achieves the cumulation of the SMs with all four BMs. Then, in step $(k^\star \rightarrow k+1)$, the partial survivors are compared and the final survivors selected. Here, Figure III–9(a) updates states 0 and 3, and Figure III–9(b) updates states 1 and 2.

To capture these processing steps formally we use the following notation. The state connectivity of an R2 butterfly is defined in Figure 2.6(a). Assume that the two states at time k are named u' and v' with state metrics $\Gamma(u', k)$ and $\Gamma(v', k)$, respectively. The two ACS operations leading to two updated state metrics for states u and v at stage $k+1$ are expressed as butterfly operation $B_{u'v'}$. Without loss of generality, the λ_i are distributed as in

$$B_{u'v'} = \begin{pmatrix} \Gamma(u, k+1) \\ \Gamma(v, k+1) \end{pmatrix} = \begin{pmatrix} \min \begin{pmatrix} \Gamma(u', k) + \lambda_0 \\ \Gamma(v', k) + \lambda_1 \end{pmatrix} \\ \min \begin{pmatrix} \Gamma(v', k) + \lambda_2 \\ \Gamma(u', k) + \lambda_3 \end{pmatrix} \end{pmatrix} . \qquad \text{(III–7)}$$



(a)



(b)

Figure III–9: A decomposed radix-4 butterfly and the two partial computations leading to the updated metrics for states 0 and 3 in (a) and 1 and 2 in (b). As an example, the necessary operations to update state metric 0 are drawn bold in (a).

It is already seen from Figure III–9 that there are four such R2 butterflies between k and $k^\star$, so four operations as in Equation III–7 are needed. For example, B_{01} is shown in Figure III–9(a), that is, $u' = 0$ and $v' = 1$.

Processing the R4 butterfly based on Equation III–7 preserves the compatibility with the base R2 architecture. The scheme for obtaining all partial survivors is then expressed as

$$\mathbf{B}' = \begin{pmatrix} B_{01} & B_{10} \end{pmatrix}, \tag{III–8}$$

where the columns determine the instance of an iteration. So far we have only computed half of the partial survivors needed; to complete the R4 butterfly another unit has to carry out

$$\mathbf{B}'' = \begin{pmatrix} B_{23} & B_{32} \end{pmatrix}. \tag{III–9}$$

The operations in Equation III–8 and Equation III–9 guarantee that all state metrics at stage k are added to all BMs, that is, 16 partial sums are reduced to 8 partial survivors at intermediate stage $k^\star$ by means of CS operations. The final surviving state metrics at stage $k+1$ are obtained by CS operations on the hitherto surviving metric pairs. Note that the partial survivors are not altered and therefore the final state metrics are not changed compared to a straightforward implementation.

4.3 Architectural Issues

Consider a trellis unit that updates N states in parallel by means of $N/4$ PEs, each consuming and producing 4 state metrics. A PE is configured with butterfly (Bf) units of different radices as in Figure III–10. In Figure III–10(c), vec means the stacking of a matrix columnwise. Either two Bf2 units, two rate-flexible Bf2/4 units, or one Bf4 unit are employed. Note that the Bf4-based architecture can also be configured for rate-flexible processing, whereas a Bf2-based design is solely intended for R2 processing and is not discussed further. That is, the basic PEs are ACS units that consume 4 inputs at a time (4-way ACS units) and thus the cumulation is done in one step.

R2-based Approach

In the rate-flexible architecture using Bf2/4, all partial survivors are calculated during two cycles, and in the third cycle the final update takes place. As an example, the operations to update state metric 0 are drawn bold in Figure III–9(a). The partial survivors needed for the final CS are created at instance 0 by operation $\mathbf{B}'$ and at instance 1 by $\mathbf{B}''$. These operations are carried out in different butterfly units; that is, the partial survivors have to

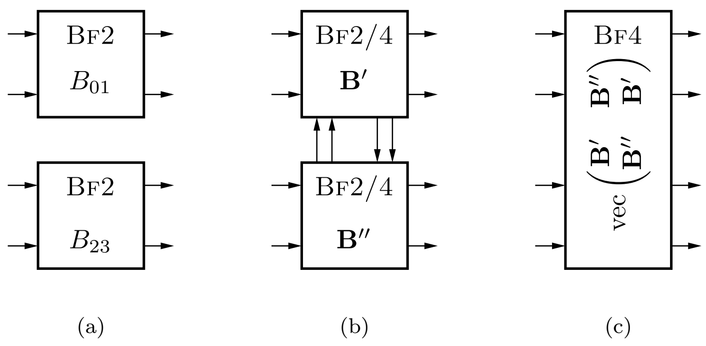

Bf2
B_{01}
Bf2
B_{23}
Bf2/4
$\mathbf{B}'$
Bf2/4
$\mathbf{B}''$
Bf4
$\operatorname{vec}\begin{pmatrix}\mathbf{B}' & \mathbf{B}'' \\ \mathbf{B}'' & \mathbf{B}'\end{pmatrix}$
(a)
(b)
(c)

Figure III–10: Butterfly units that are instantiated inside a processing element. A setup as in (a) supports only R2 processing, setups (b) and (c) are rate-flexible.

be stored temporarily. The appropriate routing for the final CS is according to the required ordering of the updated SMs. Here, the partial survivors are brought together by means of I/O channels between adjacent butterfly units as indicated in Figure III–10(b).

Figure III–11 shows the rate-flexible butterfly unit Bf2/4. Its arithmetic components, adders and the CS units, are identical to the ones in a Bf2 unit; that is, if the gray parts are removed, one gets a standard Bf2 unit. To cope with a decomposed R4 butterfly, routing resources are provided to distribute the partial survivors as dictated by the BM distribution and the state transitions. The input multiplexers (MUXes) shuffle the two input SMs to guarantee their cumulation with all four BMs. The 4 : 2 MUXes in front of the CS units select whether the partial survivors at stage $k^\star$ are to be captured into the routing unit Perm or the final comparison at stage $k+1$ is to be performed. To carry out Equation III–8 or Equation III–9, Perm is fed during two cycles. Then in the third and final cycle, the partial survivors are compared in the CS units. Here, the signals I and O provide the connections to the adjacent butterfly unit to carry out the comparison with the desired pairs of partial survivors. For example, O_0 is connected to I_0 in the adjacent butterfly unit. The CS operations at steps $(k \to k^\star)$ and $(k^\star \to k+1)$ in Figure III–9 are executed by the same CS unit, thus saving hardware.

The unit Perm simply consists of two tapped delay lines. If all SMs can be accessed in simultaneously, for example, SMs are stored in a bank of registers, these registers can be reused to temporarily store the second intermediate

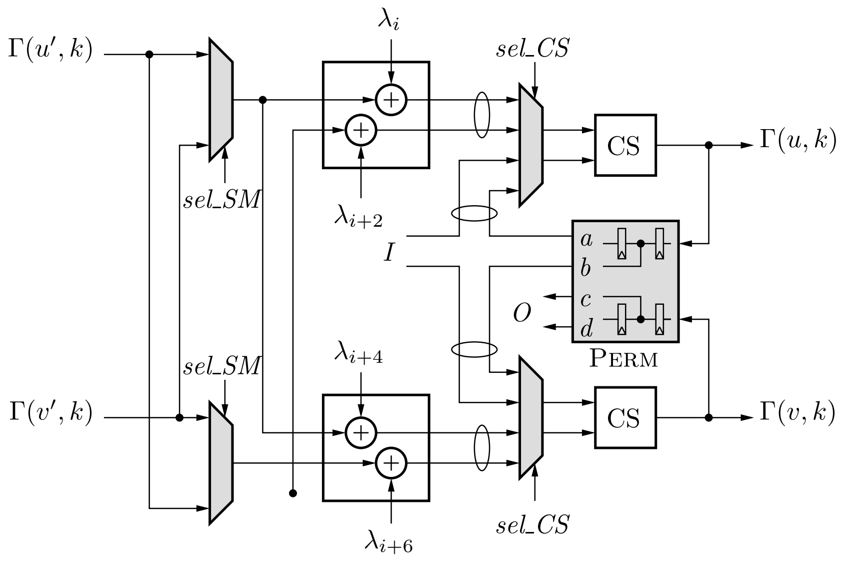


Figure III–11: The rate-flexible butterfly unit BF2/4. An R4 butterfly is updated in three clock cycles. The shaded blocks are the overhead compared to an R2 butterfly unit BF2. The connections for the routing block PERM apply to one of the two pairs of BF2/4 in the design example.

survivors. Hence, Perm would be reduced to only two storage elements that capture the first intermediate survivors. If it turns out that the metric in the global register is the surviving one, the final update can be suppressed and no switching power would be consumed. On average, 50% of the final updates are superfluous.

In order to reuse the feedback network in Figure III–8 for the TCM code, Perm carries out the same permutation in a pair of adjacent butterfly units. Both code trellises have 8 states and thus there are two such pairs processing 4 states each. For pair $i = 0$, the partial survivors on the top rail, a and b, are devoted to the same butterfly unit, whereas the bottom rail survivors, c and d, are assigned to the adjacent butterfly unit. For pair $i = 1$, it is vice versa. Now the design fits seamlessly into the base architecture. Furthermore, the survivor symbols to be processed by the SP unit become equivalent to the information symbols. It will be seen that this is beneficial for the chosen implementation of the SP unit.

The required order of state metrics can vary for different trellises. The butterflies $B_{u'v'}$ in Equation III–8 and Equation III–9 can be calculated in any other order and the partial survivors needed for the updates are shuffled accordingly. Perm allows $4! = 24$ possible permutations to handle these cases. Thus it becomes possible to cover a wider range of encoders. If the decoder implementation has to cope with different encoders and routing requirements, Perm has to be programmable.

It was mentioned that the permutation transforming previous states $\mathbf{S}'$ into $\mathbf{S}$ is the same for both codes considered. This is not generally true, and additional routing is necessary if the required permutation cannot be obtained by the Perm units. Then, SMs have to be exchanged between PEs. Essentially, this is done by inserting (in the worst case N) MUXes to either choose the R2 or R4 state connections. Depending on the application, this can be fixed or programmable.

Finally, a controller is needed to provide control signals to the MUXes (*sel_SM* and *sel_CS*) and clock enable signals to registers. Clocking is only allowed when input data is valid so that no dynamic power is consumed unnecessarily. In R2 mode, these signals are kept constant. Neglecting the controller, the rate-flexible butterfly unit only adds six 2 : 1 MUXes (a 4 : 2 MUX equals two 2 : 1 MUXes) and four (two if the global registers are reused) registers on top of a Bf2 unit, and there is no arithmetic overhead.

R4-based Approach

The presented flexible R2-based approach is now compared to an R4 architecture, which is based on Bf4 units that utilize four 4-way ACS units as in Figure III–10(c). To account for the intended use in a rate-flexible system, similar control mechanisms have to be provided as in the R2-based approach. Hence, a straightforward two-level-CS implementation is considered. Depending on the desired throughput, a butterfly can be updated in one or two clock cycles, which gives the well-known area–delay trade-off. Here, a two-cycle update is employed since this maintains the critical path of the R2-based approach and one CS unit can be reused.

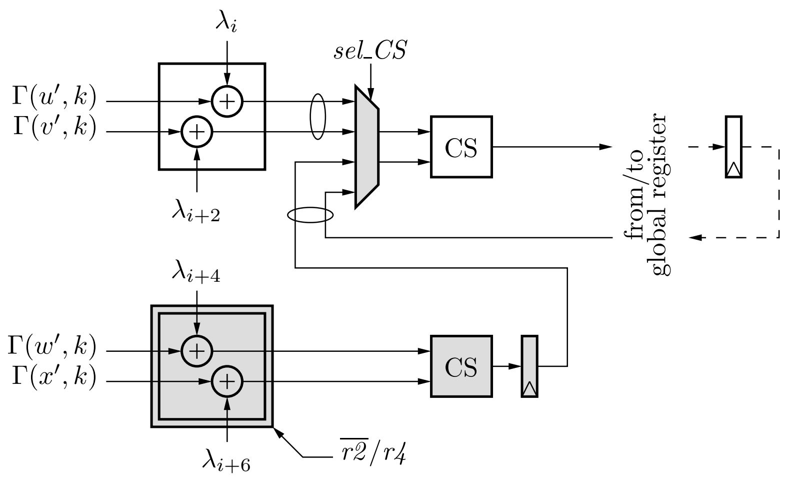


Figure III–12: A flexible 4-way ACS unit for use in BF4 of Figure III–10(c). Four such units are needed for an R4 butterfly to be updated in two clock cycles. The shaded blocks are the overhead compared to a 2-way ACS unit.

Figure III–12 shows the flexible 4-way ACS unit. In R4 mode, two partial survivors are captured in the first cycle. At time $k^\star$, the global SM register in the upper path carries the temporary survivor from either state u' or v' and the shaded register the one from either state w' or x'. In the second cycle, these survivors are compared to yield the final state metric at $k+1$. In R2 mode, only the upper ACS path is utilized and to be equally power-efficient, one needs to prevent switching activity in the lower ACS path. This is done by guarding the inputs of the adders with AND-gates, which is illustrated by the gray shading. The signal $\overline{r2}/r4$, which determines the processing mode, controls whether the addition block is enabled or not. Compared to a conventional 2-way ACS unit, two adders, a CS unit, a register, and a $4:2$ MUX are counted as overhead.

Table III–2 lists the necessary hardware to update an R4 butterfly for the two approaches. The R4-PE requires twice the number of additions compared to the R2-PE, considering that a CS unit consists of an adder and a MUX. The number of MUXes is the same, whereas, in the worst case, twice the number of registers are required in the R2-PE. As mentioned, this can be circumvented by reusing the global SM registers.

Table III–2: Necessary hardware to update an R4 butterfly for the R2- and R4-based approaches. Number of MUXes is expressed as number of equivalent 2 : 1 MUXes. A CS unit consists of an adder and a MUX.

	R2 PE	R4 PE
Add.	8+4	16+8
MUX	12+4	8+8
Reg.	8(4)	4
Cycles	3	2

4.4 Evaluation of Synthesized Trellis Blocks

To show the effects of the architectural considerations in an actual implementation, the trellis blocks are synthesized using a design kit from Faraday for the United Microelectronics Company (UMC) 0.13 μm digital CMOS process. Evaluations apply to synthesized cell area, where different throughput requirements are put as design constraints.

To avoid manual normalization of the otherwise unbounded wordlength increase of SMs in the trellis datapath, the modulo normalization technique from [56] is used. Details of this approach were discussed in Part I.

The datapath wordlengths vary due to varying symbol quantization for the different constellations. We choose $q = 7$ bits, which leads to acceptable degradation for 64-QAM according to Table III–1. The largest representable unsigned BM is $\lambda_{\max} = 127$. From Equation I–16 and Equation I–18, we see that 10 bits are needed to represent the SMs. This wordlength is used for synthesis. Note that for 16-QAM and 32-CR ($q = 5, 6$), only 9 bits are required for the SMs, whereas in the QPSK case ($q = 3$), the wordlength turns out to be 7 bits. That is, in order to power-efficient for both convolutional and TCM decoding, one could slice the datapath to 7 plus 3 extension bits, and prevent the latter bits from toggling in QPSK mode.

Figure III–13 shows the required cell area for synthesized trellis blocks that process R2 and R4 butterflies. Here, $t_{k\rightarrow k+1}$ denotes processing time for a trellis stage from k to $k + 1$. The Bf2/4 architecture takes 3 cycles for an R4 update, whereas the Bf4-based one only needs 2 cycles. For an R2 update, both architectures need one clock cycle. It is seen that the Bf2/4 architec-

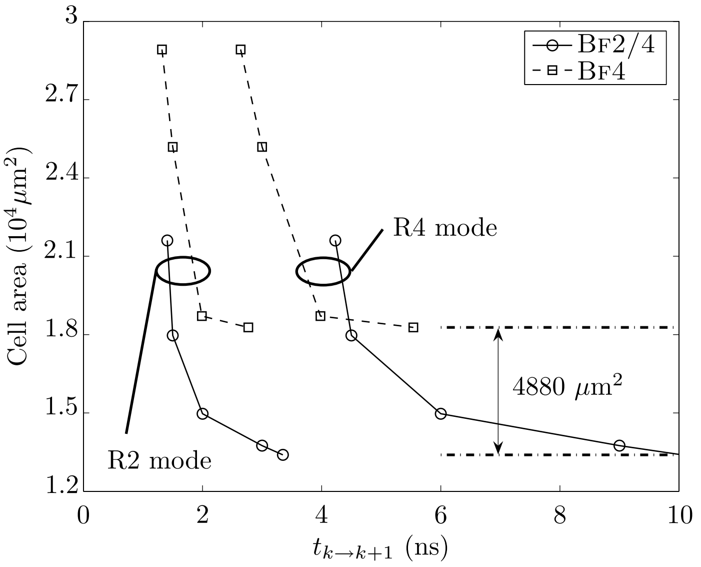


Figure III–13: Cell area versus time for a decoding stage in R2 or R4 mode for architectures based on different radices.

ture becomes somewhat larger than the Bf4 approach as the requirement on $t_{k\to k+1}$ in R4 mode becomes tighter, that is, less than about 4.5 ns. However, the provided throughput at this stage is beyond the speed requirement of considered applications, for example, the high data-rate WPANs discussed in the introduction. In the figure, this means that the actual design space to be considered is to the right hand side. Here, the Bf2/4 architecture is more suitable due to the lower area requirement of about 27% (4880 μm^2). Furthermore, this approach already provides routing resources (Perm) to support a wider range of codes, that is, the four state metrics belonging to an R4 butterfly can be shuffled by Perm in any order to maintain compatibility to the feedback connections of the basic R2 architecture. Considering R2 processing, the Bf2/4 architecture is better suited even down to a $t_{k\to k+1}$ of about 1.4 ns.

Both designs need a controller that provides control signals for MUXes and clock enable. The controller for the Bf2/4 architecture is about three times larger than the one needed when using Bf4 units. This is mostly due to the more advanced clock enabling strategy in the former design. However, since a controller is instantiated only once, it is a negligible contribution to the overall area, especially if the state space grows larger. In the R2- and R4-based designs, the controllers are always smaller than 3% of the total design size.

5 Survivor Path Unit

Some basic algorithms for SP processing, namely register-exchange (RE) and trace-back/trace-forward (TB/TF), were already discussed in Part II. Recall that L denotes the necessary decoding depth of a convolutional code after which the survivor paths are expected to have merged with sufficiently high probability. In this section, the necessary depth is determined by simulations. Furthermore, to cope with rate flexibility, specific architectural trade-offs are investigated for this unit.

5.1 Decoding Depth

Rate 1/2 codes need to be observed over a length of around five times the constraint length of the code [57]. To estimate the largest necessary L for both code rates (1/2 and 2/3), we compare TB/TF and RE approaches and their expected performance degradation for the different transmission schemes, as shown in Figure III–14. E_b/N_0 is set to the theoretical value that gives BER 10^{-5}. It is seen that both approaches do not need more than $L = 20$. The degradation of RE compared to TB/TF for smaller decoding depths is caused by using fixed-state decoding, where the decoded bit is always derived from a predefined state. This is less complex than taking a majority decision among all RE outputs and saves memory by neglecting bits connected to states that cannot reach this predefined state at step L. If one took a majority decision, the performance of RE and TB/TF is the same.

5.2 The Designed Rate-flexible Survivor Path Unit

For this design, the RE approach is chosen since the number of states is rather low. From the point of view of the additional subset signal memory needed for TCM, the least overhead is introduced since the decoding latency is by far the lowest. If $R_{\max}$ denotes the maximum rate of the TCM transmission, $L \times (R_{\max} - b)P$ extra bits are required, whereas three times more are needed for a TB/TF approach with one TF unit because its latency is three times higher. In this design $R_{\max}$ equals 5 bits per two dimensions for the 64-QAM constellation.

Additionally, for TCM a demapper has to be employed that delivers the most likely subset signal at a certain time. This is a MUX which chooses a subset signal depending on the decoded subset number from the SP unit. Recall that for convolutional decoding, b information bits are decoded every cycle. In case of TCM decoding, however, $b + 1$ must be decoded per trellis stage since the subset number consists of $b + 1$ bits. Hence, the RE algorithm must store in total $(b + 1)NL$ bits.

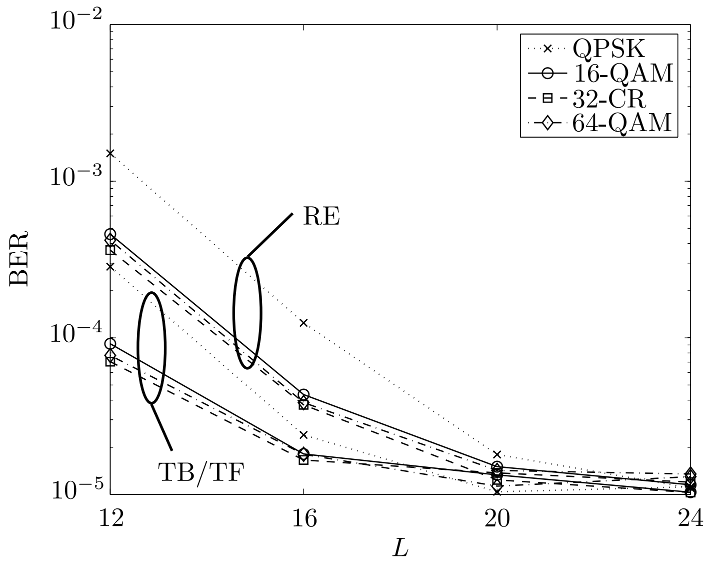


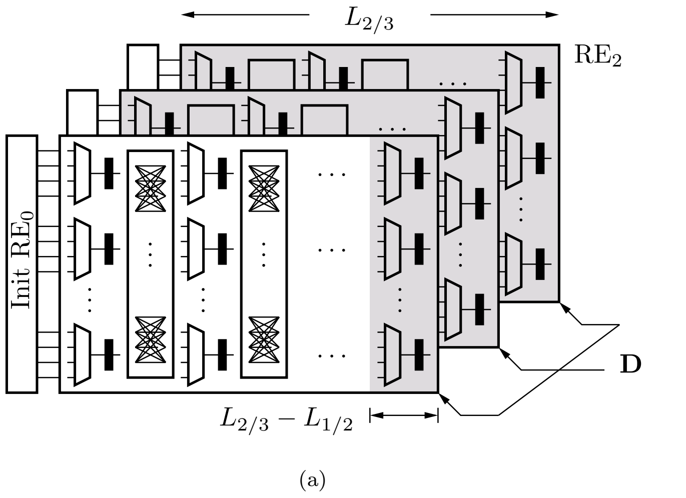
Figure III–14: BER performance for RE and TB/TF algorithms and decoding depth L. Assumed E_b/N_0 for the TCM schemes that use rate 2/3 subset selectors appears in Table III–1 in row $q = \infty$. For the Gray-mapped QPSK scheme with rate 1/2 convolutional coding, the E_b/N_0 is 5.4 dB.

To reduce memory, an alternative approach for TCM decoding only considers b information bits per RE stage as in the case of convolutional decoding. Since there are 8 subsets in our case, a subset number consists of 3 bits z_2, z_1, z_0 (Figure 2.10). The two most significant bits (MSBs) are the uncoded information bits u_2, u_1 of the systematic rate 2/3 subset selector from Figure 2.3. The RE network now provides an estimate $\hat{u}_2, \hat{u}_1$. Feeding these bits into the original subset selector yields the estimated coded bit $\hat{z}_0$. That is, the subset number is complete and one can decide the most likely subset signal.

Once there is a deviation from the correct path, a distorted sequence for the coded bits is created in the decoder, which in turn chooses wrong subset signals during the error event. Although the error event for the uncoded bits can be quite short, the resulting event for the decoded subset number becomes much longer since the encoder has to be driven back into the correct state. From examination of the encoder properties, 50% of the coded bits are expected to be wrong during this event. Simulations show that the alternative approach is quite sensitive to the SNR, which determines the number of unmerged paths at depth L that cause these error events. Hence, the decoding depth has to be increased, eventually to the point where the total number of registers is larger than in the case where $b + 1$ bits were processed per RE state and stage. For the E_b/N_0 considered, however, it turned out that L is at least 24, that is, the total number of stored bits becomes 384. This is 20% less than for the original approach with $L = 20$, which gives 480 bits in total. The latter's robustness, on the other hand, is more beneficial for this implementation.

Which radix is most beneficial for a building block in the RE network? Recall the decomposition approach from the trellis unit in Section 4, which saved arithmetic units at the cost of a slight increase in storage elements. One is tempted to apply the same approach to the R4 RE network. Again, one wants to break the R4 structure into R2 blocks. Note that in an RE network there is no arithmetic, and, contrary to the trellis unit, not only one but in total L duplicates of a trellis stage are connected in series. Per trellis stage, there are 4 additional bits to be stored per butterfly and there are $N/2^b$ butterflies per stage. This overhead is not acceptable in this implementation. Therefore, we will pursue a straightforward R4 implementation of the RE network.

An architecture for the R4 RE network is depicted in Figure III–15(a). The network is visually split into three separate slices, to represent the processing of the three survivor path bits representing a subset. Two of these slices are considered overhead for binary convolutional codes. The basic processing element of a slice consists of a 4 : 1 MUX (equals three 2 : 1 MUXes), connected to a 1-bit register. Expressed in terms of 2 : 1 MUXes, the hardware requirement for this approach is $9NL$ MUXes and $3NL$ registers. However, the network can be improved by matching it to the throughput of the trellis unit. Recall



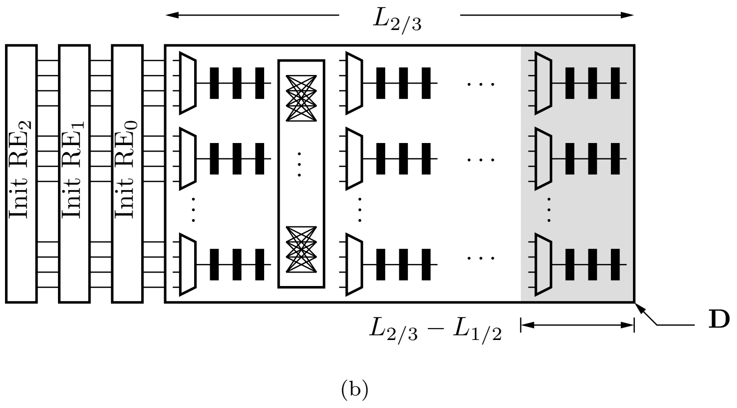


Figure III–15: Two RE architectures to suit the combined convolutional and TCM decoder. (a) shows the straightforward approach. The second architecture in (b) is matched to the throughput of the trellis unit.

that R4 processing takes 3 clock cycles and thus the RE update can also be carried out sequentially; that is, the registers can be placed in series such that three cycles are needed to update the complete survivor sequence, as in Figure III–15(b). The hardware requirement is dramatically lowered since 66% of the MUXes and interconnections become obsolete. At the same time, the utilization for both modes is effectively increased.

Were it only for R4 processing, the sequential elements could be simply realized as edge-triggered master-slave flip-flops. However, R2 processing, which allows only one cycle for the survivor path update, requires the first two registers to be bypassed. There are two solutions to the problem: either one introduces another 2 : 1 MUX in front of the third register in a stage, or the first two sequential elements in a stage are latches that are held in transparent mode. Since flip-flop-based designs have easier timing and testability, the first approach is applied.

Parts of the RE network can often be disabled since the decoding depth L of the rate 1/2 code is expected to be less than for the 2/3 code. This is indicated by the shaded parts in Figure III–15. However, following the simulations in Figure III–14, we choose $L = 24$ for both code rates to have some extra margin for varying SNR. The initial values fed into the network (Init RE$_i$) are derived from the decision bits $\mathbf{D}$ and, in case of R4 processing, state numbers.

6 Hardware Evaluation and Discussion

In this section, it is first studied how and in which parts an increased flexibility impacts the design. Then a chip implementation is presented.

Consider designs that provide up to three settings to adapt to varying channel conditions. For low SNR, rate $R = 1$ (data bits per two-dimensional channel use) is employed, which uses a rate 1/2 convolutional code together with Gray-mapped QPSK. A rate $R = 3$ mode is incorporated in the design in case there is higher SNR. In addition to the previous setup, TCM with a rate 2/3 subset selector and 16-QAM as master constellation is provided. On top of this, the third mode uses TCM with 64-QAM, thus adding $R = 5$. In the following, the different designs are named by their maximum transmission rate. Note that design ONE is fixed since it only provides $R = 1$, whereas THREE and FIVE are flexible. Note that FIVE includes THREE includes ONE, that is, there is no extra hardware needed to run a design at a lower transmission rate.

Recall from Section 4 that the architecture of the trellis unit consists of R2 PEs. Thus, the processing for the convolutional code in system ONE is one symbol per cycle. However, the other two systems need to support R2 and R4 processing. The trellis unit determines the computation rate of the whole system, which becomes one symbol per three cycles for R4 processing. Having already discussed implementation aspects for trellis and SP units, we now turn to trade-offs in the BM unit.

In Section 3 it was shown that the BM unit requires additional resources for designs THREE and FIVE because of TCM's subset decoding in combination with larger constellations. Additional hardware resources due to flexibility can be minimized by matching the rates of the processing units in the designs. This is done by interleaving the BM calculations and reusing the subset decoding units for the other subsets.

Subset decoding for 16-QAM is simply an MSB check of $y_1 \pm y_0$. This comparison unit is reused for 64-QAM, where the maximum number of boundaries is 9 according to Equation III–6 and Figure III–6(c).[†] Simulations show that removing the 4 extra horizontal/vertical comparisons needed to resolve ambiguities in 64-QAM has no noticeable effect on the overall BER.

With the given subset distribution, some slicing operations can be reused for a pair of subsets; for example, for $\mathcal{C}_4$ and $\mathcal{C}_6$ in Figure 2.8, the comparison $y_1 - y_0 < 0$ is used in both cases. For 64-QAM, there are up to three joint comparison results for the diagonal boundaries $y_1 - y_0$ (see Figure 175 in [101]). It is therefore beneficial to process such subset pairs together in one cycle. Subset decoding units are reused by translating the input symbols, here only y_0, for the subset pair in question. Two operations are required to form $y_0 \pm D_{\min}$; in the first cycle $y_0 + D_{\min}$ is processed and in the second $y_0 - D_{\min}$. Thus, in total it takes three cycles to decode ς_j and calculate λ_j for all subsets. Now the computation rate is matched to the trellis unit. Note that a latency of three cycles is introduced by this hardware sharing, which has to be accounted for in the depth of the subset signal memory.

Since the processing in the BM unit is purely feedforward, it can be easily pipelined such that the throughput of the design is determined by the feedback loop of the trellis unit. Therefore, two additional pipeline stages were introduced and the depth of the subset signal memory was adjusted accordingly. This memory could be partitioned in order to efficiently support 16-QAM and 64-QAM, which use 1 and 3 bits to specify a subset signal, respectively. The required memory width is the number of subsets times the maximum number of subset signal bits, that is, 8×3. Based on area estimates from custom memory macros, a single 24 bit wide implementation gives less overhead than three separate 8 bit wide blocks. Nevertheless, memory words are partitioned into 8 bit segments that can be accessed separately and shut down to save power. To account for all latency in the flexible designs (4 pipeline stages), a single-port

[†]From Figure III–6 it is also seen that 32-CR would need an additional two slicers, which are not used by the other two constellations, causing an overhead of 18% for the slicers. To lower the already high complexity of the BM unit, 32-CR is thus omitted in the flexible designs.

memory of size 28×24 is used. Since simultaneous read and write accesses are not allowed in single-port architectures, an additional buffer stage is required.

6.1 Impact of Flexibility

The cost of the flexible processing blocks was characterized in Sections 3–Section 5 at the architecture level. Now, three flexible designs are considered and their most important characteristics are shown in Table III–3. The numbers for the cell area in this table apply to synthesized blocks at gate level. Design constraints are chosen such that the implementation is still in the flat part of the area–delay curve, see Figure III–13 for an example of such a curve, and the resulting critical path for the designs lies in the trellis unit.

Table III–3: Three designs with different transmission rates. Power consumption for different R estimated at $V_{\text{dd}} = 1.2$ V and $f_{\text{clk}} = 250$ MHz.

	ONE	THREE	FIVE
R	1 — —	1 3 —	1 3 5
Highest mod.	QPSK Gray	16-QAM TCM	64-QAM TCM
Area (μm^2)	16289	67205	76922
BM unit	3.9%	12.6%	17.8%
Trellis unit	46.8%	19.3%	18.7%
SP unit	49.3%	52.0%	42.8%
Subset memory	—	14.6%	18.5%
Demapper	—	1.5%	2.2%
Power (mW)	4.9 — —	9.9 13.4 —	10.3 14.7 15.2

As flexibility is introduced, for example, from design ONE to THREE, the BM unit gets a larger share of the total area. In design ONE, it is negligible, whereas in design THREE, it is approaching the size of the trellis unit; in design ONE the trellis unit took half of the total size, declining to about a fifth in THREE. The growth of the BM unit is not only due to TCM; it is mainly due to the required slicing operations, which stem from larger QAM constellations and would have to be considered even for a Gray-mapped convolutional code.

The higher code rate of the subset selector for design THREE and FIVE impacts the trellis and SP units. It is seen that the dramatic decrease of the share of the trellis unit certainly justifies the R4 emulation by R2 elements. Recall that this emulation would not have made sense for the SP unit, where one has to accept the R4 processing character. Furthermore, although trellis and SP units are in theory independent of the code details, the size of the SP unit is partially influenced by TCM; the $b + 1$ bits that represent a subset number are processed per RE stage and state, instead of b bits in the case of conventional convolutional decoding. Thus, contrary to the trellis unit, the SP unit becomes a larger part at the transition from ONE to THREE, which is due to the unit's R4 processing character.

For design FIVE, the BM share grows further, while the relative cost of the trellis unit decreases slightly, despite the additional transmission rate. Task flexibility has a larger impact on the size of an implementation than rate flexibility. That is, the underlying trellis structure of a code is much easier to reuse than BM calculations that are specialized for a certain task. The observed trend should continue for larger constellations. The BM unit takes even larger portions, whereas trellis and SP units, which are principally fixed except for the growth of the wordlength, see Equation I–18, drop in percent. The parts exclusive of the TCM, subset memory and demapper, consume roughly a fifth of the cell area. Higher TCM rates (larger constellations) are expected to increase this share since more "uncoded" bits have to be stored.

Power estimation in Table III–3 is carried out with Synopsys Power Compiler on the synthesized netlists, which are back-annotated with state- and path-dependent toggle information obtained from a simulation run. There are in principle two comparison scenarios that need to be distinguished: first, convolutional decoding using either a fixed (ONE) or one of the flexible designs (THREE or FIVE) to find out how much power one has to sacrifice for a certain flexibility; and second, a comparison between designs THREE and FIVE to see how much more power has to be spent for additional transmission rates. These comparisons yield a measure of the cost of flexibility.

Not surprisingly, power consumption is sacrificed for flexibility. Scenario one indicates that from design ONE to THREE, there is twice the power spent to run the flexible design with transmission rate 1. For design FIVE, the number is slightly higher but still roughly twice the amount of power for rate 1. Comparing designs THREE and FIVE, there is a 4% and a 9.7% increase in power consumption, respectively, over rate 1 and 3 configurations. Furthermore, rate 5 mode in design FIVE only requires an extra 3.4% power, a low number considering the additional rate provided.

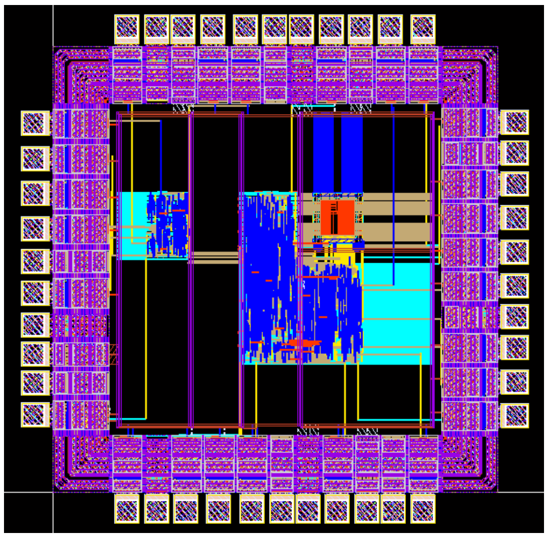


Figure III–16: Layout of the routed chip. Designs ONE and FIVE are shown on the left and right side, respectively. Row utilization is approximately 80% in both implementations.

To conclude, having accepted the initial impact of task flexibility in the TCM-designs, it makes sense to strive for more transmission rates. Therefore, the two designs ONE and FIVE will be implemented on a chip. If QPSK is used often and power consumption is a critical factor for the application at hand, it makes sense to accept the additional fixed design. Otherwise, one flexible design that covers all transmission rates, here FIVE, is sufficient.

6.2 Silicon Implementation

The complete design was modeled in VHDL at register-transfer level (RTL) and then taken through a design flow that includes Synopsys Design Compiler for synthesis and Cadence Encounter for routing. A high-speed standard cell library from Faraday for the 0.13 μm digital CMOS process from UMC is used. The RTL and gate level netlists are all verified against test vectors generated from a MATLAB fixed-point model. Post-layout timing is verified using Synopsys Prime Time with net and cell delays back-annotated in standard delay format.

Figure III–16 shows the routed chip. It is pad-limited due to test purposes and measures 1.44 mm^2. Designs ONE and FIVE are placed on the same die with separate V_{dd} to measure their power consumption independently. In TCM mode, design FIVE achieves a symbol rate of 168 Mbaud/s, a throughput of 504 Mbit/s and 840 Mbit/s using $R = 3$ and $R = 5$ configurations. Design ONE achieves a throughput of 606 Mbit/s; flexibility causes a speed penalty in that FIVE provides 504 Mbit/s in $R = 1$ mode. If WPANs are the application, all these throughputs are higher than specified in [101]. Thus, the supply voltage can be lowered to save energy. Further measurements on the fabricated chip (Figure III–17), which to date only has been functionally verified, will show how much speed has to be traded for each energy reduction.

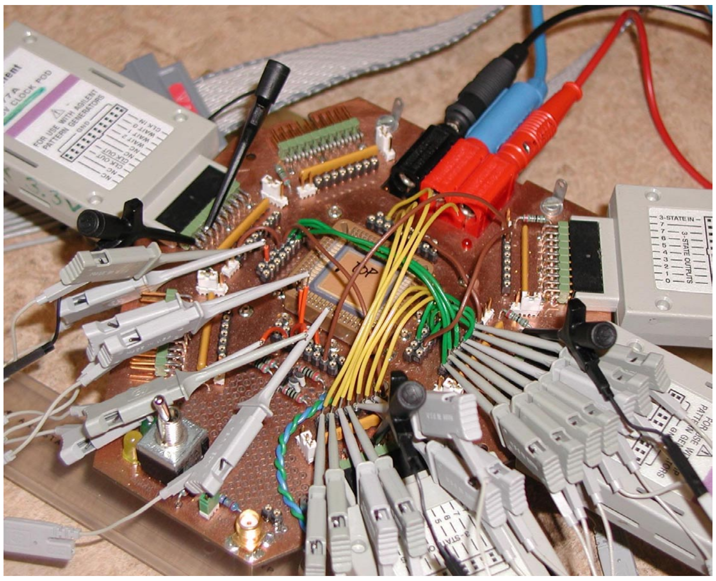


Figure III–17: The fabricated flexible Viterbi decoder chip. Its functionality was verified in the shown test environment.

6.3 An Alternative Approach

As far as throughput is concerned, the presented design is originally limited by the trellis unit, which processes an R4 butterfly in three clock cycles. It was already mentioned in Section 4 that an R4 butterfly can also be updated in one clock cycle. In the following, the implications on throughput and size of this alternative approach are estimated.

To begin with, throughput is not only improved for R4 processing but also in R2 mode since two R2 stages can be collapsed into one R4 stage (Figure 3.3), that is, two decoded bits per stage are put out. A reported speed-up for such an architecture compared to R2 processing is 1.7 [15]. This implementation is based on full-custom datapaths. A standard cell implementation only achieved a speed-up of 1.26 [37]. Hence, mode $R = 1$ provides a throughput of at least 767 Mbit/s, working at a symbol rate of 383 Mbaud/s. For $R = 3, 5$, a throughput of 1.1 and 1.9 Gbit/s is estimated, respectively.

The size of the R2 trellis unit from design ONE grows by a factor of 3.8 using the straightforward (not full-custom) 6-comparator approach for a 4-way ACS operation [37]. This trellis unit is now utilized in all designs, both fixed and flexible. However, the trellis collapsing implies that feedback connections are not directly reusable anymore. The inherent flexibility provided by PERM in BF2/4 is achieved by an additional routing stage instead, which slightly lowers the previously estimated throughputs.

To take advantage of the improved processing speed, the BM unit in the flexible (TCM) designs has to carry out subset decoding in one clock cycle, too. That is, 8 subset decoders must work in parallel, instead of 4 that work in an interleaved manner such that only half the constellation points must be stored. This is not possible in the alternative approach, and thus there is a more than twofold area increase. For the SP unit, the update also has to be carried out in parallel as in Figure III–15(a) and the size of this unit increases by roughly 2.5 times.

Based on the preceding considerations, the sizes of the different processing units can be scaled by the mentioned factors: the size of design ONE is expected to grow by a factor of 2.5, whereas the size of flexible design FIVE increases by about 2.2 times.

Recall, though, that throughput is not the major design issue our work. The envisioned applications never utilize the provided processing power. Furthermore, considering power consumption of shrinking process technologies, where static power consumption surpasses dynamic power consumption, this alternative approach becomes even less feasible compared to an R2-based architecture.

7 Conclusion

A design for a Viterbi decoder that decodes both convolutional and TCM codes to cope with varying channel conditions is presented. Sacrifices in the speed, which is not actually required, of the trellis unit result in large hardware savings for the other processing blocks by applying computation rate matching. Synthesized designs that provide different transmission rate combinations show that task flexibility inherent in the BM unit impacts the design size far more than rate flexibility of trellis and SP units. Furthermore, power estimation figures at gate-level indicate that the flexible designs become more cost-effective if provided with more than two transmission rates. To yield a quantitative cost on flexibility, a silicon implementation is crucial. The implementation was carried out in a 0.13 μm digital CMOS process, and the fabricated chip was verified for functionality. Thus, the performance of two designs, one fixed, one flexible, can be compared. For good channel SNRs, the flexible design enables a 28% higher throughput than the fixed, while it only lags by 17% when run in low SNR configuration.

Conclusion and Outlook

This thesis explores flexible architectures for VLSI implementation of channel decoding algorithms. As flexibility becomes an increasingly important performance measure in VLSI design, quantitative numbers of its cost are needed. An architecture that implements a decoder for bandwidth-efficient codes is taken as a vehicle to investigate the impact of flexibility on area, throughput, and power consumption. Time-multiplexing the trellis engine, which copes with different radices, lowered the area requirement in all parts of the design. It was concluded that one should strive for more than two transmission rates once the price for rate flexibility in the trellis unit has been paid. As a proof of concept, a VLSI circuit was implemented, fabricated, and successfully verified for functionality.

Further results obtained in this thesis concern simplifications and hybrid approaches that tackle two key parts in a Viterbi decoder. An algorithmic simplification was derived that reduces the arithmetic complexity by 17% without degrading decoding performance. In an actual implementation, this saving varied with the number of trellis states. Also, a new survivor path processing architecture is proposed, which is based on two formerly not connected approaches. This hybrid architecture shows a better latency–complexity behavior compared to competing approaches.

Much future work is needed on flexible VLSI design. A specific project for the case of trellis decoding could involve further improvements for trellis and survivor path processing blocks, which solely depend on the radix of the trellis. In particular, the survivor path unit is likely to be based on a different algorithm. Other tasks for such a generic trellis processor need to be evaluated, too. Examples could be Viterbi equalization or space–time trellis decoding. Since the branch metric calculations are highly dependent on the actual task, it is crucial to find a suitable subset of tasks which can be efficiently mapped onto hardware.

Bibliography

[1] S. M. Alamouti. A simple transmit diversity technique for wireless communications. *IEEE Journal on Selected Areas in Communications*, 16(8):1451–1458, Oct. 1998.

[2] J. B. Anderson. Limited search trellis decoding of convolutional codes. *IEEE Transactions on Information Theory*, 35(5):944–955, Sept. 1989.

[3] J. B. Anderson. Best short rate 1/2 tailbiting codes for the bit-error rate criterion. *IEEE Transactions on Communications*, 48(4):597–610, Apr. 2000.

[4] J. B. Anderson. *Digital Transmission Engineering.* IEEE Press, Piscataway, NJ, 2nd edition, 2005.

[5] J. B. Anderson and K. Balachandran. Decision depths of convolutional codes. *IEEE Transactions on Information Theory*, 35(2):455–459, Mar. 1989.

[6] J. B. Anderson and A. Svensson. *Coded Modulation Systems.* Plenum, New York, 2003.

[7] J. B. Anderson and K. E. Tepe. Properties of the tailbiting BCJR decoder. In *Codes, Systems, and Graphical Models*, IMA Volumes in Mathematics and Its Applications. Springer, Heidelberg, 2000.

[8] 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*, 20(2):284–287, Mar. 1974.

[9] G. Battail. Pondération des symboles décodés par l'algorithme de Viterbi. *Annales des Télécommunications*, 42(1–2):31–38, Jan./Feb. 1987.

[10] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming.* Princeton University Press, Princeton, NJ, 1962.

[11] C. Berrou and A. Glavieux. Near optimum error-correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications*, 44(10):1261–1271, Oct. 1996.

[12] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proceedings of IEEE International Conference on Communications*, pages 1064–1070, Geneva, Switzerland, May 1993.

[13] V. Betz. FPGAs and structured ASICs: Overview and research challenges. [Online]. http://www.iic.umanitoba.ca/docs/vaughn-betz.ppt, Oct. 2006.

[14] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R.-H. Yan. A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-μm CMOS. *IEEE Journal of Solid-State Circuits*, 37(11):1555–1564, Nov. 2002.

[15] P. J. Black and T. H.-Y. Meng. A 140-Mb/s, 32-state, radix-4 Viterbi decoder. *IEEE Journal of Solid-State Circuits*, 27(12):1877–1885, Dec. 1992.

[16] P. J. Black and T. H.-Y. Meng. Hybrid survivor path architectures for Viterbi decoders. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 433–436, Minneapolis, MN, Apr. 1993.

[17] P. J. Black and T. H.-Y. Meng. A 1-Gb/s, four-state, sliding block Viterbi decoder. *IEEE Journal of Solid-State Circuits*, 32(6):797–805, June 1997.

[18] H. Blume, H. T. Feldkämper, and T. G. Noll. Model-based exploration of the design space for heterogeneous systems on chip. *The Journal of VLSI Signal Processing*, 40(1):19–34, May 2005.

[19] M. Bóo, F. Argüello, J. D. Bruguera, R. Doallo, and E. L. Zapata. High-performance VLSI architecture for the Viterbi algorithm. *IEEE Transactions on Communications*, 45(2):168–176, Feb. 1997.

[20] E. Boutillon and N. Demassieux. A generalized precompiling scheme for surviving path memory management in Viterbi decoders. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1579–1582, Chicago, IL, May 1993.

[21] E. Boutillon, W. J. Gross, and P. G. Gulak. VLSI architectures for the MAP algorithm. *IEEE Transactions on Communications*, 51(2):175–185, Feb. 2003.

[22] J. B. Cain, G. C. Clark, Jr., and J. M. Geist. Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding. *IEEE Transactions on Information Theory*, 25(1):97–100, Jan. 1979.

[23] J. R. Cavallaro and M. Vaya. Viturbo: A reconfigurable architecture for Viterbi and turbo decoding. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 497–500, Hong Kong, Apr. 2003.

[24] K. Chadha and J. R. Cavallaro. A reconfigurable Viterbi decoder architecture. In *Proceedings of Asilomar Conference on Signals, Systems, and Computers*, pages 66–71, Pacific Grove, CA, Nov. 2001.

[25] A. P. Chandrakasan and R. W. Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, Apr. 1995.

[26] Y.-N. Chang, H. Suzuki, and K. K. Parhi. A 2-Mb/s 256-state 10-mW rate-1/3 Viterbi decoder. *IEEE Journal of Solid-State Circuits*, 35(6):826–834, June 2000.

[27] J. Chen and R. M. Tanner. A hybrid coding scheme for the Gilbert–Elliott channel. *IEEE Transactions on Communications*, 54(10):1787–1796, Oct. 2006.

[28] H.-M. Choi, J.-H. Kim, and I.-C. Park. Low-power hybrid turbo decoding based on reverse calculation. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 2053–2056, Island of Kos, Greece, May 2006.

[29] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker. Applications of error-control coding. *IEEE Transactions on Information Theory*, 44(6):2531–2560, Oct. 1998.

[30] R. Cypher and C. B. Shung. Generalized trace back techniques for survivor memory management in the Viterbi algorithm. In *Proceedings of IEEE Global Telecommunications Conference*, pages 1318–1322, San Diego, CA, Dec. 1990.

[31] H. Dawid. *Algorithmen und Schaltungsarchitekturen zur Maximum a Posteriori Faltungsdecodierung*. PhD thesis, RWTH Aachen, Mar. 1996.

[32] H. Dawid, S. Bitterlich, and H. Meyr. Trellis pipeline-interleaving: A novel method for efficient Viterbi decoder implementation. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1875–1878, San Diego, CA, May 1992.

[33] H. Dawid, G. Fettweis, and H. Meyr. A CMOS IC for Gb/s Viterbi decoding: System design and VLSI implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(1):17–31, Mar. 1996.

[34] Z. Ding and S. Lin. Channel equalization and error correction for high rate Wireless Personal Area Networks. Technical Report MICRO 01-029, Dept. Electrical and Computer Engineering, University of California, Davis, 2001.

[35] P. Elias. Coding for noisy channels. *IRE Convention Record*, 2, pt. 4:37–47, Mar. 1955.

[36] R. M. Fano. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, 9(2):64–76, Apr. 1963.

[37] H. T. Feldkämper, H. Blume, and T. G. Noll. Study of heterogeneous and reconfigurable architectures in the communication domain. *Advances in Radio Science—Kleinheubacher Berichte*, 1:165–169, May 2003.

[38] G. Fettweis. Parallel Viterbi algorithm implementation: Breaking the ACS bottleneck. *IEEE Transactions on Communications*, 37(8):785–790, Aug. 1989.

[39] G. Fettweis. Algebraic survivor memory management for Viterbi detectors. *IEEE Transactions on Communications*, 43(9):2458–2463, Sept. 1995.

[40] G. Fettweis and H. Meyr. High-speed parallel Viterbi decoding: Algorithm and VLSI-architecture. *IEEE Communications Magazine*, 29(5):46–55, May 1991.

[41] G. Feygin and P. G. Gulak. Architectural tradeoffs for survivor sequence memory management in Viterbi decoders. *IEEE Transactions on Communications*, 41(3):425–429, Mar. 1993.

[42] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, Apr. 1985.

[43] G. D. Forney, Jr. Review of random tree codes. Final Report Appendix A Contract NAS2-3737, NASA CR 73176, NASA Ames Research Center, Dec. 1967.

[44] G. D. Forney, Jr. Convolutional codes I: Algebraic structure. *IEEE Transactions on Information Theory*, 16(6):720–738, Nov. 1970.

[45] G. D. Forney, Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, Mar. 1973.

[46] G. D. Forney, Jr. and G. Ungerböck. Modulation and coding for linear Gaussian channels. *IEEE Transactions on Information Theory*, 44(6):2384–2415, Oct. 1998.

[47] G. J. Foschini. Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. *Bell Labs Technical Journal*, 1(2):41–59, Autumn 1996.

[48] M. P. Fossorier and S. Lin. Differential trellis decoding of convolutional codes. *IEEE Transactions on Information Theory*, 46(3):1046–1053, May 2000.

[49] R. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, 8(1):21–28, Jan. 1962.

[50] Y. Gang, A. T. Erdogan, and T. Arslan. An efficient pre-traceback architecture for the Viterbi decoder targeting wireless communication applications. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 53(9):1918–1927, Sept. 2006.

[51] T. Gemmeke, M. Gansen, and T. G. Noll. Implementation of scalable power and area efficient high-throughput Viterbi decoders. *IEEE Journal of Solid-State Circuits*, 37(7):941–948, July 2002.

[52] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their applications. *IEEE Transactions on Communications*, 36(4):389–400, Apr. 1988.

[53] J. Hagenauer and P. Höher. A Viterbi algorithm with soft-decision outputs and its applications. In *Proceedings of IEEE Global Telecommunications Conference*, pages 1680–1686, Dallas, TX, Nov. 1989.

[54] J. H. Han, A. T. Erdogan, and T. Arslan. A power efficient reconfigurable max-log-MAP turbo decoder for wireless communication systems. In *Proceedings of IEEE International Symposium on System-on-Chip*, pages 247–250, Tampere, Finland, Nov. 2005.

[55] J.-S. Han, T.-J. Kim, and C. Lee. High performance Viterbi decoder using modified register-exchange methods. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 553–556, Vancouver, Canada, May 2004.

[56] A. P. Hekstra. An alternative to metric rescaling in Viterbi decoders. *IEEE Transactions on Communications*, 37(11):1220–1222, Nov. 1989.

[57] J. A. Heller and I. M. Jacobs. Viterbi decoding for satellite and space communication. *IEEE Transactions on Communications*, 19(5):835–848, Oct. 1971.

[58] D. E. Hocevar and A. Gatherer. Achieving flexibility in a Viterbi decoder DSP coprocessor. In *Proceedings of IEEE Vehicular Technology Conference*, pages 2257–2264, Boston, MA, Sept. 2000.

[59] F. Jelinek. A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13(3):675–685, Nov. 1969.

[60] O. Joeressen, M. Vaupel, and H. Meyr. High-speed VLSI architectures for soft-output Viterbi decoding. In *Proceedings of IEEE International Conference on Application-Specific Array Processors*, pages 373–384, Berkeley, CA, Aug. 1992.

[61] R. Johannesson and K. S. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Press, Piscataway, NJ, 1999.

[62] M. Kamuf, J. B. Anderson, and V. Öwall. A simplified computational kernel for trellis-based decoding. *IEEE Communications Letters*, 8(3):156–158, Mar. 2004.

[63] M. Kamuf, V. Öwall, and J. B. Anderson. Area and power efficient trellis computational blocks in 0.13 μm CMOS. In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 344–347, Kobe, Japan, May 2005.

[64] I. Kang and A. N. Willson, Jr. Low-power Viterbi decoder for CDMA mobile terminals. *IEEE Journal of Solid-State Circuits*, 33(3):473–482, Mar. 1998.

[65] J. Karaoğuz. High-rate wireless personal area networks. *IEEE Communications Magazine*, 39(12):96–102, Dec. 2001.

[66] L.-N. Lee. On optimal soft-decision demodulation. *IEEE Transactions on Information Theory*, 22(4):437–444, July 1976.

[67] C.-C. Lin, Y.-H. Shih, H.-C. Chang, and C.-Y. Lee. Design of a power-reduction Viterbi decoder for WLAN applications. *IEEE Transactions on Circuits and Systems—Part I: Regular Papers*, 52(6):1148–1156, June 2005.

[68] H.-L. Lou. Implementing the Viterbi algorithm. *IEEE Signal Processing Magazine*, 12(5):42–52, Sept. 1995.

[69] H.-L. Lou, P. Tong, and J. M. Cioffi. A programmable codec design for trellis coded modulation. In *Proceedings of IEEE Global Telecommunications Conference*, pages 944–947, Phoenix, TX, Nov. 1997.

[70] J. L. Massey. *Threshold Decoding.* MIT Press, Cambridge, MA, 1963.

[71] G. J. Minty. A comment on the shortest-route problem. *Operations Research*, 5(5):724, Oct. 1957.

[72] T. Miyauchi, K. Yamamoto, T. Yokokawa, M. Kan, Y. Mizutani, and M. Hattori. High-performance programmable SISO decoder VLSI implementation for decoding turbo codes. In *Proceedings of IEEE Global Telecommunications Conference*, pages 305–309, San Antonio, TX, Nov. 2001.

[73] J. K. Omura. On the Viterbi decoding algorithm. *IEEE Transactions on Information Theory*, 15(1):177–179, Jan. 1969.

[74] I. M. Onyszchuk, K.-M. Cheung, and O. Collins. Quantization loss in convolutional decoding. *IEEE Transactions on Communications*, 41(2):261–265, Feb. 1993.

[75] E. Paaske, S. Pedersen, and J. Sparsø. An area-efficient path memory structure for VLSI implementation of high speed Viterbi decoders. *INTEGRATION, the VLSI journal*, 12(1):79–91, Nov. 1991.

[76] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford University Press, New York, 2000.

[77] K. K. Parhi. *VLSI Digital Signal Processing Systems.* Wiley, New York, 1999.

[78] J. G. Proakis. *Digital Communications.* McGraw-Hill, New York, 4th edition, 2001.

[79] J. M. Rabaey, A. Chandrakasan, and B. Nikolić. *Digital Integrated Circuits.* Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2003.

[80] P. Robertson, E. Villebrun, and P. Höher. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. In *Proceedings of IEEE International Conference on Communications*, pages 1009–1013, Seattle, WA, June 1995.

[81] P. Robertson and T. Wörz. Bandwidth-efficient turbo trellis-coded modulation using punctured component codes. *IEEE Journal on Selected Areas in Communications*, 16(2):206–218, Feb. 1998.

[82] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming: Series A and B*, 66(2):181–199, Sept. 1994.

[83] C. Schurgers, F. Catthoor, and M. Engels. Memory optimization of MAP turbo decoder algorithms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(2):305–312, Apr. 2001.

[84] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 & 623–656, July & Oct. 1948.

[85] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, Jan. 1949.

[86] C. B. Shung, H.-D. Lin, R. Cypher, P. H. Siegel, and H. K. Thapar. Area-efficient architectures for the Viterbi algorithm—Part I: Theory. *IEEE Transactions on Communications*, 41(4):636–644, Apr. 1993.

[87] C. B. Shung, H.-D. Lin, R. Cypher, P. H. Siegel, and H. K. Thapar. Area-efficient architectures for the Viterbi algorithm—Part II: Applications. *IEEE Transactions on Communications*, 41(5):802–807, May 1993.

[88] C. B. Shung, P. H. Siegel, G. Ungerböck, and H. K. Thapar. VLSI architectures for metric normalization in the Viterbi algorithm. In *Proceedings of IEEE International Conference on Communications*, pages 1723–1728, Atlanta, GA, Apr. 1990.

[89] J. Simpson and E. Weiner, editors. *The Oxford English Dictionary.* Clarendon Press, Oxford, 2nd edition, 1989.

[90] Software defined radio forum. [Online]. http://www.sdrforum.org, Feb. 2007.

[91] V. Tarokh, H. Jafarkhani, and A. R. Calderbank. Space-time codes for high data rate wireless communication: Performance criterion and code construction. *IEEE Transactions on Information Theory*, 44(2):744–765, Mar. 1998.

[92] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Göckel, and W. Burleson. A reconfigurable, power-efficient adaptive Viterbi decoder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(4):484–488, Apr. 2005.

[93] H. K. Thapar and J. M. Cioffi. A block processing method for designing high-speed Viterbi detectors. In *Proceedings of IEEE International Conference on Communications*, pages 1096–1100, Atlanta, GA, Apr. 1990.

[94] F. Tosato and P. Bisaglia. Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2. In *Proceedings of IEEE International Conference on Communications*, pages 664–668, New York, Apr./May 2002.

[95] G. Ungerböck. Channel coding with multilevel/phase signals. *IEEE Transactions on Information Theory*, 28(1):55–67, Jan. 1982.

[96] G. Ungerböck. Trellis-coded modulation with redundant signal sets. *IEEE Communications Magazine*, 25(2):5–21, Feb. 1987.

[97] L. Van der Perre, B. Bougard, J. Craninckx, W. Dehaene, L. Hollevoet, M. Jayapala, P. Marchal, M. Miranda, P. Raghavan, T. Schuster, P. Wambacq, F. Catthoor, and P. Vanbekbergen. Architectures and circuits for software-defined radios: Scaling and scalability for low cost and low energy. In *Digest of Technical Papers IEEE International Solid-State Circuits Conference*, pages 568–569, San Francisco, Feb. 2007.

[98] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, Apr. 1967.

[99] A. J. Viterbi. A pragmatic approach to trellis-coded modulation. *IEEE Communications Magazine*, 27(7):11–19, July 1989.

[100] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* IEEE Standard 802.11a, 1999.

[101] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs).* IEEE Standard 802.15.3, 2003.

[102] J. K. Wolf and E. Zehavi. P^2 codes: Pragmatic trellis codes utilizing punctured convolutional codes. *IEEE Communications Magazine*, 33(2):94–99, Feb. 1995.

[103] J. M. Wozencraft. Sequential decoding for reliable communication. *IRE Convention Record*, 5, pt. 2:11–25, Jan. 1957.

[104] Y. Yasuda, K. Kashiki, and Y. Hirata. High-rate punctured convolutional codes for soft decision Viterbi decoding. *IEEE Transactions on Communications*, 32(3):315–319, Mar. 1984.

[105] A. K. Yeung and J. M. Rabaey. A 210-Mb/s radix-4 bit-level pipelined Viterbi decoder. In *Digest of Technical Papers IEEE International Solid-State Circuits Conference*, pages 88–92, San Francisco, CA, Feb. 1995.

[106] N. Zhang and R. W. Brodersen. Architectural evaluation of flexible digital signal processing for wireless receivers. In *Proceedings of Asilomar Conference on Signals, Systems, and Computers*, pages 78–83, Pacific Grove, CA, Oct./Nov. 2000.

[107] Y. Zhu and M. Benaissa. Reconfigurable Viterbi decoding using a new ACS pipelining technique. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 360–368, The Hague, The Netherlands, June 2003.