Chapter 17

Viterbi Decoders: High Performance Algorithms and Architectures

Herbert Dawid
Synopsys, Inc.
DSP Solutions Group
Kaiserstr. 100
D-52134 Herzogenrath
Germany
dawid@synopsys.com

Olaf J. Joeressen
Nokia Mobile Phones
R&D Center Germany
Meesmannstr. 103
D-44807 Bochum
Germany
Olaf.Joeressen
@nmp.nokia.com

Heinrich Meyr
Aachen University
of Technology
ISS –611 810–
D–52056 Aachen
Germany
meyr@ert.
rwth-aachen.de

17.1 Introduction

Viterbi Decoders (VDs) are today widely used as forward error correction (FEC) devices in many digital communications and multimedia products, including mobile (cellular) phones, video and audio broadcasting receivers, and modems. VDs are implementations of the *Viterbi Algorithm* (VA) used for decoding *convolutional* or *trellis codes*[1].

The continuing success of convolutional and trellis codes for FEC applications in almost all modern digital communication and multimedia products is based on three main factors:

- The existence of an optimum *Maximum Likelihood decoding* algorithm – the VA – with limited complexity, which is well suited for implementation.
- The existence of classes of good (convolutional and trellis) codes suited for many different applications.

[1] Other important applications of the VA are e.g. equalization for transmission channels with memory like multipath-fading channels and numerous applications apart from digital communications like pattern, text and speech recognition as well as magnetic recording. Due to lack of space, only Viterbi decoding is considered here.

- The advances in digital silicon technology which make the implementation of the VA possible even for sophisticated codes and high bit rate applications.

The coarse system level block diagram shown in Fig. 17.1 illustrates the use of VDs in digital communication systems. The well known discrete time model with the discrete time index k is used here in order to model transmitter, channel and receiver.

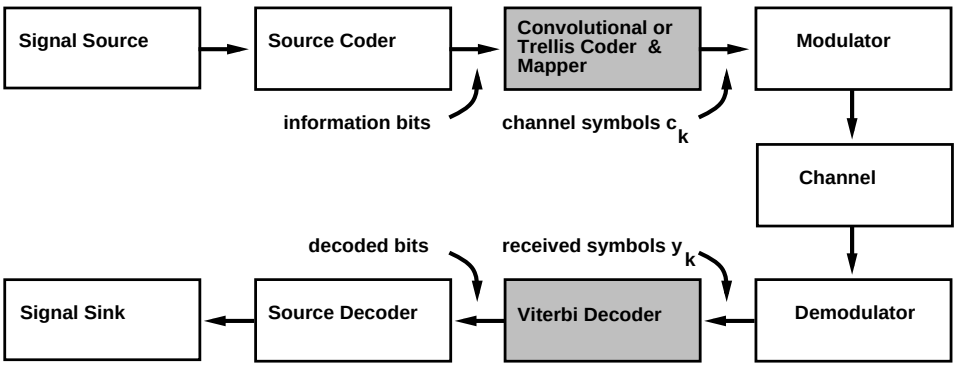

Figure 17.1 Viterbi Decoders in digital communication systems.

The signals emitted by the signal source are first compressed in a source encoder (e.g. a speech, audio or video encoder). The compressed information bits then enter a convolutional encoder or trellis encoder, which introduces *channel coding*. While the source encoder removes redundant and irrelevant information from the source signal in order to reduce the transmission rate, the channel encoder deliberately introduces the redundancy necessary to combat transmission impairments by forward error correction (FEC). Coded symbols from a predefined and sometimes multidimensional symbol alphabet are generated by the encoder and mapped onto complex channel symbols which enter the modulator. Here, the signal is modulated according to a chosen modulation scheme and carrier frequency. After transmission over the channel, the received signal is first demodulated. Following, the demodulated received symbols enter the VD. The soft (quantized) channel symbols available in the demodulator can very advantageously be used by the VD. Hence, rather than generating hard symbol decisions, the demodulator delivers a *soft decision* input to the VD[2]. The corrected information bits are finally decompressed in the source decoder.

17.1.1 Viterbi Decoding Applications

Among the numerous applications of Viterbi decoding, we consider three areas to be most important. The completely different characteristics of these applications emphasize the widespread use of Viterbi decoders in almost all modern telecommunication standards.

[2]Soft decision Viterbi decoding leads to an increase in coding gain of about 2dB compared to hard decision Viterbi decoding[9].

1. Mobile (Cellular) Phones:
   For mobile or cellular phone applications, the transmission channels are subject to various impairments like fading. Channel coding is essential in order to obtain the desired transmission quality. Many communication standards like the Global System for Mobile Communications (GSM) standard, the IS–54 digital cellular phone standard and the IS–95 CDMA standard specify the use of convolutional codes.

2. Video and Audio Broadcasting Receivers:
   Channel coding using convolutional codes is essential for almost all satellite communication standards, among them the recent Digital Video Broadcasting (DVB-S) standard for satellite transmission [8], the DSS standard as well as the terrestrial Digital Audio Broadcasting (DAB) standard.

3. Modems:
   Modems represent an application area where very advanced channel coding techniques are used. Viterbi decoding for convolutional and trellis coded modulation codes is e.g. employed in modems according to the recent 32kbps and 54 kbps modem standards. Software DSP implementations are commonly used due to the relatively low bit rate.

For low bit rate applications, Viterbi decoding is implemented in software on digital signal processors (DSPs). The bit rate required by modern high quality speech transmission represents the current limit for VD software implementations due to the high computational requirements imposed by the VA. Hybrid DSP architectures were developed with special datapaths supporting the particular VA processing requirements. We focus here on higher bit rate applications, where the VD is implemented in Very Large Scale Integration (VLSI) technology as a separate hardware unit.

17.2 The Viterbi algorithm

In order to introduce the VA [1, 2] and the used notation, we exemplarily discuss the simple convolutional encoder shown in Fig. 17.2.

The input stream of information bits is mapped onto k-bit *information symbols* u_k, which are input to a finite state machine (FSM) generating $n > k$ coded bits from the information symbols. The ratio k/n (here $1/2$) is called the *code rate*. The larger the code rate, the smaller the amount of redundancy introduced by the coder. With $k = 1$, only code rates $1/n$ are possible. Higher rate codes are known for $k > 1$. Alternatively, higher rate codes can be created by using a $1/n$ *base* or *mother code* and omitting (puncturing) a part of the coded bits after encoding as specified by a given puncturing pattern or puncture mask [12, 13, 14]. It is shown in [12, 13] that the resulting *punctured codes* lead to reduced decoding complexity compared to standard codes with the same code rate and $k > 1$ at negligible performance losses. Today, $k = 1$ holds for virtually all practically relevant base codes [6], therefore we consider only this case.

The n coded bits $b_{i,k}$ with $i \in \{1, \ldots, n\}$ represent the *code symbols* $b_k = \sum_{j=1}^{n} b_{j,k} \cdot 2^{j-1}$ of a given symbol alphabet: $b_k \in \{0, \ldots, 2^n - 1\}$. If the encoder FSM has a memory of ν bits, the code symbols are calculated from $K = \nu + 1$ bits, the FSM memory and the current input bit, respectively. K is called the *constraint*

length of the code. The kth encoder state can be conveniently written as an integer number:

$$x_k = \sum_{j=0}^{\nu-1} x_{j,k}\ 2^j \quad \text{with} \quad x_k \in \{0,\ldots,2^\nu - 1\}; \quad x_{j,k} \in \{0,1\} \tag{1}$$

Virtually all commonly used convolutional coders exhibit a feedforward shift register structure. Additionally, in contrast to *systematic* codes, where the sequence of input symbols appears unchanged at the output together with the added redundancy, these convolutional codes are *nonsystematic* codes (NSCs). The coder is described by a convolution of the sequence of input bits with polynomials G_i over GF(2)

$$b_{i,k} = \sum_{j=0}^{\nu} g_{i,j} \cdot u_{k-j}; \qquad G_i = \sum_{j=0}^{\nu} g_{i,j} \cdot 2^j \tag{2}$$

The *generator polynomials* G_i are of degree ν and are usually not written as polynomials, but as numbers in octal notation as shown in Eq. (2). Here, $g_{i,j}$ are the binary coefficients of the generator polynomial G_i. For the rate 1/2, $\nu = 2$ coder in Fig. 17.2, the generator polynomials are $G_0 = 7|_{\text{octal}} = 111|_{\text{binary}}$ and $G_1 = 5|_{\text{octal}} = 101|_{\text{binary}}$. Therefore, the structure of the encoder as shown in Fig. 17.2 results[3].

The code symbols generated by the encoder are subsequently mapped onto complex valued *channel symbols* according to a given modulation scheme and a predefined mapping function. In general, the channel symbols $\boldsymbol{c_k}$ are tuples of complex valued symbols. As an example, in Fig. 17.2, the symbol constellation according to *BPSK* (binary phase shift keying) is shown. Here, each code symbol is mapped onto a tuple of two successive BPSK symbols.

The concatenation of modulator, channel and demodulator as shown in Fig. 17.1 is modeled by adding (complex valued) white noise $\boldsymbol{n_k}$ to the channel symbols $\boldsymbol{c_k}$[4]. Hence, for the received symbols $\boldsymbol{y_k}$

$$\boldsymbol{y_k} = \boldsymbol{c_k} + \boldsymbol{n_k} \tag{3}$$

holds. This model is adequate for a number of transmission channels e.g. in satellite and deep space communication. Even if a given transmission channel can not be described by additive white noise (e.g. in the case of fading channels), theory [7] shows that the optimum demodulator or *inner receiver* has to be designed in a way that the concatenation of modulator, channel and demodulator appears again as an additive white noise channel. Sometimes, if successive demodulator outputs are correlated (e.g. if equalization is employed in the demodulator or if noise bursts occur), an *interleaver* is introduced in the transmitter at the coder output and the

[3] If not all k bits of the information symbols u_k enter the coder, parallel state transitions occur in the trellis: The parallel transitions are independent of the bypassed bits. Hence, a symbol-by-symbol decision has to be implemented in the receiver for the bypassed bits. This situation can be found in trellis encoders for trellis coded modulation (TCM) [10] codes.

[4] Note that, below, bold letters denote complex valued numbers, and capital letters denote sequences of values in mathematical expressions.

corresponding deinterleaver is introduced prior to Viterbi decoding. Interleaving reduces the correlations between successive demodulator outputs.

The behavior of the encoder is illustrated by drawing the state transitions of the FSM over time, as shown in Fig. 17.2. The resulting structure, the *trellis diagram* or just *trellis*, is used by the Viterbi decoder to find the most likely sequence of information symbols (indicated by the thick lines in Fig. 17.2) given the received symbols $\boldsymbol{y_k}$. In the trellis, all possible encoder states are drawn as nodes, and the possible state transitions are represented by lines connecting the nodes. Given the initial state of the encoder FSM, there exists a one-to-one correspondence of the FSM state sequence to the sequence of information symbols $U = \{u_k\}$ with $k \in \{0, \ldots, T-1\}$.

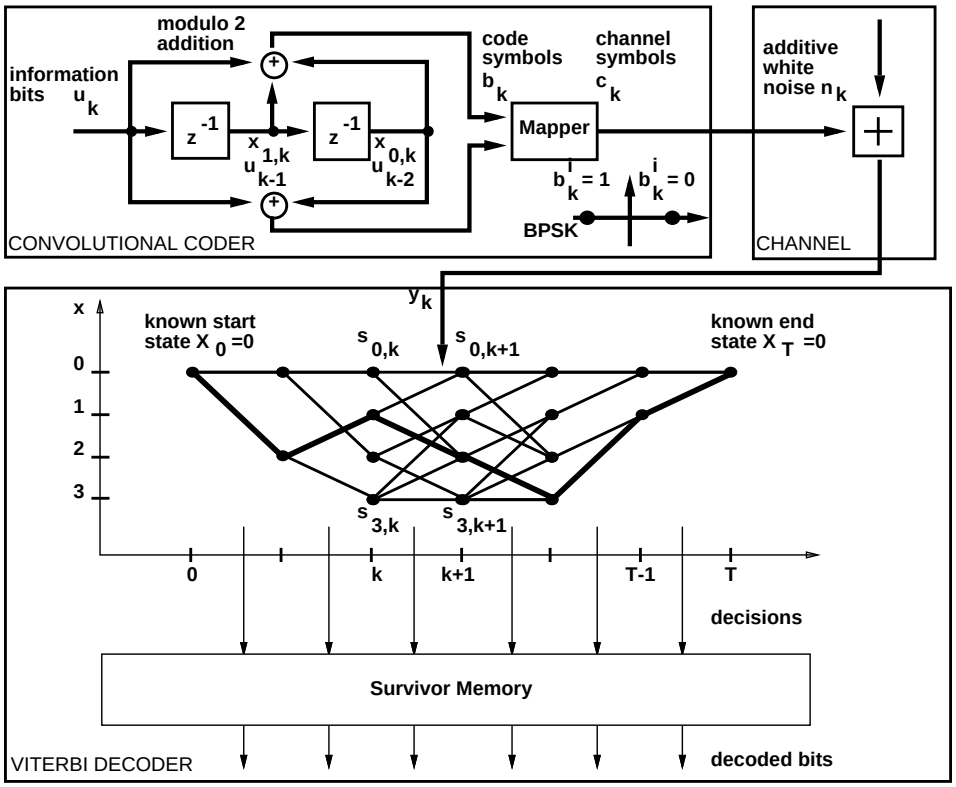

Figure 17.2 Convolutional Coder and Trellis diagram.

The number of trellis states is $N = 2^{\nu} = 2^{K-1}$ and the number of branches merging into one state is called M, where $M = 2^k$ is equal to the number of possible information symbols u_k. For binary symbols, $M = 2$ holds as shown in Fig. 17.2. The trellis nodes representing state $x_k = i$ at time k are denoted as $s_{i,k}$.

A possible state transition is a branch in the trellis, and a possible state sequence represents a path through the trellis.

In order to optimally retrieve the transmitted information one searches for the channel symbol sequence $\hat{\boldsymbol{C}}$ which has most likely generated the received symbol

sequence $\boldsymbol{Y}$. This approach is called Maximum Likelihood Sequence Estimation (MLSE). Mathematically, this can be stated as follows. Given the received sequence $\boldsymbol{Y}$ one sequence $\hat{\boldsymbol{C}}$ is searched which maximizes the value of the *likelihood function* $P(\boldsymbol{Y}|\boldsymbol{C})$:

$$\hat{\boldsymbol{C}} = \arg\{\max_{\text{all sequences } \boldsymbol{C}} P(\boldsymbol{Y}|\boldsymbol{C})\} \tag{4}$$

Since the noise samples in Eq. (4) are statistically independent and the underlying shift register process is a Markov process, the sequence likelihood function can be factorized [2]:

$$P(\boldsymbol{Y}|\boldsymbol{C}) = \prod_{k=0}^{T-1} P(\boldsymbol{y_k}|\boldsymbol{c_k}) \tag{5}$$

Here, $P(\boldsymbol{y_k}|\boldsymbol{c_k})$ is the conditional *probability density function* (PDF) of one received sample $\boldsymbol{y_k}$ given $\boldsymbol{c_k}$. In order to express $P(\boldsymbol{y_k}|\boldsymbol{c_k})$, the PDF of the noise has to be known.
Since the logarithm is a monotonic function, we can equally well maximize:

$$\hat{\boldsymbol{C}} = \arg\{\max_{\text{all sequences } \boldsymbol{C}} \sum_{k=0}^{T-1} \log(P(\boldsymbol{y_k}|\boldsymbol{c_k}))\} \tag{6}$$

The log–likelihood function $\log(P(\boldsymbol{y_k}|\boldsymbol{c_k}))$ is given the name *branch metric* or *transition metric*[5].

We recall that to every branch in the trellis (see Fig. 17.2) there corresponds exactly one tuple of channel symbols $\boldsymbol{c_k}$. We therefore assign a branch metric $\lambda_k^{(m,i)}$ to every branch in the trellis. $\lambda_k^{(m,i)}$ denotes the branch metric of the m-th branch leading to trellis state $s_{i,k}$, which is equal to the encoder state $x_k = i$. Instead of using $\lambda_k^{(m,i)}$, which expresses the branch metric as a function of the branch label m and the current state $x_k = i$, it is sometimes more convenient to use $\lambda_{ij,k}$, which denotes the branch metric of the branch from trellis state $s_{j,k}$ to trellis state $s_{i,k+1}$. The unit calculating all possible branch metrics in a Viterbi decoder is called *transition metric unit* (TMU).

As an important example we consider zero mean complex valued additive white Gaussian noise (AWGN) with uncorrelated inphase and quadrature components and channel symbols $\boldsymbol{c_k}$ consisting of a single complex value. We obtain for the branch metric:

$$P(\boldsymbol{y_k}|\boldsymbol{c_k}) = \frac{1}{\pi\sigma^2} \exp \frac{|\boldsymbol{y_k} - \boldsymbol{c_k}|^2}{\sigma^2} \tag{7}$$

and

$$\log(P(\boldsymbol{y_k}|\boldsymbol{c_k})) \sim |\boldsymbol{y_k} - \boldsymbol{c_k}|^2 \tag{8}$$

where σ^2 is the variance of the complex valued gaussian random variable $\boldsymbol{n_k}$. From Eq. (8) we observe the important fact that the branch metric is proportional to the *Euclidean distance* between the received symbol $\boldsymbol{y_k}$ and the channel symbol $\boldsymbol{c_k}$. The sum in Eq. (6) represents the accumulation of the branch metrics along a given path through the trellis according to the sequence $\boldsymbol{C}$. It is called *path metric*. The

[5] The advantage of using the logarithm for the branch metrics will soon become apparent.

path metric for a path leading to state $s_{i,k}$ is called $\gamma_k^{(m,i)}$, where $m \in \{0, \ldots, M-1\}$ denotes the path label of one of the M paths leading to state $s_{i,k}$.

Conceptually, the most likely sequence $\hat{\boldsymbol{C}}$ can be found by an exhaustive search as follows. We compute the path metric for every possible sequence $\boldsymbol{C}$, hence for every possible path through the trellis. The maximum likelihood path, which is the path with the smallest Euclidean distance, corresponds to $\hat{\boldsymbol{C}}$:

$$\hat{\boldsymbol{C}} = \arg\{\min_{\text{all sequences } \boldsymbol{C}} \sum_{k=0}^{T-1} |\boldsymbol{y_k} - \boldsymbol{c_k}|^2\} \tag{9}$$

Hence, maximizing the log–likelihood function as in Eq. (6) is equivalent to minimizing the Euclidean distance as in Eq. (9).

Since the number of paths increases exponentially as a function of the length of the sequence, the computational effort then also increases exponentially. Fortunately, there exists a much more clever solution to the problem which carries the name of its inventor, the Viterbi algorithm [1]. When using the VA, the computational effort increases only linearly with the length of the trellis, hence the computational effort per transmitted bit is constant.

The VA recursively solves the problem of finding the most likely path by using a fundamental principle of optimality first introduced by Bellman [5] which we cite here for reference:

The Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

In the present context of Viterbi decoding, we make use of this principle as follows. If we start accumulating branch metrics along the paths through the trellis, the following observation holds: Whenever two paths merge in one state, only the most likely path (the best path or the *survivor path*) needs to be retained, since for all possible extensions to these paths, the path which is currently better will always stay better: For any given extension to the paths, both paths are extended by the same branch metrics. This process is described by the *add-compare-select* (ACS) recursion: The path with the best path metric leading to every state is determined recursively for every step in the trellis. The metrics for the survivor paths for state $x_k = i$ at trellis step k are called *state metrics* $\gamma_{i,k}$ below.

In order to determine the state metric $\gamma_{i,k}$, we calculate the path metrics for the paths leading to state $x_k = i$ by adding the state metrics of the predecessor states and the corresponding branch metrics. The predecessor state x_{k-1} for one branch m of the M possible branches $m \in \{0 \ldots M-1\}$ leading to state $x_k = i$ is determined by the value resulting from evaluation of the *state transition function* $Z()$: $x_{k-1} = Z(m, i)$.

$$\gamma_k^{(m,i)} = \gamma_{Z(m,i),k-1} + \lambda_k^{(m,i)} \quad , \quad m \in \{0, \ldots, M-1\} \tag{10}$$

The state metric is then determined by selecting the best path:

$$\gamma_{i,k} = \text{Max}\{\gamma_k^{(0,i)}, \ldots, \gamma_k^{(M-1,i)}\} \tag{11}$$

A sample ACS recursion for one state and $M = 2$ is shown in Fig. 17.3. This

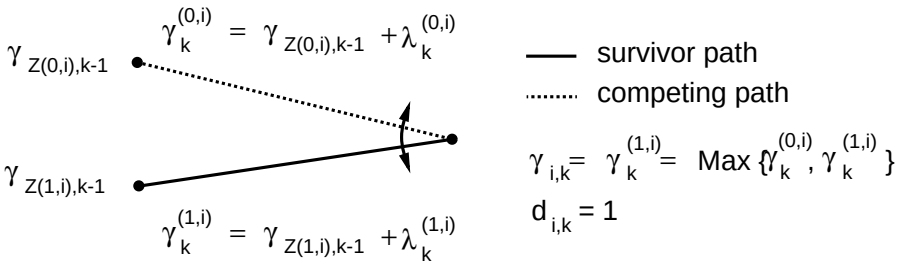

Figure 17.3 ACS recursion for $M = 2$

ACS recursion is performed for all N states in the trellis. The corresponding unit calculating the ACS recursion for all N states is called ACS unit (ACSU).

Despite the recursive computation, there are still N best paths pursued by the VA. The maximum likelihood path corresponding to the sequence $\hat{\boldsymbol{C}}$ can be finally determined only after reaching the last state in the trellis. In order to finally retrieve this path and the corresponding sequence of information symbols u_k, either the sequences of information symbols or the sequences of ACS decisions corresponding to each of the N survivor paths for all states i and all trellis steps k have to be stored in the *survivor memory unit* (SMU) as shown in Fig. 17.2 while calculating the ACS recursion. The decision for one branch m of $M = 2^k$ possible branches is represented by the *decision bits* $d_{i,k} = m$.

So far, we considered only the case that the trellis diagram is terminated, i.e. the start and end states are known. If the trellis is terminated, a final decision on the overall best path is possible only at the very end of the trellis. The decoding latency for the VA is then proportional to the length of the trellis. Additionally, the size of the SMU grows linearly with the length of the trellis. Finally, in applications like broadcasting, a continuous sequence of information bits has to be decoded rather than a terminated sequence, i.e. no known start and end state exists.

Fortunately, even in this case, certain asymptotic properties allow an approximate maximum likelihood sequence estimation with negligible performance losses and limited implementation effort. These are the acquisition and truncation properties [3] of the VA. Consider Fig. 17.4: the VA is pursuing N survivor paths

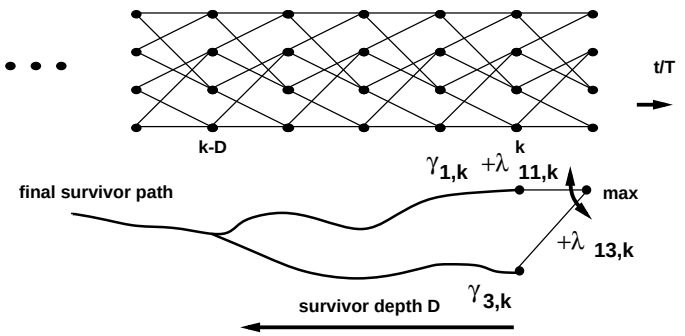

Figure 17.4 Path trajectories for the VA at an intermediate trellis step k.

at time instant k while decoding a certain trellis diagram. These paths merge, when traced back over time, into a single path as exemplarily shown by the path trajectories in Fig. 17.4. This path is called the *final survivor* below. For trellis steps smaller than $k - D$, the paths have merged into the final survivor with very high probability. The *survivor depth* D, which guarantees this behavior, depends strongly on the used code. Since *all* N paths at trellis step k merge into the final survivor, it is sufficient to actually consider only *one* path. Hence, it is possible to uniquely determine the final survivor path for the trellis steps with index smaller than $k - D$ already after performing the ACS recursion for trellis step k. This property enables decoding with a fixed latency of D trellis steps even for continuous transmission. Additionally, the survivor memory can be truncated: The SMU has to store only a fixed number of decisions $d_{i,j}$ for $i \in \{0, \ldots, N-1\}$ and $j \in \{k - D, k - D + 1, \ldots, k - 1, k\}$.

If the overall best path (the path with the best state metric) at trellis step k is used for determining the final survivor, the value of D guaranteeing that the final survivor is acquired with sufficiently high probability is the survivor depth. This procedure is called *best state decoding* [15, 16]. Sometimes, an arbitrary path is chosen instead, in order to save the computational effort required in order to determine the overall best path, which is called *fixed state decoding*. The properties of these decoding schemes will be discussed in section 17.5.4.

A phenomenon very similar to the just described truncation behavior occurs when the decoding process is started in midstream at trellis step k with an unknown start state. Due to the unknown start state, the ACS recursion is started with equal state metrics for all states. However, the decoding history which is necessary for reliable decoding of the survivor path is not available for the initial trellis steps. What happens if we perform the ACS recursion and try to decode the best path? As indicated in Fig. 17.5, the probability that the final survivor path differs from the correct path is then much larger than for decoding with a known start state. Fortunately, the same decoding quality as for decoding with known start state is achieved after processing a number of initial trellis steps. The number of trellis steps which are required here is called *acquisition depth*. It can be shown that the acquisition depth is equal to the survivor depth D [3, 17, 18]. This is also indicated in Fig. 17.5, where the merging of the paths takes place at trellis step $k + D$.

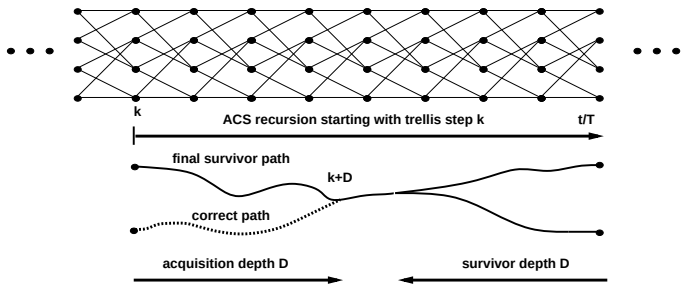

Figure 17.5 Path trajectories for acquisition.

Summarizing, the three basic units of a VD are depicted in Fig. 17.6. The branch metrics are calculated from the received symbols in the Transition Met-

ric Unit (TMU). These branch metrics are fed into the add–compare–select unit (ACSU), which performs the ACS recursion for all states. The decisions generated in the ACSU are stored and retrieved in the Survivor Memory Unit (SMU) in order to finally decode the source bits along the final survivor path. The ACSU is the only recursive part in a VD, as indicated by the latch. The branch metric computation is the only part which differs significantly if the VA is used for equalization instead of decoding.

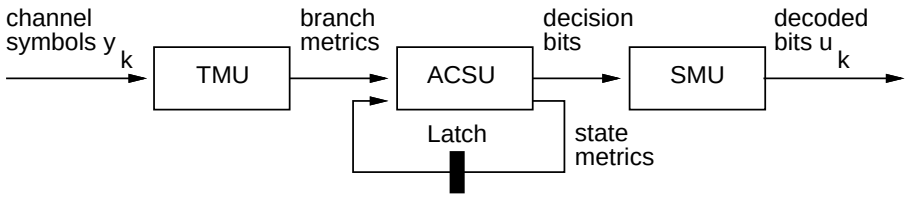

Figure 17.6 Viterbi Decoder block diagram.

Following, we state a computational model for transmitter, AWGN channel and receiver, that will be used in the subsequent sections. The model is shown in Fig. 17.7.

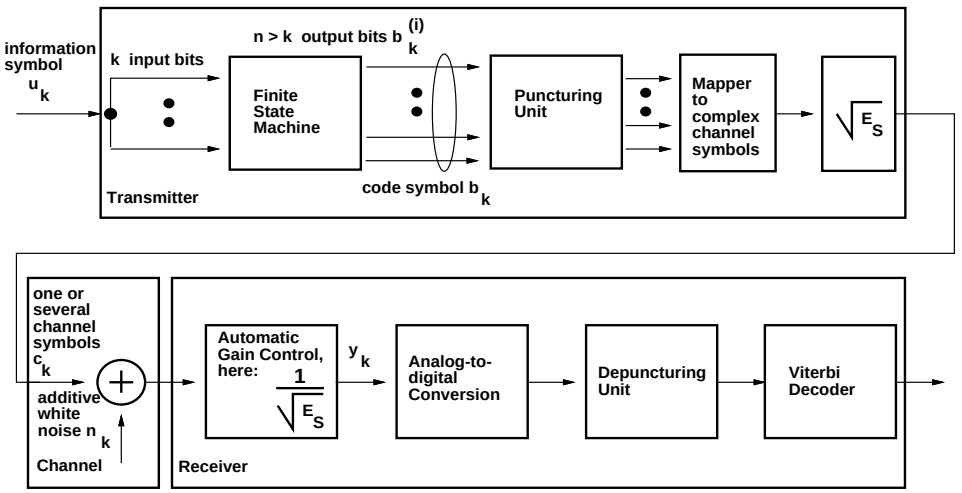

Figure 17.7 Computational model for transmitter, AWGN channel and receiver.

In our model, we assume that the channel symbols have energy normalized to unity after leaving the mapper for reasons of simplicity. Varying transmission conditions are modeled by changing the signal energy to noise ratio E_s/N_o. E_s is the signal energy, and N_o is the one sided power spectral density of the noise. Since the additive noise is assumed to have a constant variance in the model, changes in E_s/N_o are modeled by changing the gain in the scaling block at the transmitter output: $\sqrt{E_s}$. In the receiver, a unit implementing *automatic gain control* (AGC) is necessary in front of the analog–to–digital converter (ADC). In our computational

model, the AGC just implements a fixed scaling by $\frac{1}{\sqrt{E_s}}$, in order to normalize the energy of the received demodulated symbols y_k to unity again, which is just a matter of mathematical convenience. Therefore, the reference symbols in the decoder have the same magnitude and energy as in the encoder. Several issues related to AGC and ADC are discussed in section 17.3. For actual Viterbi decoder system design and assessment of the performance impact of all parameters and quantization effects, system design and simulation tools like COSSAP^{TM}[4] are indispensable.

17.2.1 Example: $K = 3$ Convolutional Code with BPSK

As an implementation example, we will use the $K = 3$ rate $1/2$ code with generator polynomials $(7, 5)$ with the trellis shown in Fig. 17.2. For BPSK, the $n = 2$ coded bits for a state transition in the encoder are mapped onto two complex valued BPSK symbols $\boldsymbol{c_k} = (\boldsymbol{c_{1,k}}, \boldsymbol{c_{2,k}})$ according to the mapping function:

$$\begin{aligned} \boldsymbol{c_{1,k}} &= 1 - 2b_{1,k} = \exp(i\pi b_{1,k}) \\ \boldsymbol{c_{2,k}} &= 1 - 2b_{2,k} = \exp(i\pi b_{2,k}) \end{aligned}$$

If the additive noise is gaussian, the channel is AWGN and the likelihood function $P(\boldsymbol{y_k}|\boldsymbol{c_k})$ for the two successive received complex valued symbols $\boldsymbol{y_k} = (\boldsymbol{y_{1,k}}, \boldsymbol{y_{2,k}})$ corresponding to a trellis transition is given by:

$$\begin{aligned} P(\boldsymbol{y_k}|\boldsymbol{c_k}) &= P(\boldsymbol{y_{1,k}}|\boldsymbol{c_{1,k}}) \cdot P(\boldsymbol{y_{2,k}}|\boldsymbol{c_{2,k}}) \\ &= \sqrt{\frac{E_s}{\pi N_o}} \exp\left(-\frac{E_s}{N_o}|\boldsymbol{y_{1,k}} - \boldsymbol{c_{1,k}}|^2\right) \cdot \sqrt{\frac{E_s}{\pi N_0}} \exp\left(-\frac{E_s}{N_o}|\boldsymbol{y_{2,k}} - \boldsymbol{c_{2,k}}|^2\right) \end{aligned} \tag{12}$$

Hence, the corresponding branch metric is given by:

$$\lambda_k^{(m,i)} = -\frac{E_s}{N_o}\left\{|\boldsymbol{y_{1,k}} - \boldsymbol{c_{1,k}}|^2 + |\boldsymbol{y_{2,k}} - \boldsymbol{c_{2,k}}|^2\right\} + 2ln(\sqrt{\frac{E_s}{\pi N_o}}) \tag{13}$$

The term $2ln(\ )$ which is common for all branch metrics can be neglected, since this does not affect the path selection. Since the imaginary part of the channel symbols is always zero, the imaginary part $y_{i,k,\mathrm{im}}$ of the received symbols $\boldsymbol{y_{i,k}} = y_{i,k,\mathrm{re}} + iy_{i,k,\mathrm{im}}$ only leads to an additive value which is common for all branch metrics and can be neglected. Furthermore, if the quotient of signal energy and noise power spectral density is constant over time, the factor $\frac{E_s}{N_o}$ can also be neglected:

$$\lambda_k^{(m,i)} \sim -\left\{(y_{1,k,\mathrm{re}} - c_{1,k,\mathrm{re}})^2 + (y_{2,k,\mathrm{re}} - c_{2,k,\mathrm{re}})^2\right\} \tag{14}$$

This calculation of the branch metrics is performed in the transition metric unit TMU.

In order to calculate the ACS recursion in the ACS unit, we have to define the state transition function for the used $K = 3$ code. For feedforward shift register coders, this function is given by

$$Z(m, x_k) = Z(m, \{x_{K-2,k}, \ldots, x_{0,k}\}) = \{x_{K-3,k}, \ldots, x_{0,k}, m\} \tag{15}$$

if the branch label m is chosen to be equal to the bit shifted out of the encoder for trellis step k.

For the resulting trellis with $N = 2^{3-1} = 4$ states (see Fig. 17.2) the ACS recursion is given by:

$$\begin{aligned}
\gamma_{0,k+1} &= \mathrm{Max}(\gamma_{0,k} + \lambda_{k+1}^{(0,0)}, \gamma_{1,k} + \lambda_{k+1}^{(1,0)}) \\
\gamma_{1,k+1} &= \mathrm{Max}(\gamma_{2,k} + \lambda_{k+1}^{(0,1)}, \gamma_{3,k} + \lambda_{k+1}^{(1,1)}) \\
\gamma_{2,k+1} &= \mathrm{Max}(\gamma_{0,k} + \lambda_{k+1}^{(0,2)}, \gamma_{1,k} + \lambda_{k+1}^{(1,2)}) \\
\gamma_{3,k+1} &= \mathrm{Max}(\gamma_{2,k} + \lambda_{k=1}^{(0,3)}, \gamma_{3,k} + \lambda_{k+1}^{(1,3)})
\end{aligned} \tag{16}$$

A generalization to more complex codes is obvious.

17.3 The transition metric unit (TMU)

In the TMU of a Viterbi decoder the branch metrics $\lambda_k^{(m,i)}$ are computed, which are used in the ACSU to update the new state metrics $\gamma_{i,k}$. The number of *different* branch metrics depends on the number of coded bits that are associated with a branch of the trellis. For a code of rate $\frac{k}{n}$, 2^n different branch metrics need to be computed for every trellis step. Since the ACSU uses only differences of path metrics to decide upon survivor selection, arbitrary constants can be added to the branch metrics belonging to a single trellis step without affecting the decisions of the Viterbi decoder. Choosing these constants appropriately can simplify implementations considerably.

Although the TMU can be quite complex if channel symbols of high complexity (e.g. 64-QAM, etc) need to be processed, its complexity is usually small compared to a complete Viterbi decoder. We restrict the discussion here to the case of BPSK modulation, rate 1/2 codes and additive white gaussian noise. We use $y_{i,k}$ instead of $y_{i,k,\mathrm{re}}$ and $c_{i,k}$ instead of $c_{i,k,\mathrm{re}}$ (cf Eq. (14)) in order to simplify the notation[6].

Starting from Eq. (14), we write the branch metrics as

$$\lambda_k^{(m,i)} = C_0 \left\{ (y_{1,k}^2 - 2y_{1,k}c_{1,k} + {c_{1,k}}^2) + (y_{2,k}^2 - 2y_{2,k}c_{2,k} + {c_{2,k}}^2) \right\} + C_1 \tag{17}$$

with C_0, C_1 being constants.

Since $c_{1,k}$ and $c_{2,k} \in \{-1, 1\}$ holds and the squared received symbols appear in all different branch metrics independently of the channel symbols that are associated with the branches, the squared terms are constant for a set of branch metrics and can be removed without affecting the decoding process[7]. Thus we can write the actually computed branch metrics as

$$\lambda_k^{(m,i)'} = C_2 \left\{ -y_{1,k}c_{1,k} - y_{2,k}c_{2,k} \right\} + C_3 \qquad \text{with constants} \qquad C_2 < 0, C_3 \tag{18}$$

[6] Note that extension to QPSK (quaternary phase shift keying) is obvious. Then, $y_{1,k}$ and $y_{2,k}$ denote the real and imaginary part of a single received complex valued symbol, respectively. $c_{1,k}$ and $c_{2,k}$ denote the real and imaginary part of a single complex valued QPSK channel symbol.

[7] The terms $(c_{x,k})^2$ are not constant for every modulation scheme (e.g. for 16-QAM) and thus cannot be neglected generally.

In Eq. (18), C_3 can be chosen independently for every trellis step k, while C_2 must be constant for different k to avoid deterioration of the decoding process. For hardware implementations C_3 is advantageously chosen such that $\lambda_k^{(m,i)'}$ is always positive. This enables the use of unsigned arithmetic in the ACSU for path metric computations. For SW implementations it is often advantageous to chose $C_3 = 0$ since then $\lambda_k^{(0,i)'} = -\lambda_k^{(1,i)'}$ holds for all *good* rate 1/2 codes. This can be used to reduce the computational complexity of the ACS computations.

17.3.1 Branch metric quantization

While the TMU usually has only a minor impact on the complexity of a Viterbi decoder, the ACSU is a major part. The complexity of the ACSU depends strongly on the wordlength of the branch metrics. It is thus important to reduce the branch metric wordlength to the required minimum.

It is well known for a long time that a wordlength of $w = 3$ bits is almost optimum for the received symbols in the case of BPSK modulation [6]. However, this requires virtually ideal gain control before the Viterbi decoder. Thus larger wordlengths are often used in practice to provide some margin for gain control. For actual determination of the wordlengths in the presence of a given gain control scheme and analog–to–digital conversion, system simulation have to be performed, which can be done easily using tools like COSSAP^{TM}[4].

To compute the branch metrics correctly it must be known how the "original" input value is quantized to the input value of the TMU consisting of w bits. As is pointed out already in [6], the quantization steps do not necessarily have to be equidistantly spaced. However, only such "linear" schemes are considered here.

Step at zero quantization

Probably the most widely used quantization characteristic is a symmetrical interpretation of a w-bit 2's complement number, by adding implicitely 0.5. Fig. 17.8 shows the characteristic of such a quantizer for 2 bits output wordlength. Q is the 2's complement output value of the quantizer, on the x-axis the normalized input value is given and on the y-axis the interpretation of the output value Q which actually is $Y = Q + 0.5$.

Table 17.1 shows range and interpretation again for a 3-bit integer output value of such a quantizer.

2's complement quantizer output value	-4	...	-1	0	...	3
interpretation due to quantizer characteristic	-3.5	...	-0.5	0.5	...	3.5

Table 17.1 Step at zero quantizer output value interpretation

Clearly, the quantizer input value 0 needs to be the decision threshold of the quantizer between the associated normalized integer values -1 and 0, that are interpreted as -0.5 and 0.5, respectively. Thus, the value zero cannot be represented and the actual range of a 2^w-level quantizer is symmetric. Even with a very low average signal level before the quantizer the sign of the input signal is still retained behind the quantizer. Thus the worst case performance using such a quantizer characteristic is equivalent to hard decision decoding. Using this interpretation and chosing $C_2 = 1$ in Eq. (18) and $w = 3$, the resulting range of the (integer

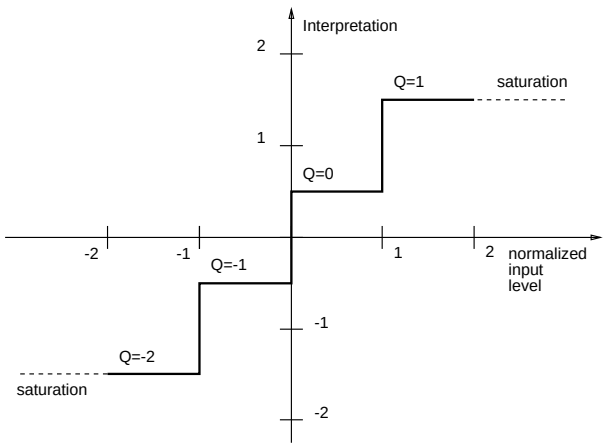

Figure 17.8 Characteristic of a 2-bit step-at-zero quantizer.

valued) branch metrics is

$$\begin{aligned} \mathrm{Min}(\lambda_k^{(m,i)}) &= C_3 - 3.5 - 3.5 = C_3 - 7 \\ \mathrm{Max}(\lambda_k^{(m,i)}) &= C_3 + 7 \end{aligned} \tag{19}$$

thus, $w+1$ bits are sufficient for the branch metrics and $C_3 = 2^w - 1$ can be chosen to obtain always positive branch metrics.

Dead zone quantizer

A second quantization approach is to take the usual 2's complement value without any offset. Fig. 17.9 shows the characteristic of a 2-bit dead zone quantizer.

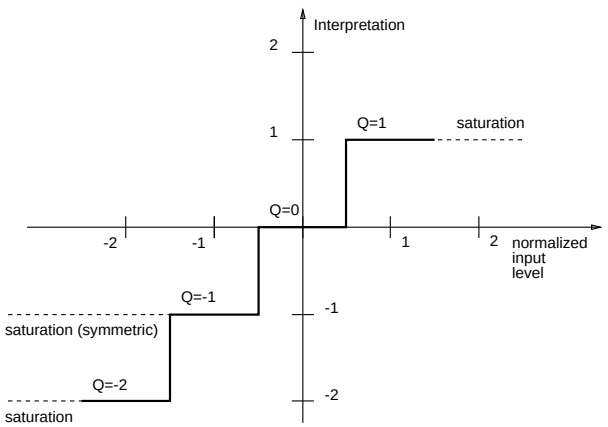

Figure 17.9 Characteristic of a 2-bit dead zone quantizer.

In this case the value 0 is output of the quantizer for a certain range around input value 0, nominally for $-0.5 < x \leq 0.5$. In contrast to *step at zero* quantization, very low average signal levels before quantization will ultimately result in

loosing the information about the input signal completely (even the sign), since the quantizer then outputs zero values only. When using this quantizer characteristic it is advantageous to compute the branch metrics as

$$\lambda_k^{(m,i)'} = C_3 + \frac{1}{2}\left\{(y_{1,k}c_{1,k} + \mathrm{Abs}(y_{1,k})) + (y_{2,k}c_{2,k} + \mathrm{Abs}(y_{2,k}))\right\} \tag{20}$$

This choice is legal since $\mathrm{Abs}(y_{1,k}) + \mathrm{Abs}(y_{2,k})$ is constant for every trellis step and thus does not influence the selection decisions of the ACSU. By choosing $C_3 = 0$ the range of the branch metrics is now

$$0 \leq \lambda_k^{(m,i)'} \leq 2\ \mathrm{Max}(\mathrm{Abs}(y_k)) \tag{21}$$

It is easily shown that the branch metrics are still integer values if computed according to Eq. (20). For a usual w-bit integer with range $\{-2^{w-1}, \ldots, 2^{w-1}-1\}$ the resulting branch metric range is $0, \ldots, 2^w$ which requires $(w+1)$-bit branch metrics. However, by making the quantizer output range symmetrical, i.e. constraining $y_{1,k}$ and $y_{2,k}$ to the interval $\{-2^{w-1}+1, \ldots, 2^{w-1}-1\}$ the branch metric range becomes $\{0, \ldots, 2^w - 1\}$ which can be represented with w-bit unsigned branch metrics (cf [19]). Since symmetry is anyway advantageous to avoid a biased decoding process, this is the option of choice.

With this approach we can either reduce the branch metric wordlength by one bit and reduce the quantization levels by one (e.g. from 8 levels to 7 levels for a 3-bit input value) or increase the input wordlength by one bit and thereby provide more margin for non-ideal gain control. Thus the approach either leads to decreased ACSU complexity or better performance at equal complexity since the TMU complexity is in most cases still marginal.

17.3.2 Support of punctured codes

Since punctured codes are derived from a *base* code by deleting some code bits prior to transmission, the decoder of the base code can be used if the TMU can compute the branch metrics such that the missing information does not affect the decisions of the remaining decoder. Assuming without loss of generality that the second received value $y_{2,k}$ in the example above is missing, the TMU has to compute the branch metrics such, that the terms

$$y_{2,k}c_{2,k} \qquad \text{respectively} \qquad y_{2,k}c_{2,k} + \mathrm{Abs}(y_{2,k}) \tag{22}$$

evaluate to a constant for all different branch metrics. To achieve this it is possible to either replace $y_{2,k}$ with 0, or to manipulate the metric computation such that $c_{2,k}$ is constant for all computed branch metrics, which is equivalent to relabeling part of the branches of the trellis with different code symbols.

Clearly, the first approach is applicable only if one of the quantized values is actually interpreted as 0 (as for the Dead zone quantizer discussed above) since $y_{2,k} = 0$ can easily be chosen. For step at zero quantization, where the quantized values are interpreted with an implicit offset of 0.5, manipulating the branch labels is the better choice since a replacement value of 0 is not straightforwardly available.

17.4 The Add–Compare–Select Unit

Given the branch metrics, the ACSU calculates the state metrics according to the ACS recursion, which represents a system of nonlinear recurrence equations. Since the ACS operation is the only recursive part of the Viterbi algorithm, the achievable data (and clock) rate of a VLSI implementation is determined by the computation time of the ACS recursion. Due to the repeated accumulation of branch metrics to state metrics, the magnitude of these metrics is potentially unbounded. Hence, *metric normalization schemes* are necessary for a fixed wordlength implementation.

17.4.1 Metric normalization schemes

In order to prevent arithmetic overflow situations and in order to keep the register effort and the combinatorial delay for the add and compare operations in the ACSU as small as possible, metric normalization schemes are used.

Several methods for state metric normalization are known, which are based on two facts [20]:

1. The differences $\Delta\gamma_k$ between all state metrics at any trellis step k are bounded in magnitude by a fixed quantity $\Delta\gamma_{Max}$ independent of the number of ACS operations already performed in the trellis.

2. A common value may be subtracted from all state metrics for any trellis step k, since the subtraction of a common value does not have any impact on the results of the following metric comparisons.

Consider all paths starting from a given state $s_{i,k}$ in the trellis, corresponding to the state $x_k = i$ in the encoder. After a certain number n of trellis steps, all other states can be reached starting with $x_k = i$. Since one bit is shifted into the encoder shift register for every trellis step, n is obviously equal to $K-1$. In other words, after $K-1$ steps, an arbitrary shift register state is possible independent of the initial state. Hence, the interval n ensures complete connectivity for all trellis states. In the trellis, there are N distinct paths from the starting state to all other states $s_{j,k+n}$, $j \in \{0, \ldots, N-1\}$. An upper bound on the state metric difference $\Delta\gamma_{Max}$ can be found assuming that for one of these paths, the added branch metric $\lambda_k^{(m,i)}$ was minimum for all n transitions, and for another of these paths, the branch metric was always maximum. Hence, an upper bound on the maximum metric difference is given by

$$\Delta\gamma_{Max} = n \cdot (\max(\lambda_k^{(m,i)}) - \min(\lambda_k^{(m,i)}))$$

with $n = K-1$ and $\max(\lambda_k^{(m,i)})$ and $\min(\lambda_k^{(m,i)})$ being the maximum and minimum metric value possible using the chosen branch metric quantization scheme. The wordlength necessary to represent $\Delta\gamma_{Max}$ is the minimum wordlength required for the state metrics[8]. However, depending on the chosen normalization scheme, a larger wordlength has actually to be used in most cases. We now state two normalization schemes:

[8] Even tighter bounds on the state metric differences were derived in [21].

1. Subtracting the minimum State Metric
After a given number of trellis steps, the minimum state metric is determined and subtracted from all other state metrics. This scheme leads to the minimum state metric wordlength as derived above, if it is performed for every trellis step. The resulting architecture for a single ACS processing element (PE) using this normalization scheme is shown in Fig. 17.10.
If a normalization is performed only after a certain number of trellis steps, an increased wordlength of the state metrics has to be taken into account.

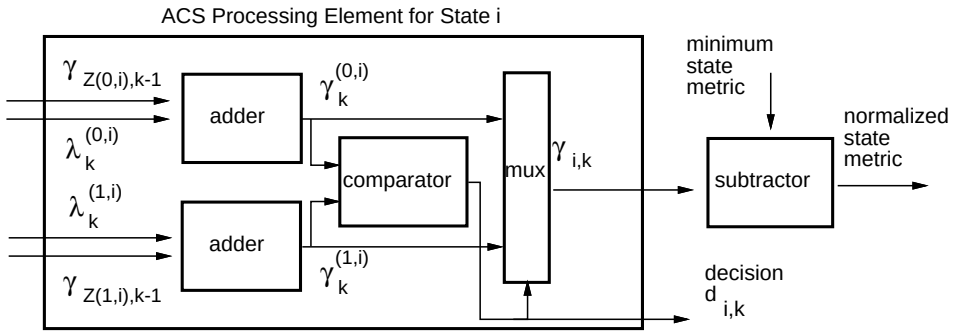

Figure 17.10 ACS processing element and minimum state metric subtraction.

The additional computational effort involved with this scheme is relatively large: first, the minimum state metric has to be determined, second, a subtraction has to be performed in addition to the usual add–compare–select operation. However, it may be suited for low throughput architectures and software applications. The minimum state metric can then be determined sequentially while successively calculating the new state metrics for a trellis transition, and the effort for the additional subtraction does not pose a significant problem.

2. "On the fly" Normalization schemes
For high throughput applications, the ACS recursion is implemented with a dedicated ACS PE per trellis state. In this case, N new state metrics are calculated in parallel for all states. Determining the minimum of these metrics would require much more processing delay than the ACS calculation itself, hence more efficient ways have to be found for normalization.

A very efficient normalization scheme can be found again exploiting the upper bound on the metric difference $\Delta\gamma_{Max}$. The idea is simply to subtract a fixed value from all state metrics if the state metrics exceed a certain threshold t. Simultaneously, it has to be guaranteed that no overflows or underflows occur for all state metrics. The value of the threshold t can be chosen such that the detection of a threshold excess and the necessary subtraction can be implemented as efficiently as possible, while keeping the state metric wordlength as small as possible. In the following, one of the possible solutions for unsigned branch and state metrics is presented:
The unsigned branch metrics are quantized to b bits, leading to a maximum branch metric value of $2^b - 1 = \max(\lambda_k^{(m,i)})$. The unsigned state metrics are quantized with p bits, corresponding to a maximum value of $2^p - 1$. Of course, $\Delta\gamma_{Max} \leq 2^p - 1$

must hold. If the number of bits p is chosen such that

$$\Delta\gamma_{Max} \leq 2^{p-2}$$

a very efficient normalization without additional subtraction can be derived. It is now sufficient to observe just the value of the most significant bit (MSB). If any state metric value gets equal to or exceeds the value $t = 2^{p-1}$, it is simultaneously known that all other state metrics are equal to or larger than 2^{p-2} because of the limited state metric difference. Hence, it is possible to subtract the value of 2^{p-2} from all state metrics while guaranteeing that all state metrics remain positive.

Both the test of the MSB and the subtraction of 2^{p-2} can be implemented using very simple combinatorial logic involving only the two MSBs and a few combinatorial gates.

The inspection of the MSBs for all state metrics still requires global communication between all ACS PEs. This drawback can be removed by using modulo arithmetic for the state metrics as proposed in [20]. Metric values exceeding the range just wrap around according to the modulo arithmetic scheme, hence no global detection of this situation is necessary. However, the state metric wordlength has also to be increased to a value larger than the minimum given by $\Delta\gamma_{Max}$. Details can be found in [20].

Due to the recursive nature of the ACS processing, the combinatorial delay through the ACS PE determines the clock frequency (and hence the decoded bit rate) of the whole Viterbi decoder. Arithmetic and logic optimization of the ACS PE is therefore essential. Many proposals exist for optimizing the arithmetic in the ACS. Every conventional addition scheme suffers from the fact that the combinatorial delay is some function of the wordlength, since a carry propagation occurs. Redundant number systems allow carry free or limited carry propagation addition [22, 23]. However, the maximum selection can not be solved straightforwardly in redundant number systems. Nevertheless, a method was proposed allowing to use the redundant carry–save number system for the ACS processing, which can be very beneficial if large wordlengths have to be used [17, 18].

17.4.2 Recursive ACS Architectures

We first consider the case that one step of the ACS recursion has to be calculated in one clock cycle, and later briefly disuss lower throughput architectures. If a dedicated ACS PE is used for every state in the trellis, the resulting node parallel architecture with a throughput of one trellis step per clock cycle is shown in Fig. 17.11. For reasons of simplicity, the state metric normalization is not shown in this picture. A complete vector of decisions $d_{i,k}$ is calculated for every clock cycle. These decisions are stored in the SMU in order to facilitate the path reconstruction. Obviously, a large wiring overhead occurs, since the state metrics have to be fed back into the ACS PEs. The feedback network is a *shuffle–exchange* network. The possible state transitions for the states $x_k = \sum_{j=0}^{\nu-1} x_{j,k}\ 2^j$ are given by a cyclic shift (perfect shuffle) $x_{0,k}, x_{\nu-2,k}, \ldots, x_{1,k}$ and an exchange $\overline{x_{0,k}}, x_{\nu-2,k}, \ldots, x_{1,k}$. where $\overline{x_{0,k}}$ denotes inversion of $x_{0,k}$. Many proposals exist for optimum placement and routing of this type of interconnection network (see e.g. [24]).

For lower throughput applications, several clock cycles are available for a single ACS recursion. Here, the fact can be exploited that the trellis diagram for nonrecursive rate $1/n$ codes includes butterfly structures as known from FFT pro-

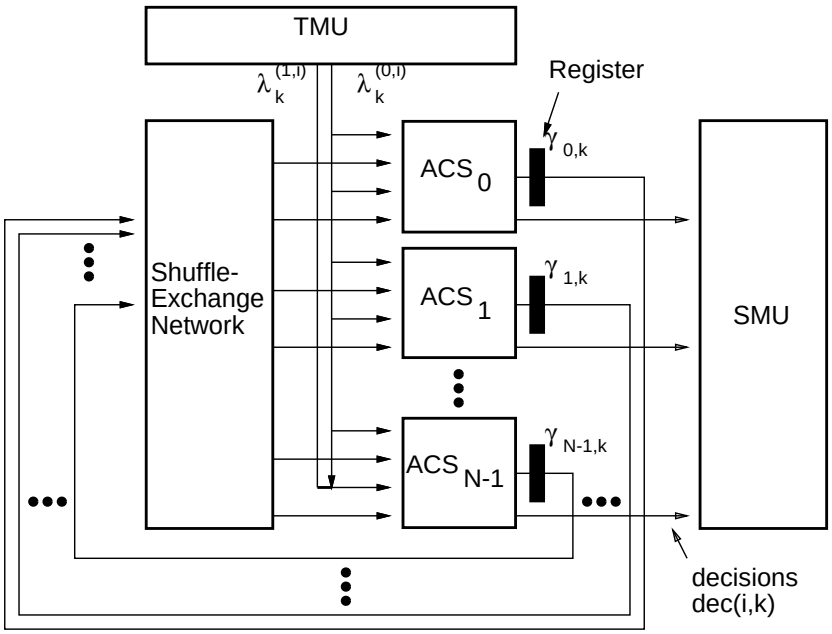

Figure 17.11 Node parallel ACS architecture.

cessing. Since for rate $1/n$ codes, just a single bit is shifted into the encoder FSM, the transition function specifying the two predecessor states for a current state $x_k = \{x_{K-2,k}, \ldots, x_{0,k}\}$ is given by $Z(m, x_k) = \{x_{K-3,k}, \ldots, x_{0,k}, m\}$ as stated in Eq. (15). These two predecessor states $Z(0, x_k)$ and $Z(1, x_k)$ have exactly two successor states: $\{x_{K-2,k}, \ldots, x_{0,k}\}$ and $\{\overline{x_{K-2,k}}, \ldots, x_{0,k}\}$ with $\overline{x_{K-2,k}}$ denoting bit inversion. This results in the well known butterfly structure as shown in Fig. 17.12.

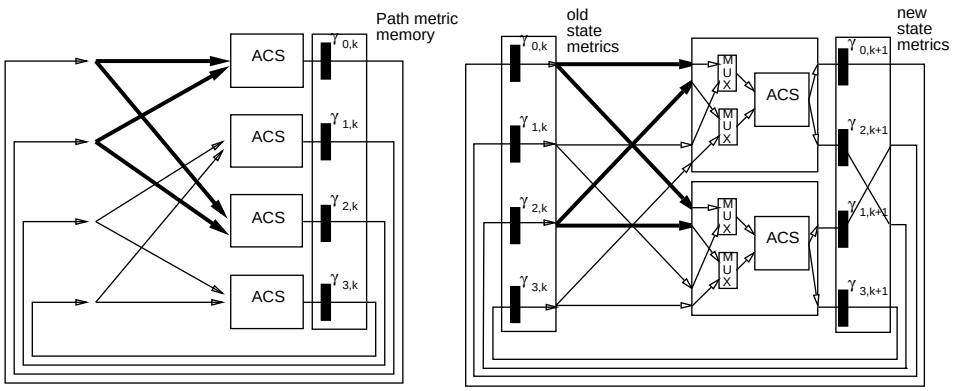

Figure 17.12 Butterfly trellis structure and resource sharing for the $K = 3$, rate 1/2 code.

In order to calculate two new state metrics contained in a butterfly, only two predecessor state metrics and two branch metrics have to be provided. Hence a sequential schedule for calculating all ACS operations in the trellis is given by a sequential calculation of the butterflies. This is shown on the right hand side of Fig. 17.12: two ACS PEs calculate sequentially the two ACS operations belonging to a butterfly, hence, a complete trellis step takes two clock cycles here. The ACS operations according to the thick butterfly are calculated in the first clock cycle, and the remaining ACS operations are calculated in the second clock cycle.

In Fig. 17.12, a parallel ACS architecture with four ACS PEs and a resource shared architecture with two ACS PEs are shown for the $K = 3$ code. As shown in Fig. 17.12, it seems to be necessary to double the memory for the state metrics compared to the parallel ACS architecture, since the state metrics $\gamma_{i,k+1}$ are calculated while the old metrics $\gamma_{i,k}$ are still needed. It was shown in [25], however, that an in–place memory access for the state metrics is possible with a cyclic metric addressing scheme. Here, only the same amount of memory is necessary as for the parallel ACS architecture. Several proposals for resource shared ACSU implementations can be found in [26, 27, 28, 29].

17.4.3 Parallelized ACS Architectures

The nonlinear data dependent nature of the recursion excludes the application of known parallelization strategies like pipelining or look-ahead processing, which are available for parallelizing linear recursions[30]. It was shown [31, 18, 32] that a linear algebraic formulation of the ACS recursion can be derived, which, together with the use of the acquisition and truncation properties of the Viterbi algorithm, allows to derive purely feedforward architectures[31]. Additionally, the linear algebraic formulation represents a very convenient way to describe a variety of ACS architectures.

Below, the algebraic multiplication $\otimes$ denotes addition and the algebraic addition $\oplus$ denotes maximum selection. The resulting algebraic structure of a semiring defined over the operations $\oplus$ and $\otimes$ contains the following neutral elements:

- neutral element concerning $\oplus$ (maximum selection): $Q(= -\infty)$
- neutral element concerning $\otimes$ (addition): $\underline{1}(= 0)$

Using the semiring algebra, the ACS recursion for the $K = 3$, rate $1/2$ code as stated in Eq. (15) can be written as:

$$\begin{aligned}
\begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{pmatrix}_{k+1} &= \begin{pmatrix} \max(\lambda_{00}+\gamma_0;\lambda_{01}+\gamma_1) \\ \max(\lambda_{12}+\gamma_2;\lambda_{13}+\gamma_3) \\ \max(\lambda_{20}+\gamma_0;\lambda_{21}+\gamma_1) \\ \max(\lambda_{32}+\gamma_2;\lambda_{33}+\gamma_3) \end{pmatrix}_k = \begin{pmatrix} (\lambda_{00}\otimes\gamma_0)\oplus(\lambda_{01}\otimes\gamma_1) \\ (\lambda_{12}\otimes\gamma_2)\oplus(\lambda_{13}\otimes\gamma_3) \\ (\lambda_{20}\otimes\gamma_0)\oplus(\lambda_{21}\otimes\gamma_1) \\ (\lambda_{32}\otimes\gamma_2)\oplus(\lambda_{33}\otimes\gamma_3) \end{pmatrix}_k \\
&= \begin{pmatrix} \lambda_{00} & \lambda_{01} & Q & Q \\ Q & Q & \lambda_{12} & \lambda_{13} \\ \lambda_{20} & \lambda_{21} & Q & Q \\ Q & Q & \lambda_{32} & \lambda_{33} \end{pmatrix}_k \otimes \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{pmatrix}_k
\end{aligned} \tag{23}$$

Transition metrics not corresponding to allowed state transitions are assigned the metric $Q = -\infty$. Of course, no computational effort is required for terms including

the Q–value. Given a state metric vector $\Gamma_k = (\gamma_{0,k}, \ldots, \gamma_{N-1,k})^T$ and an NxN transition matrix Λ_k containing all the transition metrics $\lambda_{ij,k}$, the above equation can be written as a matrix–vector product:

$$\Gamma_{k+1} = \Lambda_k \otimes \Gamma_k$$

It can be shown, that all rules known from linear algebra are applicable to this linear algebraic formulation of the ACS recursion as well. Hence, is represents much more than just a convenient notation, but allows to derive new algorithms and architectures. It is e.g. possible to arrive at an M-step ACS recursion:

$$\Gamma_{k+M} =_M \Lambda_k \otimes \Gamma_k = (\Lambda_{k+M-1} \otimes \Lambda_{k+M-2} \otimes \ldots \otimes \Lambda_k) \otimes \Gamma_k$$

with an M-step transition matrix ${}_M\Lambda_k$ describing the NxN optimum transition metrics from every state at trellis step k to every state at trellis step $k+M$. This approach is just another formulation of the original ACS recursion, i.e. the results are exactly equivalent. Associativity of the $\otimes$ operation allows to reformulate the recursion in this way.

This M step processing already allows a parallelization, since the M matrix–matrix products can be calculated in advance, and the actual recursion now spans M trellis steps, leading to a speedup factor of M. A disadvantage of the M-step approach is the computational effort necessary to calculate matrix-matrix products for the NxN matrices Λ_k. The matrices for single transitions contain many Q–entries, as shown in the example Eq. (22). With successive matrix–matrix multiplications, the number of Q–entries soon becomes zero, leading to an increased effort for matrix multiplications since there is no computation necessary for the Q–entries as stated above. Hence, an implementation for small M and especially $M = 2$ as reported in [33] seems to be particularly attractive. In [33] it is proposed to unfold the ACS recursion for a number of successive trellis steps, which is equivalent to introducing an M–step recursion. A two–step ACS recursion is also advantageous because there is more room for arithmetic optimization of the recursion equations, since the concatenation of successive additions can be implemented quite advantageously. Since two vectors of decison bits are generated simultaneously for a single clock cycle, the resulting decoded bit rate is two times the clock frequency.
For larger values of M, however, there is a significant increase in the computational effort when using the M–step approach.

However, it was shown by Fettweis [17, 18] that it is even possible to derive an independent processing of blocks of received symbols leading to a purely feedforward solution with an arbitrary degree of parallelism, the so called *minimized method*. The key to this approach is the exploitation of the acquisition and truncation properties of the VA.

We first review conventional Viterbi decoding with regard to the linear algebraic formulation: the M-step transition matrix contains the best paths from every state at trellis step k to every state at trellis step $k+M$:

$${}_M\Lambda_k = \begin{pmatrix} {}_M\lambda_{00} & {}_M\lambda_{01} & \cdots & {}_M\lambda_{0(N-1)} \\ {}_M\lambda_{10} & {}_M\lambda_{11} & \cdots & {}_M\lambda_{1(N-1)} \\ {}_M\lambda_{(N-1)0} & {}_M\lambda_{(N-1)1} & \cdots & {}_M\lambda_{(N-1)(N-1)} \end{pmatrix}_k$$

Each entry ${}_M\lambda_{ij}$ contains the metric of the best path from state j at trellis step k to state i at trellis step $k+M$.

The conventional VA (for $M = D$) calculates recursively $\Lambda_{k+D} \otimes (\ldots \otimes (\Lambda_k \otimes \Gamma_k))$ which is equal to ${}_D\Lambda_k \otimes \Gamma_k$. Hence the VA operation can also be interpreted as follows: the VA adds the state metrics at trellis step k to the corresponding matrix entries and then perform a rowwise (concerning ${}_D\Lambda_k$) maximum selection leading to metrics for the N best paths at time instant $k+D$. If *best state decoding* [15] is applied, the VA finally selects the overall maximum likelihood survivor path with metric $\gamma_{i,k+D} =_D \lambda_{ij} + \gamma_{j,k}$ including decoding the best state $x_k = j$ at time instant k.

The conventional VA with best state decoding for trellis step k can hence also be represented as

$$(\underline{1}, \ldots, \underline{1}) \otimes_D \Lambda_k \otimes \Gamma_k = \gamma_{i,k+D} =_D \lambda_{ij} + \gamma_{j,k} \tag{24}$$

since the multiplication with $(\underline{1}, \ldots, \underline{1})$ in the semiring algebra corresponds to the final overall maximum selection in conventional arithmetic. It is obvious that the best state $x_k = j$ can be immediately accessed via the indices of the overall best metric $\gamma_{i,k+D} =_D \lambda_{ij} + \gamma_{j,k}$.

The state metric vector Γ_k can be calculated by an acquisition iteration. It was already discussed that the acquisition depth is equal to the survivor depth D. Therefore, we start decoding in midstream at $k - D$ with all state metrics equal to zero.

$$\Gamma_k =_D \Lambda_{k-D} \otimes (\underline{1}, \ldots, \underline{1})^T$$

Replacing Γ_k in Eq. (24) leads to:

$$\begin{aligned} & \underbrace{((\underline{1}, \ldots, \underline{1}) \otimes_D \Lambda_k)}_{\text{truncation}} \otimes \underbrace{({}_D\Lambda_{k-D} \otimes (\underline{1}, \ldots, \underline{1})^T)}_{\text{acquisition}} \\ = \; & \underbrace{(({}_D\Lambda_k)^T \otimes (\underline{1}, \ldots, \underline{1})^T)^T)}_{\text{truncation}} \otimes \underbrace{({}_D\Lambda_{k-D} \otimes (\underline{1}, \ldots, \underline{1})^T)}_{\text{acquisition}} \\ = \; & (\gamma_{0,k,T}, \ldots, \gamma_{N-1,k,T}) \otimes (\gamma_{0,k,A}, \ldots, \gamma_{N-1,k,A})^T \end{aligned} \tag{25}$$

Both matrix–vector products are equal to the usual ACS operation. Since Eq. (24) is purely feedforward, we call the two iterations *ACS acquisition iterations* below. Each of the two iterations leads to a state metric vector as shown in Eq. (24), where the index T denotes truncation and the index A denotes acquisition. In a final step, the two state metrics which are resulting per state are added, and the overall maximum metric is determined. This can be verified by writing out the final expression in Eq. (24) and replacing semiring arithmetic with conventional arithmetic. The state corresponding to the global maximum is finally decoded. A parallel architecture implementing Eq. (24) is shown in Fig. 17.13.

Obviously, an arbitrary number of ACS acquisition iterations can be performed independently and hence in parallel for an arbitrary number of blocks containing $2M$ symbols with $M \geq D$. It is most efficient to use nonoverlapping contiguous blocks of length $2D$ for this operation. The result is a number of uniquely decoded states with distance $2D$ transitions, i.e. x_k, x_{k+2D}, ...

Using the known states resulting from the ACS acquisition iterations, a second ACS iteration is started. The survivor decisions generated here are used – as for a conventional Viterbi Decoder – to finally trace back the decoded paths. For the

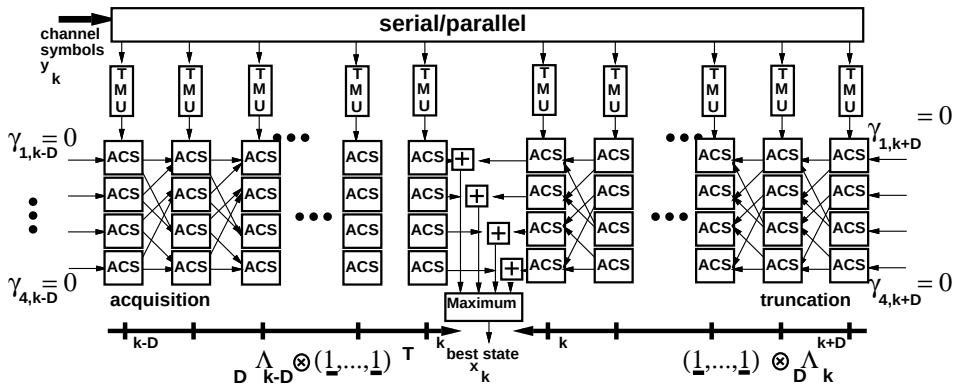

Figure 17.13 Architecture for the ACS acquisition iterations for the $K = 3$, rate 1/2 code. Each node represents a dedicated ACSU for one trellis step.

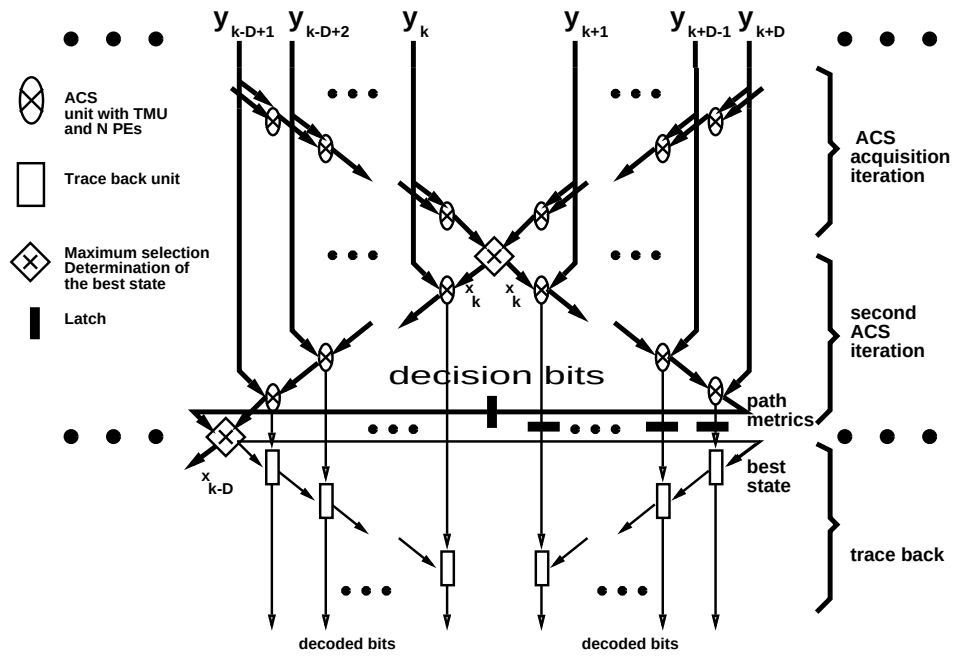

Figure 17.14 Architecture for the minimized method. The ovals represent ACS operation, the diamonds the determination of the best state, and the rectangles represent the trace back operation in forward or backward direction.

trace back, best state decoding is performed, since the overall best state x_{k-D} is determined and used as a starting point for the trace back.

The resulting architecture that processes one block at a time is shown in Fig. 17.14. It consumes one block of input symbols for every clock cycle. The

latches are necessary to store values which are needed for the following block to be processed in the next clock cycle.

It is possible to extend the architecture given in Fig. 17.14 by identical modules on the left and right hand side, leading to an even faster architecture that consumes a number of blocks at a time. Therefore, in principle, an arbitrary degree of parallelism can be achieved.

A detailed description of the Minimized Method architecture is given in [34, 35, 36]. In order to achieve Gbit/s speed, a fully parallel and pipelined implementation of the Minimized Method was developed and realized as a CMOS ASIC for Gbit/s Viterbi decoding [34]. Here, one dedicated ACS unit, with a dedicated ACS processing element (PE) for every state, is implemented for each trellis transition. Bit level pipelining is implemented for the ACS PEs, which is possible since the minimized method is purely feedforward. The fabricated ASIC [34] is one order of magnitude faster than any other known Viterbi decoder implementation.

17.5 The survivor memory unit (SMU)

As was explained earlier, in principle all paths that are associated with the trellis states at a certain time step k have to be reconstructed until they all have merged to find the *final survivor* and thus the decoded information. However, in practice only one path is reconstructed and the associated information at trellis step $k - D$ output (cf Fig. 17.4). D must be chosen such that all paths have merged with sufficiently high probablility. If D is chosen too small (taking into account code properties and whether fixed or best state decoding is performed) substantial performance degradations result. The path reconstruction uses stored path decisions from the ACSU. Clearly, the survivor depth D is an important parameter of the SMU since the required memory to store the path decisions is directly proportional to D for a fixed implementation architecture.

Fixed state decoding is usually preferred in parallel HW implementations since finding the largest state metric (for best state decoding) can be both, time critical and costly in HW. However, since a significantly larger D must be chosen for fixed state decoding [15, 16], the involved trade-off should be studied thoroughly for optimum implementation efficiency.

For the actual decoding process that is implemented in the SMU two different algorithms are known: the register exchange algorithm (REA) and the traceback algorithm (TBA) [25, 6]. While register exchange implementations of SMUs are known to be superior in terms of regularity of the design and decoding latency, traceback implementations typically achieve lower power consumption and are more easily adapted to lower speed requirements where more than one clock cycle is available per trellis cycle for processing the data [37, 38]. While we focus here on TBA and REA, it should be noted that the implementation of a hybrid REA/TBA architecture was reported in [39] and further generalizations are treated in [40, 41].

17.5.1 REA

The REA computes the information symbol sequences of the survivor paths associated with all N trellis-states based on the decisions provided by the ACSU.

If we denote the information symbol associated with the reconstructed path belonging to state i at trellis step k as $\hat{u}_k^{[i]}$ and the information symbol associated

with the m'th branch merging into state i as $u^{(m,i)}$, we can formally state the algorithm as follows:

```
Memory:
   (D + 1) · N Information Symbols (û_k^[i], ..., û_{k-D}^[i]) ;
Algorithm:
   // Update of the stored symbol sequences according to
   // the current decision bits d_{i,k}
   for t=k-D to k-1 {
      for State=0 to N-1 {
         û_t^[State] := û_{t+1}^[Z(d_{State,k,State})] ;
      }
   }
   // setting the first information symbol of the path
   for State=0 to N-1 {
      û_k^[State] := u^(d_{State,k,State)} ;
   }
```

Here, $Z(m, x_k)$ is the state transition function as defined in Equation 15. The nested loop describes how the stored information sequences corresponding to the best paths at trellis step $k-1$ are copied according to the decision bits deci,k obtained at trellis step k in the ACS recursion. In the final loop the information sequences for the N best paths are preceded with the information bits for step k. For example, if at time k and state i, the path according to the branch with label $m = 1$ is selected as the best path, the stored symbol sequence for the state branch 1 emerged from is copied as the new symbol sequence of state i preceded by the information symbol associated with branch 1.
Assuming l-bit information symbols, the algorithm requires $D \cdot N \cdot l$ bits of memory for a code with N states. If we define the decoding latency as the difference between the most recently processed trellis step k and the trellis step, the decoded information is associated with, the decoding latency of the REA is identical to the survivor depth D (neglecting implementation related delays, e.g. pipeline delay). Both figures are the minimum achievable for a given N and D. However, access bandwidth to the memory is very high. Per trellis cycle each survivor symbol sequence is completely overwritten with new survivor symbols resulting in $D \cdot N$ read and write accesses per cycle. Therefore in a fully parallel implementation the cells are usually implemented as flip-flops (registers) and the selection of the possible input symbols is done by means of multiplexors. Fig. 17.15 shows the resulting hardware architecture for the sample $K = 3$, rate $1/2$ code with 4 states and binary information symbols. The topology of the connections correspond to the trellis topology, which can be a major drawback of the approach for large number of states. Power consumption can be a problem in VLSI implementations due to the high access bandwidth [42, 43]. As a consequence, the REA is usually not used for low data rate decoders. It is applied if latency, regularity or total memory size are critical parameters.

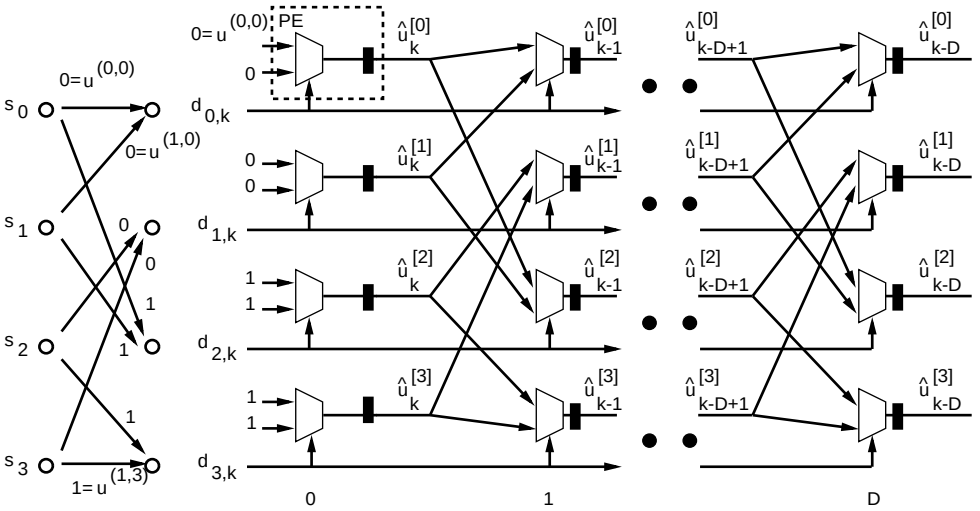

Figure 17.15 REA hardware architecture.

17.5.2 TBA

In contrast to the REA the TBA does not compute the *information symbol* sequence associated with each state. Instead, the *state* sequence is computed based on the path decisions $d_{i,k}$. In a second step the associated information symbols $\hat{u}_k$ are computed. In practice, using D reconstruction steps a state of the final survivor is acquired (D : survivor depth). Subsequently, the path is traced back M steps further to obtain M symbols that are associated with the final survivor [37, 38] (M : decoding depth). Fig. 17.16 shows a sample traceback sequence. Formally, we

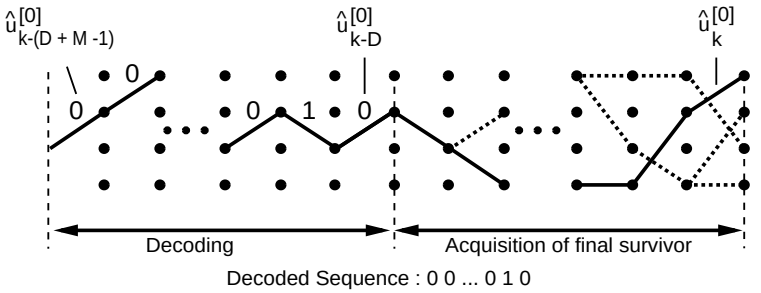

Figure 17.16 Example for the TBA.

can state the TBA as follows:

```
Memory:
   (D + M) · N decision bits (d_{i,k}, ..., d_{i,k-(D+M-1)}) ;
Algorithm:
   // every M trellis steps a trace back is started
```

```
if (k-D can be divided by M) then {
   // Initialization
   traceState := startState ;
   // Acquisition
   for t=k downto k-D+1 {
      traceState := Z(d_{traceState,t},traceState) ;
   }
   // Decoding
   for t=k-D downto k-D-M+1 {
      û_t^[d] := u^(d_{traceState,t,traceState)} ;
      traceState := Z(d_{traceState,t},traceState) ;
   }
}
```

The memory size of an implementation of our example must be at least $(D + M) \cdot N$ bits to facilitate performing a data traceback with depth M while maintaining D as survivor depth. Furthermore the decoding latency is increased to at least $D+M$ because tracing back requires $M+D$ ACS iterations to be performed before the first trace back can be started[9]. Blocks of M symbols are decoded in reverse order during the data traceback phase, thus a last-in first-out (LIFO) memory is required for reversing the order before outputting the information. Fast hardware implementations require more memory and exhibit a larger latency.

The algorithm requires write accesses to store the N decision bits. Since a trace back is started only every M trellis steps, on average $(M+D)/M$ decision bits are read and trellis steps reconstructed for the computation of a single information symbol. Thus the access bandwidth and computational requirements are greatly reduced compared to register exchange SMUs so RAMs can be used for storage which can be implemented in VLSI with a much higher density than flipflops particularly in semi-custom technologies. Thus TBA implementations are usually more power efficient compared to REA implementations.

Furthermore, the choice of M relative to D (D is usually specified) allows memory requirements to be traded against computational complexity. And the TBA can thus be adapted to constraints of the target technology more easily [37, 38, 44]. We will review the basic trade-off in the next section.

17.5.3 TBA trade offs

The inherent trade-off in the TBA is best understood if visualized. This can be done with a clock-time/trellis-time diagram, where the actual ongoing time measured in clock cycles is given on the x-axis, while the time in the trellis is given on the y-axis. Fig. 17.17 shows such a diagram for the TBA with $M = D$.

Henceforth we assume that a new set of decision bits is generated in every clock cycle as is usually required in fast hardware implementations e.g. for digital video broadcasting applications. Consider the first complete cycle of traceback processing in Fig. 17.17 (Cycle 1). During the first $2 \cdot D$ clock cycles, sets of decision bits are written to the memory. In the subsequent D clock cycles the data is retrieved to facilitate the acquisition of the final survivor (acquisition-trace). Finally the

[9] This minimum figure is only achievable in low rate applications, since the actual computation time for reconstruction is not already included!

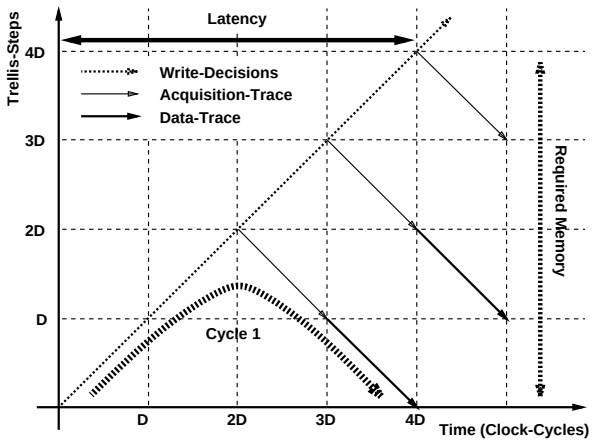

Figure 17.17 Traceback with $M = D$.

data is decoded by retrieving $M = D$ sets of decision bits and tracing the final survivor further back over time (data-trace), while concurrently a new acquisition-trace is performed starting from trellis step $3 \cdot D$. In fact we obtain a solution where acquisition-trace and data-trace are always performed concurrently, i.e. we obtained a solution which requires two read pointers. The latency of the data-trace is obtained as the difference in clock cycles from data write (at trellis step 0) until the written information is accessed the last time (at clock cycle $4 \cdot D$). The required memory is obtained on the y-axis as the number of kept decision bits before memory can be reused ($4 \cdot D$ trellis steps with a total of $4 \cdot D \cdot N$ bits). We assumed here that during the data-trace the memory is not immediately reused by writing the new data into the memory location just read. Immediate reuse is possible if the used memory technology permits a read and a write cycle to be performed in one clock cycle which is usually not possible in commodity semi-custom technologies for high throughput requirements.

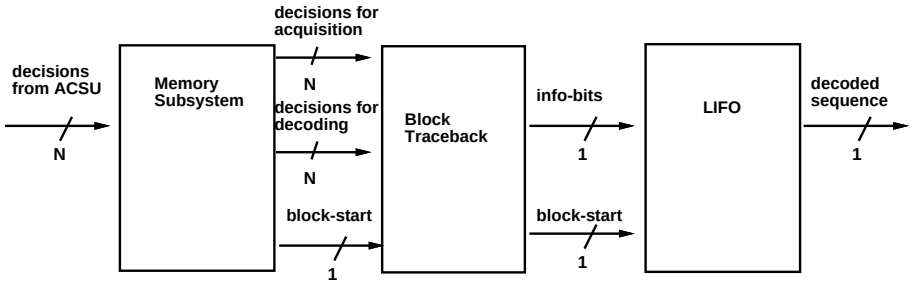

Figure 17.18 Architecture block diagram for traceback with $M = D$.

Fig. 17.18 shows a typical block diagram for a TBA with $M = D$ and one clock cycle per trellis step. Three building blocks are distinguished:
1) The memory subsystem including some control functionality and providing sets

of decision bits for acquisition and decoding, as well as a signal indicating the start of a new trace back.
2) The actual path reconstruction which outputs decoded information bits in reverse order and a block start indication.
3) The LIFO required to reverse the decoded information bits blockwise.
Typically, the memory subsystem dominates complexity and is affected strongly by the choice of M and D. By accepting the overhead implied by using dual port RAMs, a very simple architecture can be derived for $M = D$, that uses only 3 RAMs of size $N \cdot (D+1)$ bits. Fig. 17.19 shows the used addressing and multiplexing scheme for the 3 RAMs that repeats after 6 cycles. Using the dual port RAMs, a

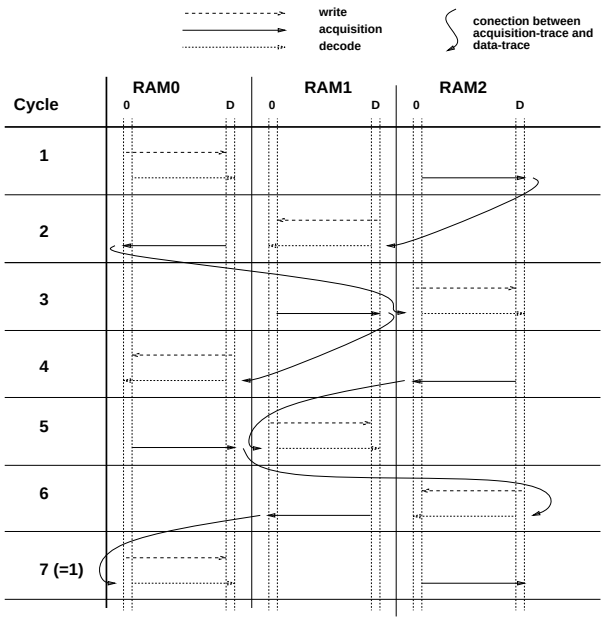

Figure 17.19 Cyclic addressing and multiplexing scheme for 3 dual port RAMs and $M = D$.

memory location is overwritten one clock cycle after it was read for the last time. This avoids concurrent read and write access to the same location, which is usually not possible. Consider e.g. cycle 3. All memories are accessed in ascending order. The first read access for acquisition, as well as data trace, is to address 1 of RAM1 and RAM2 respectively. Concurrently new data is written to address 0 of RAM2. Thus in the subsequent step the read data at address 1 of RAM2 is overwritten with new data. A closer look at the sequence of activity unvails that only a single address generator is needed, that counts up from 1 to D and subsequently down from $D-1$ to 0. The write address equals the read address of the last clock cycle. Fig. 17.20 shows the resulting architecture for the memory subsystem. The inputs *wIdle* and *rIdle* are the access control ports of the RAMs.

A reduction in the memory requirements is possible by choosing a smaller M. Fig. 17.21 shows an example for $M = 0.5 \cdot D$, which reduces latency and memory requirements to $3 \cdot D$ and $3 \cdot D \cdot N$ respectively. However, this amounts to the price

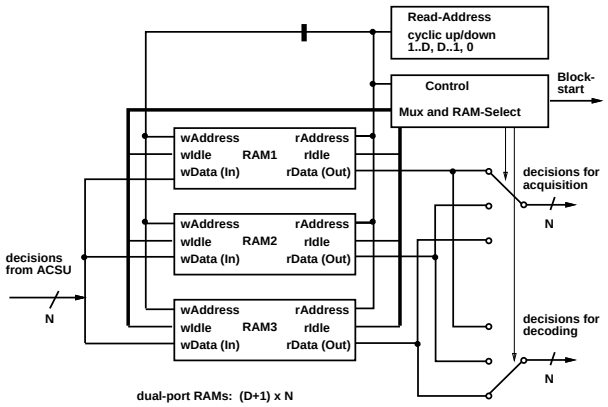

Figure 17.20 Architecture for the memory subsystem.

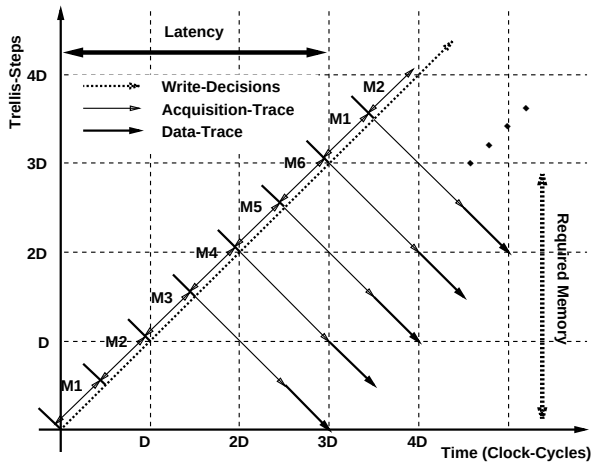

Figure 17.21 Traceback with $M = 0.5 \cdot D$.

of three concurrent pointers [37, 38]. More important, we need to slice the memory into blocks of depth $D/2$ trellis cycles (6 RAMs, denoted M1 ... M6 in the figure) rather than blocks of depth D (as in Fig. 17.17) to be able to access the required data, which complicates the architecture. Clearly, by choosing even smaller M the memory gets sliced more severely and the corresponding architecture soon becomes unattractive.

Tracing more than one trellis cycle per pointer and clock cycle has been considered for the case where more than one trellis step is decoded per clock cycle [33]. This can be done if the decisions from two subsequent trellis steps are stored in a single data word (or parallel memories) and effectively doubles the data wordlength of the RAMs while using half as many addresses. Since we can retrieve the information for two trellis steps in one clock cycle using this approach, the traceback can

evaluate two trellis steps per clock cycle, which leads to architectures with reduced memory size.

Fig. 17.22 shows the resulting clock-time/trellis-time diagram for the scheme proposed in [44] where acquisition-trace and data-trace are performed with even different speeds (cf [37]). Consider the first traceback in Fig. 17.22. While in every

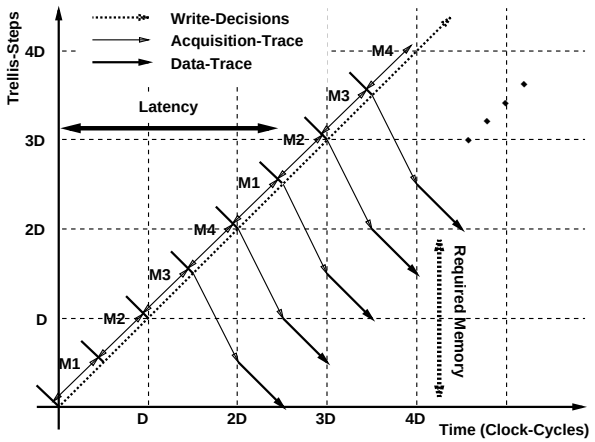

Figure 17.22 Dual Timescale Traceback with $M = 0.5 \cdot D$.

clock cycle two sets of decision bits ($2 \cdot N$ bits) are retrieved for acquisition-trace, two sets of decision bits are retrieved every other clock cycle for the data-trace. We can thus alternately retrieve the data for the data-trace and write a new set of decision bits to the same memory location, i.e. immediate memory reuse is facilitated. And since we need to read and write to one location within two cycles, this can actually be performed with commodity semi-custom technologies and single port RAMs. The obtained architecture exhibits a latency of $2.5 \cdot D$ clock cycles and we need only $2 \cdot D \cdot N$ bits of memory in four RAM-blocks with $D/4$ words of $2 \cdot N$ bits. The overall required memory is reduced, yet exchanging two sets of decision bits leads to increased wiring overhead. The approach is thus best suited to moderate N and large D, as given e.g. for punctured codes and $N = 64$.

As was pointed out, the TBA can be implemented using a wide range of architectures. The choice of the optimum architecture depends on many factors including technology constraints, throughput requirements and in some cases as well latency requirements. Of course software implementations are subject to different trade offs compared to hardware implementations and low throughput hardware implementations may well use $M = 1$ if power consumption is dominated by other components and the memory requirements need to be minimized.

17.5.4 Survivor depth

For actual dimensioning of the survivor depth D, $D = 5K$ was stated for rate 1/2 codes as a rule of thumb [6]. However, this figure is applicable only for best state decoding, i.e. if the overall best path is used for determining the final survivor as explained in section 17.2. For fixed state decoding, D must be chosen larger. In

[15, 16], it is reported that if D is doubled for fixed state decoding, even asymptotic performance losses are avoided[10].

For punctured codes, D also must be chosen larger than for the non punctured base code. In [45], $D = 96 = 13.5K$ was the reported choice for a $K = 7$ base code punctured to rate 7/8. For codes with rates smaller than 1/2, finally, D can be chosen smaller than $5K$.

Although theoretical results and simulation results available in the literature may serve as a guideline (e.g. [9, 46, 16, 15]), system simulations should be used to determine the required survivor depth for a certain SMU implementation and decoding scheme if figures are not available for the particular code and SMU architecture. System design and simulation tools like COSSAP^{TM}[4] are indispensable here. Simulation can be very useful as well if the performance of the Viterbi decoder (including the choice of D) can be traded against the performance of other (possibly less costly) parts of an overall system (cf [47]).

17.6 Synchronization of coded streams

As has been pointed out already, a step in the trellis is not generally associated with the transmission of a single channel symbol. Indeed, if punctured codes are used, the number of transmitted channel symbols per trellis step is time variant. Furthermore, channel symbols can exhibit ambiguities that cannot be resolved by synchronizers in the receiver front end. Consider the *simple* case of QPSK channel symbols in conjunction with a punctured code of rate 1/2.

trellis step	k	$k+1$	$k+2$	$k+3$
coded bits	$b_{1,k}$, $b_{2,k}$	$b_{1,k+1}$, $b_{2,k+1}$	$b_{1,k+2}$, $b_{2,k+2}$	$b_{1,k+3}$, $b_{2,k+3}$
punctured bits	$b_{1,k}$, $b_{2,k}$	$b_{1,k+1}$, $-$	$b_{1,k+2}$, $b_{2,k+2}$	$b_{1,k+3}$, $-$

Table 17.2 Puncturing of a rate 1/2 base code to a rate 2/3 code

QPSK Symbol	k	$k+1$	$k+3$
Inphase value	$I_1 = b_{1,k}$	$I_2 = b_{1,k+1}$	$I_3 = b_{2,k+2}$
Quadrature value	$Q_1 = b_{2,k}$	$Q_2 = b_{1,k+2}$	$Q_3 = b_{1,k+3}$

Table 17.3 Assignment of the punctured rate 3/4 code symbols to QPSK symbols

Clearly, 4 trellis cycles and 3 transmitted QPSK symbols are necessary to complete a mapping cycle. It has to be known how the blocks of 3 QPSK symbols are embedded in the received symbols stream to facilitate decoding. Furthermore, phase rotations of 90, 180 and 270 degrees of the QPSK symbols cannot be resolved by the receiver front end and at least the 90 degree rotation needs to be corrected prior to decoding[11]. Thus at least $2 \times 3 = 6$ possible ways of embedding the blocks

[10] This seems to be a pessimistic value, and the authors strongly recommend to run system simulations for a given application in order to determine the required value for the survivor depth.

[11] For many codes, including the (177,131) standard code, an inversion of the input symbol corresponds to valid code sequences associated with inverted information symbols. This inversion

of symbols in the received symbol stream need to be considered to find the state that is the prerequisite for the actual decoding.

The detection of the position/phase of the blocks of symbols in the received symbol stream and the required transformation of the symbol stream (i.e. rotating and/or delaying the received channel symbols) is called node synchronization in the case of convolutional coding. We call the different possible ways the blocks can be embedded in the received channel symbol stream *synchronization states*.

Node synchronization is essential for the reception of infinite streams of coded data, as applied for example in the current digital video satellite broadcasting standards. If data is transferred in frames (as in all cellular phone systems) the frame structure usually provides absolute timing and phase references that can be used to provide correctly aligned streams to the Viterbi decoder.

There are three approaches known to facilitate estimation of the correct synchronization state and thus node synchronization, which will be discussed below.

17.6.1 Metric growth based node synchronization

This approach was already suggested in the early literature on Viterbi decoding [3]. It is based on the fact that the path metrics in a decoder grow faster in a synchronized decoder compared to a decoder that is out-of-synch. However, the metric growth depends on the signal to noise ratio and the average input magnitude. This effect can substantially perturb the detection of changes in the synchronization state (and thus reacquisition) once the Viterbi decoder is correctly synchronized. Since more reliable approaches are known as described below, we do not consider this method further.

17.6.2 Node synchronization based on bit error rate estimation

This approach is based on the fact that a correctly synchronized Viterbi decoder computes an output data stream that contains much fewer errors than the input data stream. Thus the input data error rate can be estimated by re-encoding the output stream and comparing the generated sequence with the input sequence. Fig. 17.23 shows the resulting implementation architecture.

The received symbols are processed first in a block that can rotate and delay the input symbols as required for all possible trial synchronization states. The preprocessed stream is depunctured and decoded by a Viterbi decoder. The decoded stream is coded and punctured again. Additionally, the preprocessed symbols are sliced[12] to obtain the underlying hard decision bits which are then delayed according to the delay of the re-encoded data. The comparison result can be filtered to estimate the bit error rate of the input symbol stream. If the estimated error rate exceeds a certain level, a new trial with a new synchronization state is performed steered by some synchronization control functionality.

The approach has been implemented in several commercial designs. In HW implementations, the delay line can be of considerable complexity in particular if the Viterbi decoder exhibits a large decoding delay, i.e. for high rate codes and trace back based SMU architectures.

---

cannot be resolved without using other properties of the information sequence. Thus resolving the 90 degree ambiguity is sufficient for QPSK modulation in this case.

[12] Improved performance can be obtained by processing quantized symbols rather than hard decisions [48]. In this case, the required delay line is of course more costly.

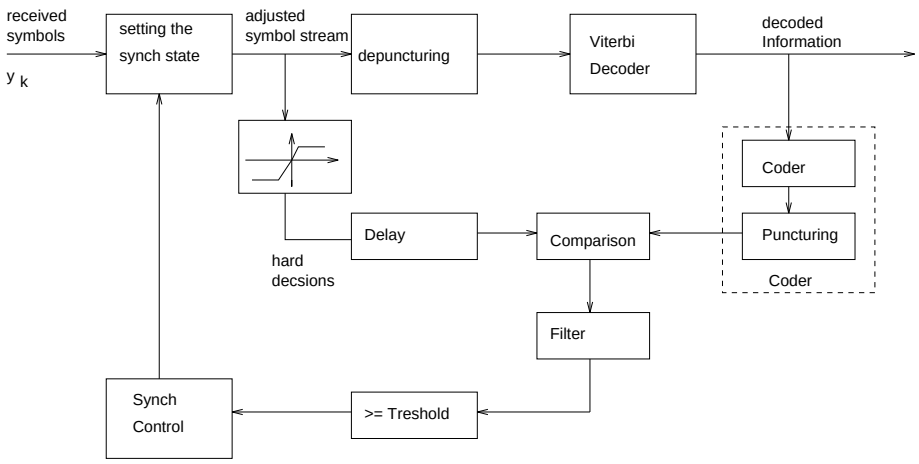

Figure 17.23 Node synchronization based on bit error rate estimation.

17.6.3 Syndrome based node synchronization

Syndrome based node synchronization was introduced as a robust high performance alternative to metric growth observation in [49].

The basic idea is depicted in Figures 17.24 and 17.25, assuming a code of rate 1/2 and BPSK transmission. All operations are performed on hard decisions in GF(2), and all signals are represented by transfer functions. The information bits u_k enter the coder, where convolution with the generator polynomials takes place. The code symbols $b_{1,k}$ and $b_{2,k}$ are then corrupted during transmission by adding the error sequences $e1_k$ and $e2_k$, respectively. In the receiver, another convolution of the received sequences with the swapped generator polynomials takes place. In practice, the values of in the receiver are calculated by slicing and de–mapping the quantized received symbols y_k.

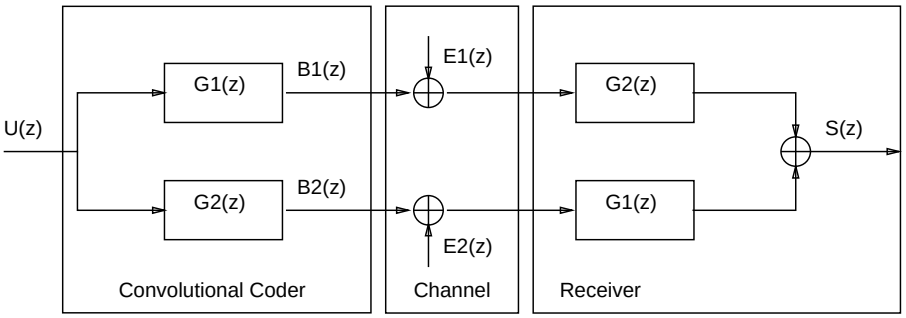

Figure 17.24 Syndrome computation in in–sync state.

From Fig. 17.24 it is easily seen that for the Z–transform of the syndrome

$$S(z) = E1(z) \cdot G2(z) + E2(z) \cdot G1(z) + 2U(z) \cdot G1(z) \cdot G2(z) \qquad (26)$$

holds. If the channel error sequences $e1_k$ and $e2_k$ and the corresponding Z–transforms are zero, respectively, the syndrome sequence s_k is also zero, since $2U(z) = 0$ holds in GF(2). Therefore, the syndrome sequence depends only on the channel error sequences $E1(z), E2(z)$. For reasonable channel error rates, the rate of ones in the syndrome stream s_k is lower than 0.5.

Fig. 17.25 shows the effect of an additional reception delay, i.e. an out-of-synch condition for the Viterbi decoder.

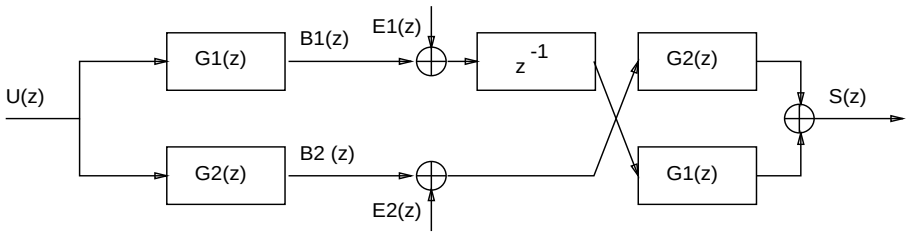

Figure 17.25 Syndrome computation in out–of–sync state.

Now $S(z)$ depends clearly on $U(z)$ as well as on the channel error sequences since

$$S(z) = E1(z) \cdot Z^{-1} \cdot G1(z) + E2(z) \cdot G2(z) + U(z) \cdot (G1^2(z) \cdot Z^{-1} + G2^2(z)) \quad (27)$$

holds. Now, $S(z)$ essentially consists of equiprobable ones and zeros, i.e. the rate of ones in the syndrome stream is 0.5. Thus an estimation of the actual rate of ones in the syndrome serves as a measure to decide whether the actual trial corresponds to an in-synch condition.

A strong advantage of syndrome based node synchronization is the complete independence of the subsequent Viterbi decoder. The involved hardware complexity is sufficiently low to enable the implementation of synchronizers that concurrently investigate all possible synchronization states, which is not economically feasible with other approaches. However, the parameters and syndrome polynomials are more difficult to determine as the parameters of the approach based on bit error rate estimation. In particular, a poor choice of the syndrome polynomials can seriously degrade performance [50]. For a detailed discussion, the reader is referred to [49, 51] for rate 1/2 codes, [52, 53, 54] for rate 1/N codes, and [50] for rate (N-1)/N codes.

17.7 Recent developments

Viterbi decoding as discussed in this chapter is applicable to all current commercial applications of convolutional codes, although the basic algorithms need to be extended in some applications (e.g. if applied to trellis coded modulation with parallel branches in the trellis [10]). It is common to all these applications that the decoder needs to compute an information sequence only (hard output decoding).

However, it has already been shown in [11] that a concatenation of several codes can provide a better overall cost/performance trade off. In fact serial concatenation (i.e. coding a stream twice subsequently with different codes) has been

chosen for deep space communcation and digital video broadcasting. While decoding such codes is possible (and actually done) by decoding the involved (component) codes independently, improved performance can be obtained by passing additional information between the decoders of the component codes [55, 56].

In fact, the most prominent class of recently developed codes, the so called *TURBO codes* [57] can no longer be decoded by decoding the component codes independently. Decoding these codes is an iterative process for which in addition to the information sequences, reliability estimates for each information symbol need to be computed (soft outputs). Thus decoding algorithms for convolutional component codes that provide soft outputs have recently gained considerable attention. Large increases in coding gain are possible for concatenated or iterative (TURBO) decoding systems [58, 59]. The most prominent soft output decoding algorithm is the soft output Viterbi algorithm (SOVA) [60, 61], which can be derived as an approximation to the optimum symbol by symbol detector, the symbol by symbol MAP algorithm (MAP)[13]. The basic structure of the Viterbi algorithm is maintained for the SOVA. Major changes are necessary in the SMU, since now, a soft quantized output has to be calculated rather than decoding an information sequence. Efficient architectures and implementations for the SOVA were presented in [63, 64, 65, 62].

In the MAP algorithm, a posteriori probabilities are calculated for every symbol, which represent the optimum statistical information which can be passed to a subsequent decoding stage as a soft output. Although the MAP was already derived in [66, 2], this was recognized only recently [60]. In its original form, the MAP algorithm is much more computationally intensive than the Viterbi algorithm (VA) or the SOVA. However, simplifications are known that lead to algorithms with reduced implementation complexity [67, 68, 69, 70], It was shown in [71] that acquisition and truncation properties can be exploited for the MAP as for the VA and the SOVA. Thereby, efficient VLSI architectures for the MAP can be derived for recursive [72] and parallelized [73] implementations with implementation complexities roughly comparable to the SOVA and the VA [71].

REFERENCES

[1] A. J. Viterbi, "Error bounds for convolutional coding and an asymptotically optimum decoding algorithm," *IEEE Trans. Information Theory*, vol. IT-13, pp. 260–269, April 1967.

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

[3] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding.* New York: McGraw-Hill, 1979.

[4] "COSSAP Overview and User Guide." Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043.

[13] In fact, it has been shown [62] for the slightly differing algorithms in [60] and [61] that the algorithm from [60] is slightly too optimistic while the algorithm from [61] is slightly too pessimistic compared to a derived approximation on the MAP. However, the performance of either approach seems to be almost identical.

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

[6] G. Clark and J. Cain, *Error-Correction Coding for Digital Communications.* New York: Plenum, 1981.

[7] H. Meyr and R. Subramanian, "Advanced digital receiver principles and technologies for PCS," *IEEE Communications Magazine*, vol. 33, no. 1, pp. 68–78, January 1995.

[8] M. Vaupel, U. Lambrette, H. Dawid, O. Joeressen, S. Bitterlich, H. Meyr, F. Frieling, and K. Müller, "An All–Digital Single–Chip Symbol Synchronizer and Channel Decoder for DVB," in *Proc. IX IFIP International Conference on Very Large Scale Integration*, IFIP, Chapman-Hall, 1997. accepted paper.

[9] J. A. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Transactions on Communications*, vol. COM-19, no. No. 5, pp. 835–848, Oct. 1971.

[10] G. Ungerboek, "Trellis Coded Modulation with Redundant Signal Sets, Parts I+II," *IEEE Communications Magazine*, vol. 25, no. 2, pp. 5–21, 1987.

[11] G. D. Forney, Jr., *Concatenated Codes.* Cambridge, MA: MIT Press, 1966. MIT Reasearch Monograph.

[12] J. Cain, J. G.C. Clark, and J. Geist, "Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding," *IEEE Transactions on Information Theory*, vol. IT-25, no. 1, pp. 97–100, Jan. 1979.

[13] Y. Yasuda, K. Kashiki, and Y. Hirata, "High-rate punctured convolutional codes for soft decision," *IEEE Transactions on Communications*, vol. COM-32, no. 3, pp. 315–319, March 1984.

[14] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389–400, April 1988.

[15] I. Onyszchuk, "Truncation Length for Viterbi Decoding," *IEEE Transactions on Communications*, vol. 39, no. 7, pp. 1023–1026, July 1991.

[16] R. J. McEliece and I. M. Onyszchuk, "Truncation effects in Viterbi decoding," in *Proceedings of the IEEE Conference on Military Communications*, (Boston, MA), pp. 29.3.1–29.3.3, October 1989.

[17] G. Fettweis and H. Meyr, "A 100 Mbit/s Viterbi decoder chip: Novel architecture and its realisation," in *IEEE International Conference on Communications, ICC'90*, vol. 2, (Atlanta, GA, USA), pp. 463–467, Apr. 1990.

[18] G. Fettweis and H. Meyr, "Parallel Viterbi decoding: Algorithm and VLSI architecture," *IEEE Communications Magazine*, vol. 29, no. 5, pp. 46–55, May 1991.

[19] O. M. Collins, "The subtleties and intricacies of building a constraint length 15 convolutional decoder," *IEEE Transactions on Communications*, vol. 40, no. 12, pp. 1810–1819, December 1992.

[20] C. Shung, P. Siegel, G. Ungerböck, and H. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proceedings of the IEEE International Conference on Communications*, pp. 1723–1528, IEEE, 1990.

[21] P. Siegel, C. Shung, T. Howell, and H. Thapar, "Exact bounds for Viterbi detector path metric differences," in *Proceedings of the International Conference on Acoustics Speech and Signal Processing*, pp. 1093–1096, IEEE, 1991.

[22] T. Noll, "Carry–Save Architectures for High–Speed Digital Signal Processing," *Journal of VLSI Signal Processing*, vol. 3, no. 1/2, pp. 121–140, June 1991.

[23] T. Noll, "Carry-Save Arithmetic for High-Speed Digital Signal Processing," in *IEEE ISCAS'90*, vol. 2, pp. 982–986, 1990.

[24] J. Sparsø, H. Jorgensen, P. S. Pedersen, and T. R"ubner-Petersen, "An area-efficient topology for vlsi implementation of the viterbi decoders and other shuffle exchange type structures," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 2, pp. 90–97, February 1991.

[25] C. Rader, "Memory Management in a Viterbi Decoder," *IEEE Transactions on Communications*, vol. COM-29, no. 9, pp. 1399–1401, Sept. 1981.

[26] H. Dawid, S. Bitterlich, and H. Meyr, "Trellis Pipeline–Interleaving: A novel method for efficient viterbi decoder implementation," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, (San Diego, CA), pp. 1875–78, IEEE, May 10-13 1992.

[27] S. Bitterlich and H. Meyr, "Efficient scalable architectures for Viterbi decoders," in *International Conference on Application Specific Array Processors (ASAP), Venice, Italy*, October 1993.

[28] S. Bitterlich, H. Dawid, and H. Meyr, "Boosting the implementation efficiency of Viterbi Decoders by novel scheduling schemes," in *Proceedings IEEE Global Communications Conference GLOBECOM 1992*, (Orlando, Florida), pp. 1260–65, December 1992.

[29] C. Shung, H.-D. Lin, P. Siegel, and H. Thapar, "Area-efficient architectures for the viterbi algorithm," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, 1990.

[30] K. K. Parhi, "Pipeline interleaving and parallelism in recursive digital filters – parts 1&2," *IEEE transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 7, pp. 1099–1134, July 1989.

[31] G. Fettweis and H. Meyr, "Feedforward architectures for parallel Viterbi decoding," *Journal on VLSI Signal Processing*, vol. 3, no. 1/2, pp. 105–120, June 1991.

[32] G. Fettweis and H. Meyr, "Cascaded feedforward architectures for parallel Viterbi decoding," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 978–981, May 1990.

[33] P. Black and T. Meng, "A 140MBit/s, 32-State, Radix-4 Viterbi Decoder," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, December 1992.

[34] H. Dawid, G. Fettweis, and H. Meyr, "A CMOS IC for Gbit/s Viterbi Decoding," *IEEE Transactions on VLSI Systems*, no. 3, March 1996.

[35] H. Dawid, G. Fettweis, and H. Meyr, "System Design and VLSI Implementation of a CMOS Viterbi Decoder for the Gbit/s Range," in *Proc. ITG-Conference Mikroelektronik für die Informationstechnik*, (Berlin), pp. 293–296, ITG, VDE-Verlag, Berlin Offenbach, March 1994.

[36] G. Fettweis, H. Dawid, and H. Meyr, "Minimized method Viterbi decoding: 600 Mbit/s per chip," in *IEEE Globecom 90*, (San Diego, USA), pp. 1712–1716, Dec. 1990.

[37] R. Cypher and C. Shung, "Generalized Trace Back Techniques for Survivor Memory Management in the Viterbi Algorithm," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, (San Diego, California), pp. 707A.1.1–707A.1.5, IEEE, Dec. 1990.

[38] G. Feygin and P. G. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Transactions on Communications*, vol. 41, no. 3, pp. 425–429, March 1993.

[39] 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*, vol. 12, no. 2, pp. 79–91, November 1991.

[40] P. Black and T. Meng, "Hybrid survivor path architectures for Viterbi decoders," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 433–436, IEEE, 1993.

[41] G. Fettweis, "Algebraic survivor memory management for viterbi detectors," in *Proceedings of the IEEE International Conference on Communications*, (Chicago), pp. 339–343, IEEE, June 1992.

[42] T. Ishitani, K. Tansho, N. Miyahara, S. Kubota, and S. Kato, "A Scarce-State-Transition Viterbi-Decoder VLSI for Bit Error Correction," *IEEE Journal of Solid-State Ciruits*, vol. SC-22, no. 4, pp. 575–581, August 1987.

[43] K. Kawazoe, S. Honda, S. Kubota, and S. Kato, "Ultra-high-speed and Universal-coding-rate Viterbi Decoder VLSIC –SNUFEC VLSI–," in *Proceedings of the IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 1434–1438, IEEE, May 1993.

[44] O. J. Joeressen and H. Meyr, "Viterbi decoding with dual timescale trace-back processing," in *Proceedings of the IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*, (Toronto), pp. 213–217, September 1995.

[45] R. Kerr, H. Dehesh, A. Bar-David, and D. Werner, "A 25 MHz Viterbi FEC Codec," in *Proceeding of the 1990 IEEE Custom Integrated Circuit Conference*, (Boston, MA), pp. 16.6.1–13.6.5, IEEE, May 1990.

[46] F. Hemmati and D. Costello, "Truncation Error Probability in Viterbi Decoding," *IEEE Transactions on Communications*, vol. 25, no. 5, pp. 530–532, May 1977.

[47] O. J. Joeressen, G. Schneider, and H. Meyr, "Systematic Design Optimization of a Competitive Soft-Concatenated Decoding System," in *VLSI Signal Processing VI* (L. D. J. Eggermont, P. Dewilde, E. Deprettere, and J. van Meerbergen, eds.), pp. 105–113, IEEE, 1993.

[48] U. Mengali, R. Pellizoni, and A. Spalvieri, "Phase ambiguity resolution in trellis-codes modulations," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2532–2539, September 1995.

[49] G. Lorden, R. McEliece, and L. Swanson, "Node synchronization for the Viterbi decoder," *IEEE Transactions on Communications*, vol. COM-32, no. 5, pp. 524–31, May 1984.

[50] O. J. Joeressen and H. Meyr, "Node synchronization for punctured convolutional codes of rate (n-1)/n," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, (San Francisco, CA), pp. 1279–1283, IEEE, November 1994.

[51] M. Moeneclaey, "Syndrome-based Viterbi decoder node synchronization and out-of-lock detection," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, (San Diego, CA), pp. 604–8, Dec 1990.

[52] M.-L. de Mateo, "Node synchronization technique for any 1/n rate convolutional code," in *Proceedings of the IEEE International Conference on Communications*, (Denver, CO), pp. 1681–7, June 1991.

[53] J. Sodha and D. Tait, "Soft-decision syndrome based node synchronisation," *Electronics Letters*, vol. 26, no. 15, pp. 1108–9, July, 19 1990.

[54] J. Sodha and D. Tait, "Node synchronisation for high rate convolutional codes," *Electronics Letters*, vol. 28, no. 9, pp. 810–12, April, 23 1992.

[55] E. Paaske, "Improved decoding for a concatenated coding scheme recommended by CCSDS," *IEEE Transactions on Communications*, vol. 38, no. 8, pp. 1138–1144, August 1990.

[56] J. Hagenauer, E. Offer, and L. Papke, "Improving the standard coding system for deep space missions," in *Proceedings of the IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 1092–1097, IEEE, May 1993.

[57] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, October 1996.

[58] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Seperable MAP 'Filters' for the decoding of product and concatenated codes," in *Proceedings of the IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 1740–1745, IEEE, May 1993.

[59] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: TURBO-Codes," in *Proceedings of the IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 1064–1070, IEEE, May 1993.

[60] J. Hagenauer and P. Höher, "A Viterbi Algorithm with Soft Outputs and It's Application," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM*, pp. 47.1.1–47.1.7, Nov. 1989.

[61] J. Huber and A. Rüppel, "Zuverlässigkeitsschätzung für die Ausgangssymbole von Trellis-Decodern," *Archiv für Elektronik und Übertragung (AEÜ)*, vol. 44, no. 1, pp. 8–21, Jan. 1990. in German.

[62] O. Joeressen, *VLSI-Implementierung des Soft-Output Viterbi-Algorithmus.* VDI-Fortschritt-Berichte, Reihe 10, Nr. 396, Düsseldorf: VDI-Verlag, 1995. ISBN 3-18-339610-6, (in german).

[63] comatlas sa, Chateubourg, France, *CAS5093, Turbo-Code Codec, Technical Data Sheet*, April 1994.

[64] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A Low Complexity Soft-Output Viterbi Decoder Architecture," in *Proceedings of the IEEE International Conference on Communications*, (Geneva, Switzerland), pp. 737–740, IEEE, May 1993.

[65] O. J. Joeressen and H. Meyr, "A 40Mbit/s soft output Viterbi decoder," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 7, pp. 812–818, July 1995.

[66] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. IT-24, no. 3, pp. 284–287, March 1974.

[67] P. Höher, *Kohärenter Empfang trelliscodierter PSK Signale auf frequenzselektiven Mobilfunkkanälen.* VDI-Fortschritt-Berichte, Reihe 10, Nr. 147, Düsseldorf: VDI-Verlag, 1990. ISBN 3-18-144710-2.

[68] G. Ungerböck, "Nonlinear Equalization of binary signals in gaussian noise," *IEEE Trans. Communications*, no. COM-19, pp. 1128–1137, 1971.

[69] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced Complexity Symbol Detectors with Parallel Structures for ISI Channels," *IEEE Transactions Communications*, vol. 42, no. 2,3,4, pp. 1661–1671, Feb./March/April 1994.

[70] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proceedings of the IEEE International Conference on Communications*, (Seattle, WA), 1995.

[71] H. Dawid, *Algorithmen und Schaltungsarchitekturen zur Maximum A Posteriori Faltungsdecodierung*. Aachen: Shaker-Verlag, 1996. ISBN 3-8265-1540-4.

[72] H. Dawid and H. Meyr, "Real–Time Algorithms and VLSI Architectures for Soft Output MAP Convolutional Decoding," in *Proceedings of the IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications*, (Toronto), pp. 193–197, September 1995.

[73] H. Dawid, G. Gehnen, and H. Meyr, "MAP Channel Decoding: Algorithm and VLSI Architecture," in *VLSI Signal Processing VI* (L. D. J. Eggermont, P. Dewilde, E. Deprettere, and J. van Meerbergen, eds.), pp. 141–149, IEEE, 1993.