Chapter 6
Turbo Codes

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

6.1 Encoder Structure

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

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

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

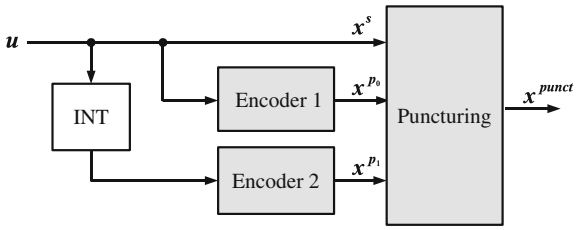


Fig. 6.1 Generic parallel turbo encoder structure with puncturing unit

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

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

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

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

Table 6.1 Turbo codes in communication standards

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

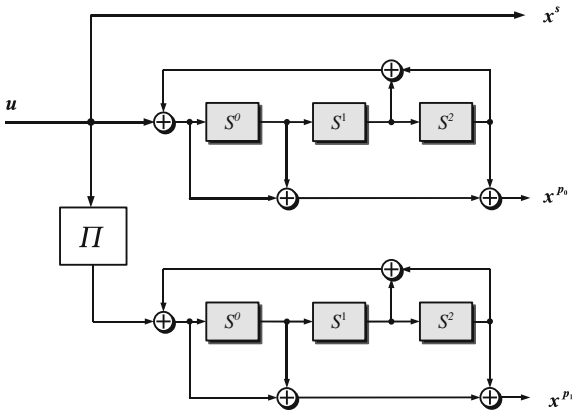


Fig. 6.2 3GPP turbo encoder with 8-state RSC component encoder

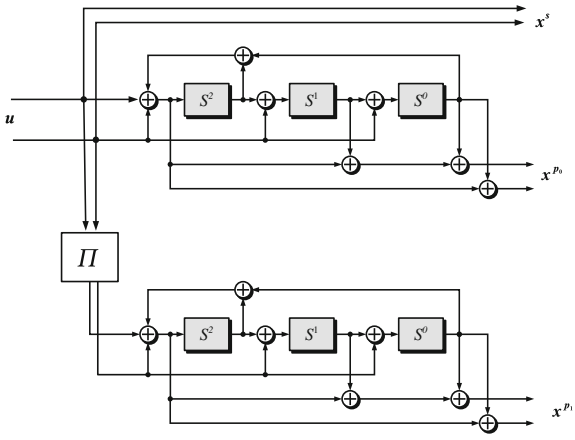


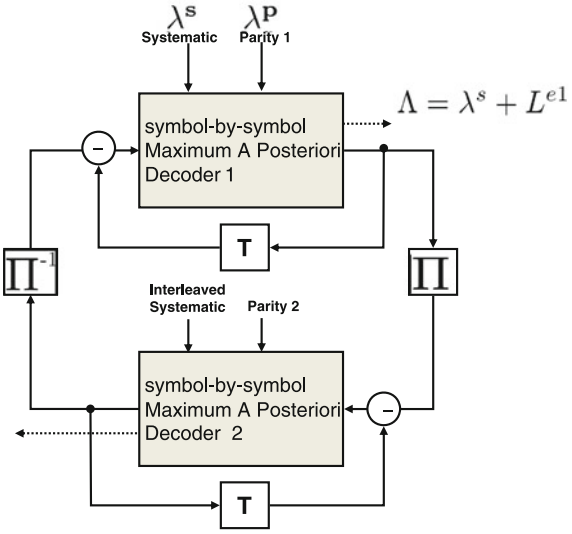
Fig. 6.3 WiMAX duo-binary encoder with 8-state RSC component encoder

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

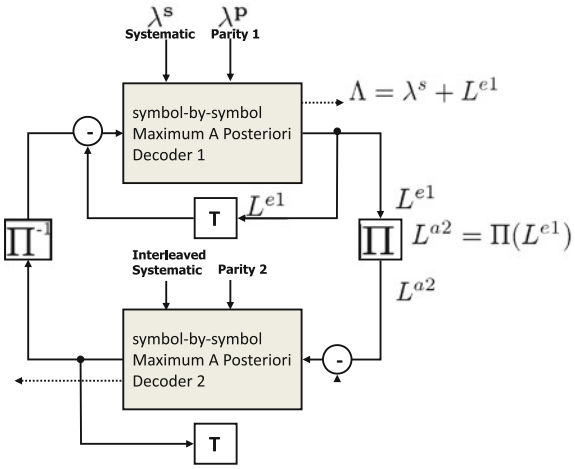
Table 6.1 shows 8 different standards featuring turbo codes. All have a parallel turbo code encoder structure with two component encoders. The turbo code decoding of all theses standards share the same basic iterative exchange of messages. This basic iterative decoding procedure is described next section.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

6.2 The Iterative Decoding Procedure

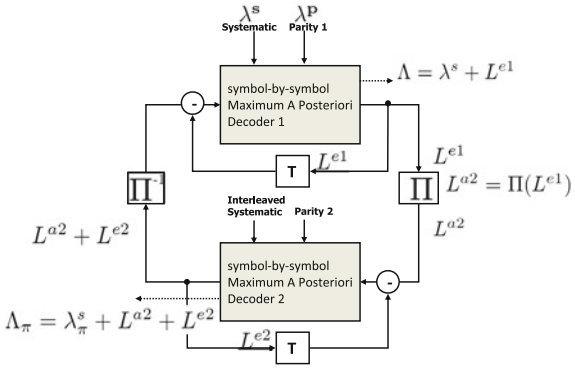
In this section the iterative decoding procedure is described step-by-step. The decoder structure presented here can decode any parallel concatenated turbo code which consist of two component codes. Thus, it fits to all encoders of Table 6.1. This section presents one possible decoder structure. Obviously,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



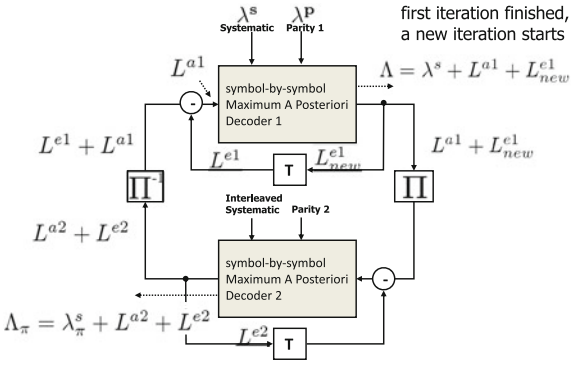
For each component encoder a corresponding component decoder exists. Decoder 1 will consume the received systematic information $\boldsymbol{\lambda}^s$ and the corresponding parity LLRs $\boldsymbol{\lambda}^{p1}$. Decoder one starts to calculate new a posteriori probability information for each systematic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



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



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



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

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

6.2.1 Convergence Progress (EXIT Charts)

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

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

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

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

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

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

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

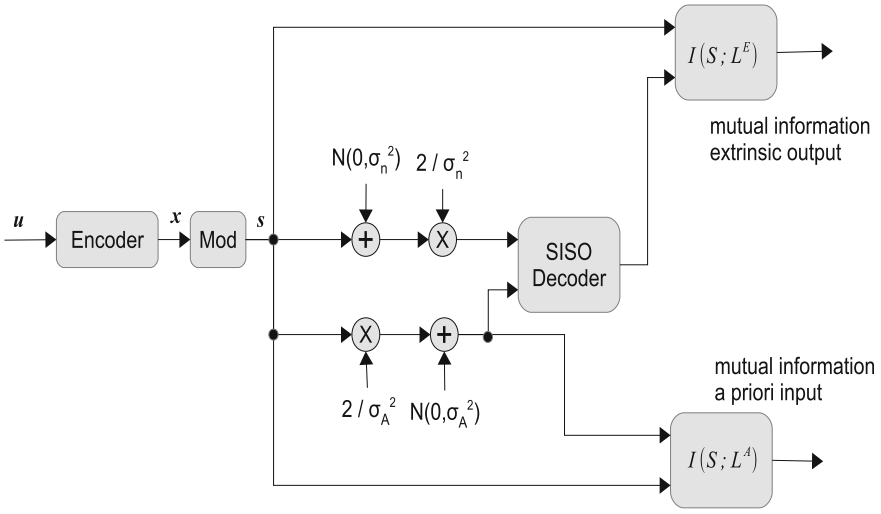


Fig. 6.4 Measurement of the EXIT chart

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

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

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

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

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

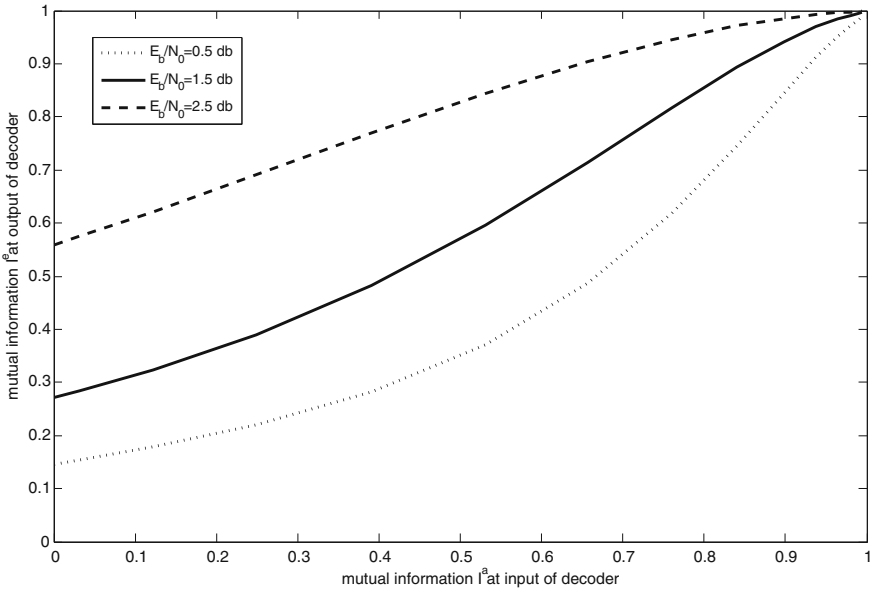


Fig. 6.5 Extrinsic information transfer characteristics of soft-in soft-out component convolutional decoder (8-state, R = 1/3, LTE)

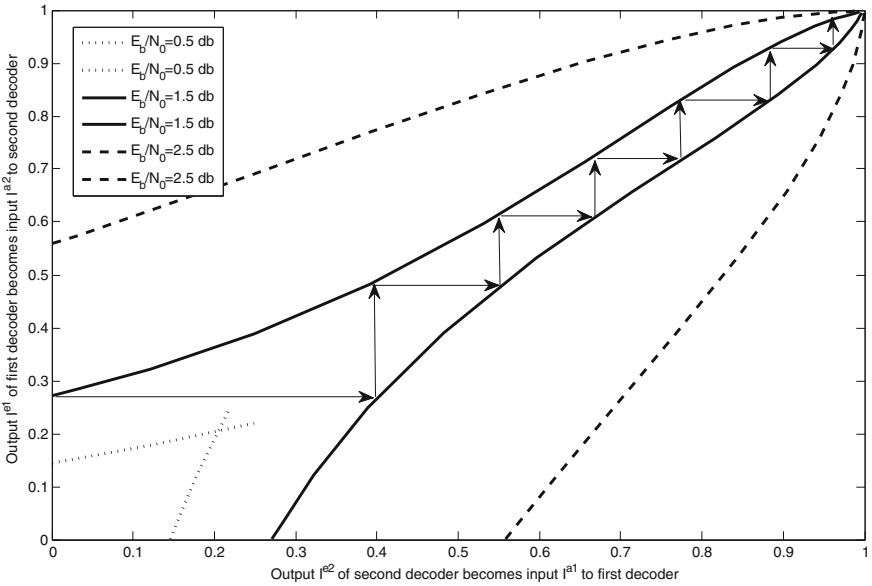


Fig. 6.6 EXIT chart characteristics for three different E_b/N_0 values. The sketched trajectory is only valid for an infinite interleaver size

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

6.2.2 Communications Performance

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

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

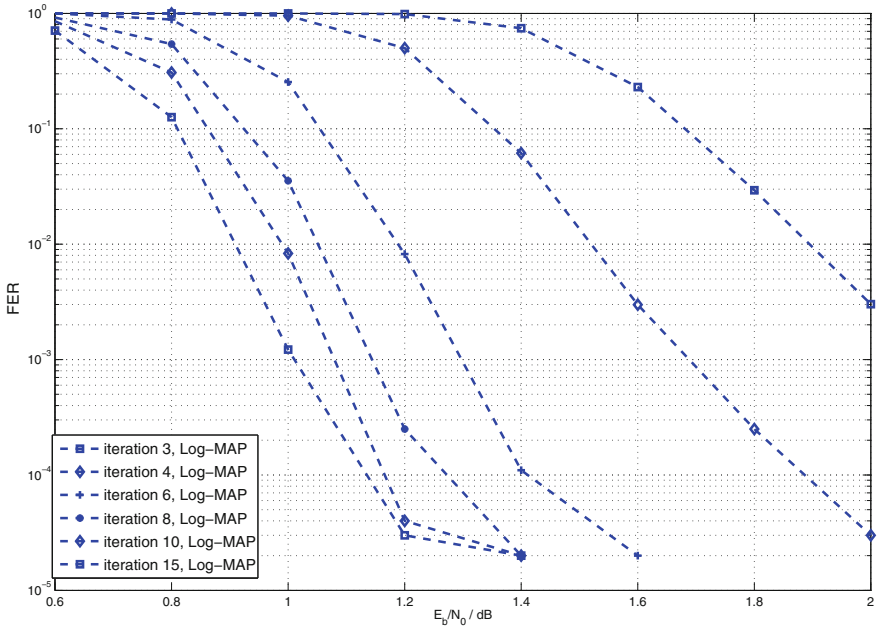


Fig. 6.7 LTE turbo code performance for code rate $R = 0.5$ and a block length of $K = 6144$ [4]

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

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

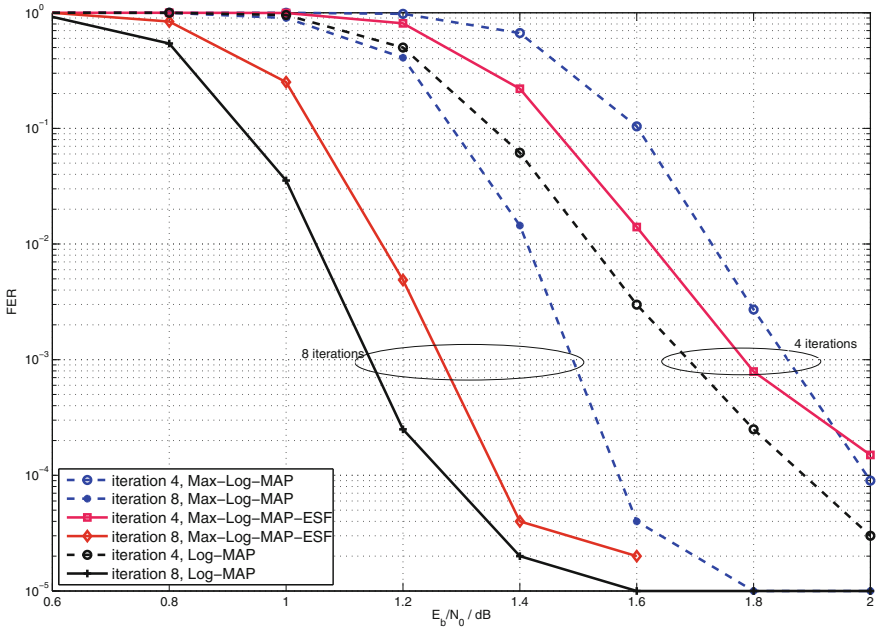


Fig. 6.8 Comparing Max-Log MAP, Log-MAP, Max-Log-MAP-ESF communications performance

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

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

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

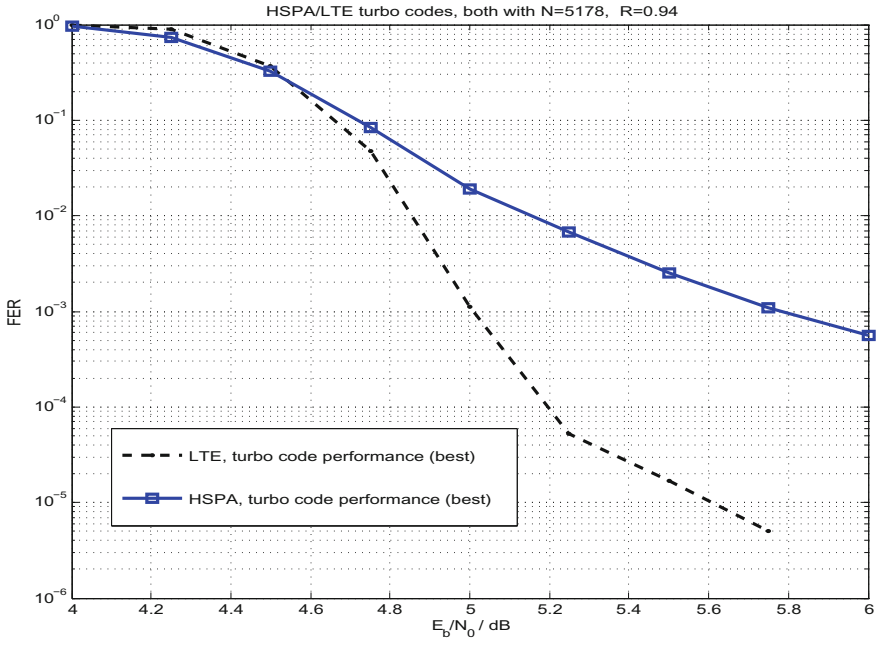


Fig. 6.9 HSPA and LTE turbo code performance for code rate $R = 0.94$ and a block length of $N = 5178$ [4]

6.2.3 Fixed-Point Realization and Robustness

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

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

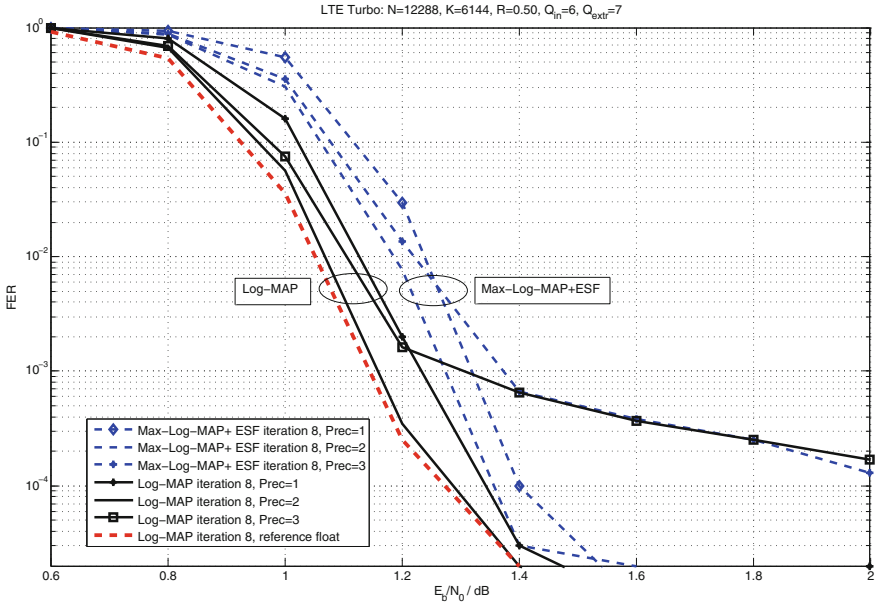


Fig. 6.10 LTE turbo code performance for code rate $R = 0.5$ and a block length of $K = 6144$ bits. All results with 6 bits input width, while the precision is varying

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

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

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

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

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

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

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

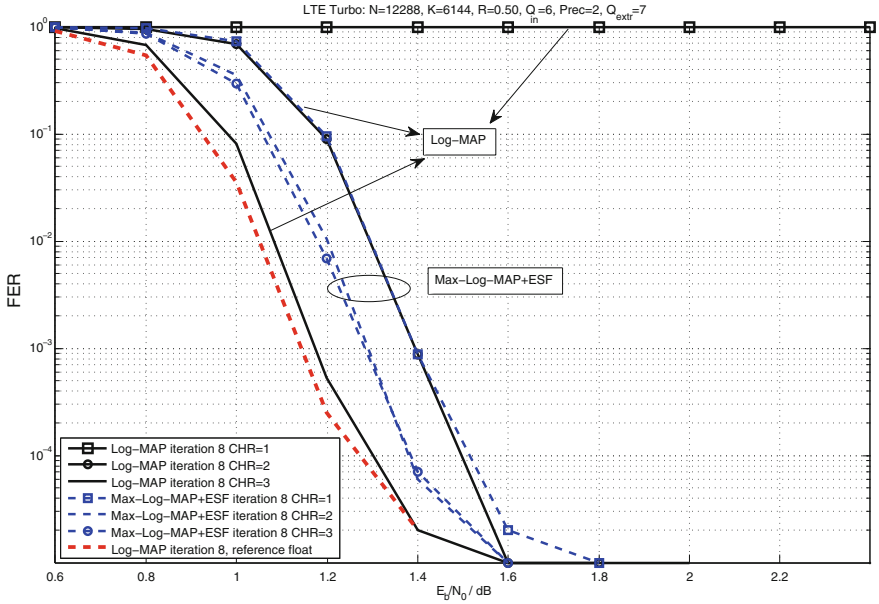


Fig. 6.11 LTE turbo code performance ($R = 0.5$, $K = 6144$). All results with 6 bits input width and 2 precision bits. The scaling of the input LLRs provided by the demodulator is varying

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

6.3 Turbo Codes Architecture

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

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

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

6.3.1 Serial MAP Architecture

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

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

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

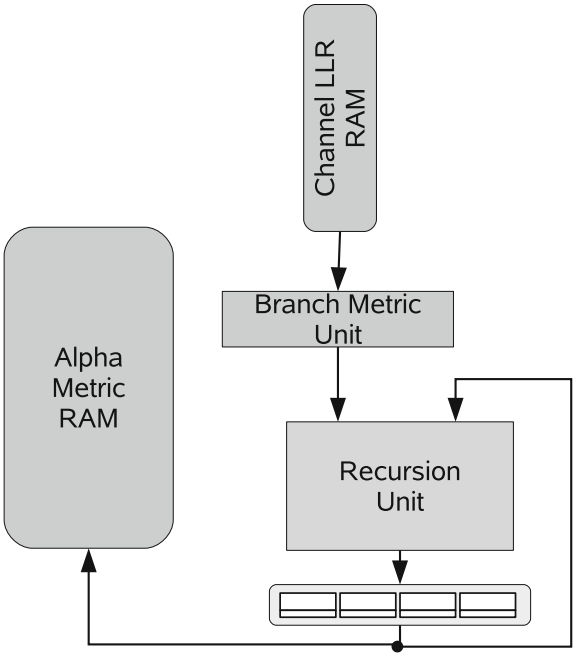


Fig. 6.12 SMAP architecture during forward recursion

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

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

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

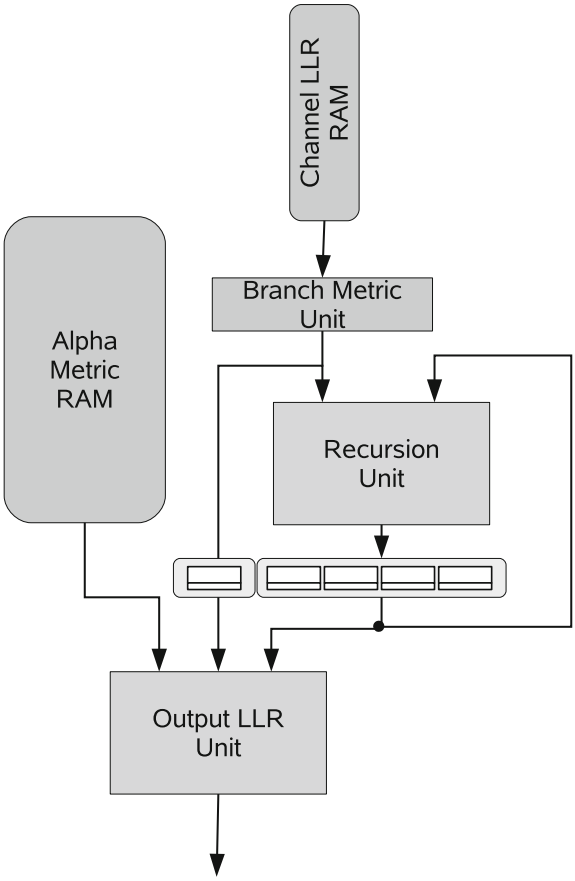


Fig. 6.13 SMAP architecture during backward recursion and output processing

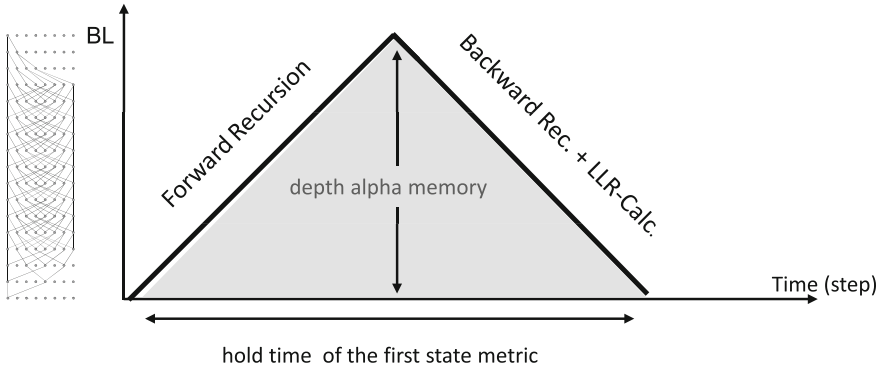


Fig. 6.14 Graphical data life time analysis. y axis reflects the position in a data block of length BL, the x axis reflects the time or trellis steps

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

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

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

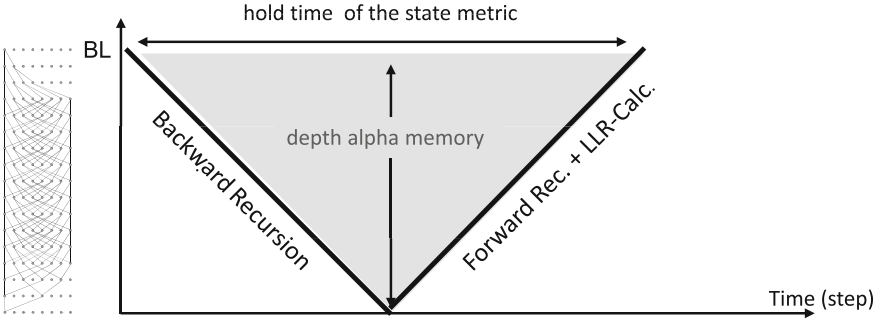


Fig. 6.15 Processing schedule: first the backward recursion then the forward recursion with output LLR processing

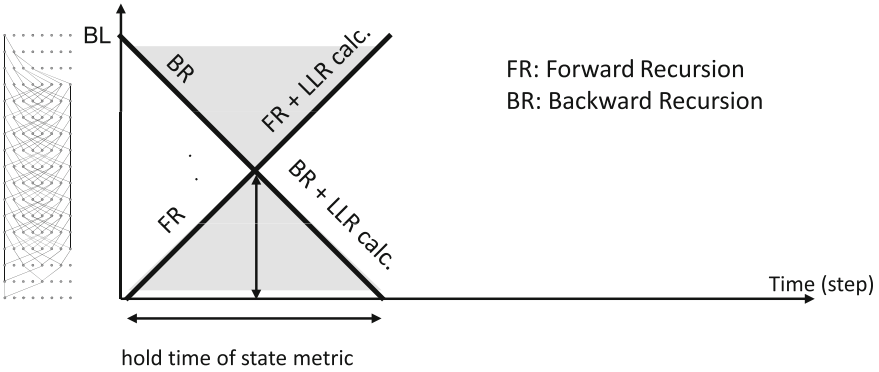


Fig. 6.16 Two recursions are running in parallel, forward and backward

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

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

6.3.2 Design Space and Design Choices

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

High Level Architecture Decisions

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

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

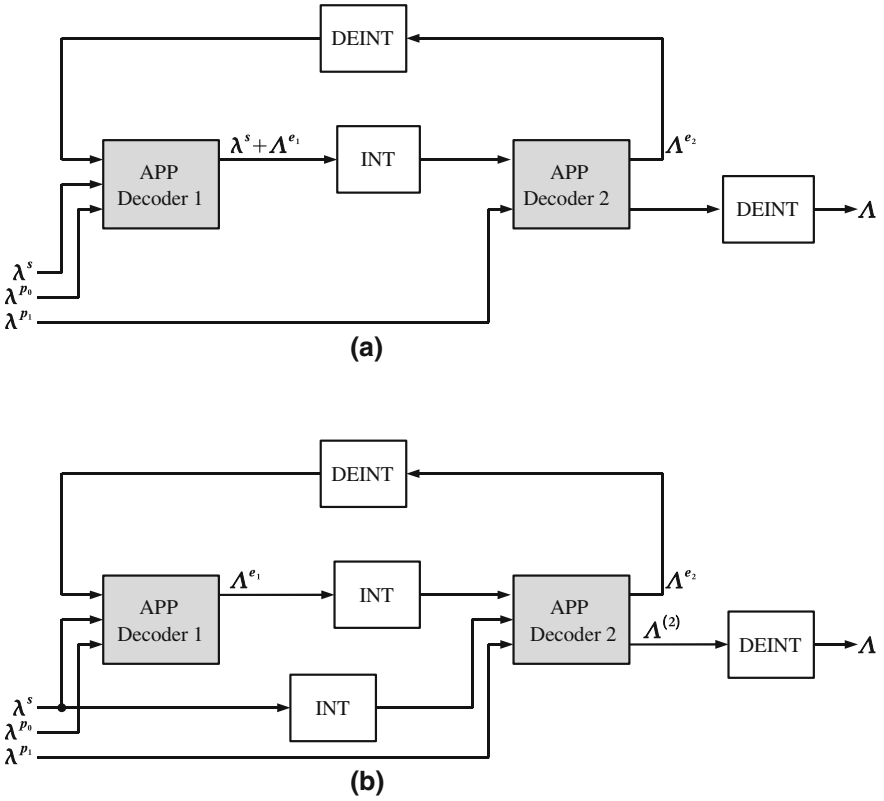


Fig. 6.17 **a** Turbo decoder which passes the full a posteriori information from MAP1 to MAP2, **b** with separate interleaved systematic information

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

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

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

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

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

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

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

Quantization Issues

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

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

Data Path MAP Component

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

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

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

Windowing Scheme

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

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

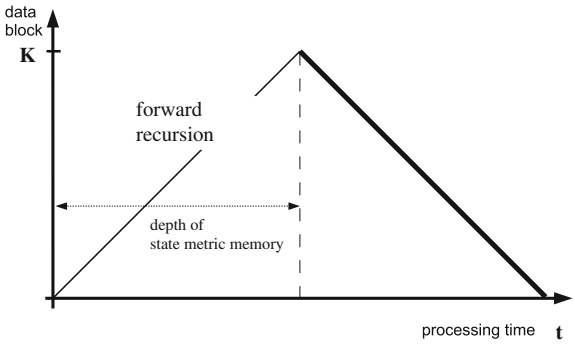


Fig. 6.18 MAP decoder which processes a full block of length K

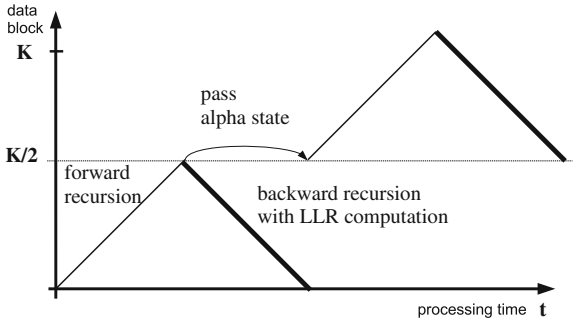


Fig. 6.19 MAP decoder which processes two windows, each with K/2 data

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

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

Recursion Unit

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

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

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

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

Parallel Interleaving

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

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

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

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

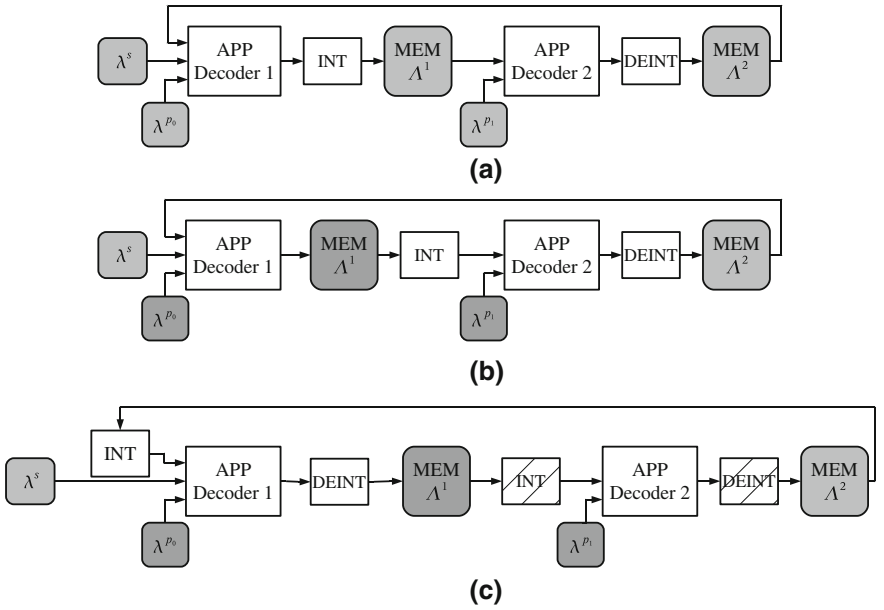


Fig. 6.20 Storage and interleaving of exchanged soft-values: **a** interleaving and deinterleaving upon write; **b** interleaving upon read, deinterleaving upon write, **c** two stage interleaving process

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

Low Power Techniques

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

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

6.3.3 Dependencies of Architectural Parameters

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

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

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

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

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

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

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

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

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

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

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

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

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

The power consumption results in

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

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

References

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

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