# Implementation and Performance of Parallelised Turbo Decoders

Enes Erdin

TUBITAK Defense Industries Research and Development Institute

Digital Electronics Design Division, Ankara, Turkey

Email: eerdin@sage.tubitak.gov.tr

Çağlar Kılcıoğlu

ASELSAN Electronics Industries Inc.

Communications Division, Ankara, Turkey

Email: ckilcioglu@aselsan.com.tr

Ali Özgür Yılmaz

Department of Electrical and Electronics Engineering

Middle East Technical University, Ankara, Turkey

Email: aoyilmaz@metu.edu.tr

**Abstract**

In this paper, we discuss the implementation of a low latency decoding algorithm for turbo codes and repeat accumulate codes and compare the implementation results in terms of maximum available clock speed, resource consumption, error correction performance, and the data (information bit) rate. In order to decrease the latency a parallellised decoder structure is introduced for these mentioned codes and the results are obtained by implementing the decoders on a field programmable gate array. The memory collision problem is avoided by using collision-free interleavers. Through a proposed quantisation scheme and normalisation in forward/backward recursions, computational issues are handled for overcoming the overflow and underflow issues in a fixed point arithmetic. Also, the effect of different implementation styles are observed.

**Index Terms**

Turbo codes, repeat accumulate codes, parallellised turbo decoding, BCJR, FPGA, metric quantisation.

## I. Introduction

In wireless communications, channel coding has an important role on enhancing the communication reliability and quality of service. Turbo codes, introduced in [1], have shown a great performance in additive white Gaussian (AWGN) channels. After the great achievement of the turbo structure, the same idea has been applied to different coding schemes. Turbo-like codes, explained in detail in [2], is the result of that idea. Following the improvements in wireless communications, high speed data transmission became a great necessity. There are mainly three sources of latency in data transmission at the physical layer: encoding latency, transmission latency, decoding latency. The former one is relatively very small and usually neglected. Since transmission latency depends drectly on the transmission bandwidth, the main bottleneck stands as the decoding latency which is the main focus of this article. Both turbo and turbo-like codes introduce huge decoding latencies since they operate in an iterative manner.

To overcome the latency problem, numerous techniques have been applied. One of the foremost techniques is parallel processing. Running multiple decoders concurrently enabled by the sliding window technique significantly reduces decoding latency ([3]–[5] and references therein). Another approach that enables concurrent operation of multiple decoders is parallellisation imposed to the encoder side which results in a neat parallel decoder structure as proposed in [6] and studied here. In our structure, decoding and encoding time is reduced by the number of simultaneously operating blocks.

Another idea is to reduce the latency of the component marginal a posteriori (MAP) decoders by applying an algorithm called “centre to top” (CTT) [3], [4], [7]. In CTT, the forward and backward recursions of decoding run concurrently and decoding latency is halved at the expense of computational resources. This technique will be utilised here as well.

A main bottleneck in all parallelised turbo decoders is the interleaving operation and the accompanying memory collision problem. When the component decoders attempt at accessing the memory elements in the same memory block, a memory collision problem occurs. This problem is circumvented in various ways. One of the methods is algebraic interleaver construction specific to parallellised decoders (see [5], [8] and references therein) where some contraints have to be inflicted on the interleaver. Another approach is to obtain a solution via special routing based on parallel computing theory [9] for ASIC systems with added hardware complexity. We make use of specially designed interleavers called row-column S-random interleavers [10] in this study due to their simplicity. By the parallelisation at the encoder side, the use of parallel interleavers is possible. When these interleavers are read/written in rows or columns, memory access problems are fully avoided in an easy way as explained in [10].

Another issue in the design of turbo decoders is the use of fixed point aritmetic preferred over floating point arithmetic in practice. The limited number of bits causes overflow and underflow problems while computing the variables utilised in decoding. In the forward and backward recursions, to be described in Section II, the variables keep on increasing or decreasing unless special precautions are taken. Amongst other techniques ([7] and references therein), the variables are compared to a threshold in [7] and then a subtraction/addition operation is performed based on the comparison. We take a simpler approach in this study where the variables are normalised in relation to the maximum at each step of the decoding algorithm. This does not add a significant latency or hardware complexity and performs very well. Another important issue is the handling of incoming channel observations and how they will be used in the decoding algorithm. In this study, the channel observations are directly mapped to probability quantised in a fixed number of bits at some granularity dependent on SNR. Although many turbo decoder implementation articles propose an ad hoc quantisation rule, we will derive a quantisation step based on analysis. All the probability variables of the decoding algorithm are kept with the same number of bits and quantisation step as the channel observation probability. This results in a uniformity within the decoder that simplifies the design and enables higher clock rates.

Our aim in this paper is to compare some parallellised turbo decoder architectures and show detailed implementation strategies to decrease decoding latency. The effects of design choices in FPGA implementation will be examined in regard to maximum clock speed, number of slices used, maximum data rate, and error rate performance. In particular, turbo codes and repeat-accumulate codes will be inspected in terms of the mentioned parameters. The strategies utilised throughout this study are not specific to an FPGA implementation and can be easily applied to an application-specific integrated circuit (ASIC) design.

The paper consists of the following sections. In Section II, we present the mathematical expressions for the BCJR algorithm. Section III describes the idea of parallellisation introduced for turbo codes. In Section IV, we explain the parallellised repeat-accumulate codes. In Section V, we provide the details of implementation and optimisations applied during the realisation of the proposed systems on an FPGA board. Some numerical results are presented in Section VI and conclusion follows in Section VII.

## II. BCJR ALGORITHM

The BCJR algorithm [11] is the most popular MAP decoding algorithm. It aims to minimize the bit error rate (BER) by maximizing the marginal a posteriori probabilities. In practice, the BCJR algorithm usually calculates the *a posteriori log-likelihood ratio* (*a posteriori L-value*) of an information bit where log-domain operation usually simplifies decoder operation. Since derivation of the algoritgm can be found in many standard textbooks such as [12], we will briefly overview some of the issues for sake of reference in the subsequent sections of the paper.

The log-likelihood ratio ($LL$) of an information bit $u_l$ can be calculated as

$$LL(u_l) = \ln \left[ \frac{p(u_l = +1|\mathbf{y})}{p(u_l = -1|\mathbf{y})} \right], \tag{1}$$

for a received signal sequence $\mathbf{y}$. Using this a posteriori L-value, a hard decision corresponding to $u_l$ can be found by

$$\widehat{u_l} = \begin{cases} +1, & LL(u_l) > 0 \\ -1, & LL(u_l) < 0 \end{cases}. \tag{2}$$

The *branch metric* at time $l$ is the probability of having a state transition from state $s'$ to $s$ at time $l$ and defined as

$$\gamma_l(s', s) = \ln p(s_{l+1} = s, \mathbf{y_l} | s_l = s'), \tag{3}$$

where $\mathbf{y_l}$ is the received signal vector at time $l$ consisting of both data and parity observations. In an AWGN channel, branch metrics can be written as

$$\gamma_l(s', s) = \frac{L_a(u_l)}{2} u_l + \frac{L_c}{2} (\mathbf{y_l} \cdot \mathbf{v_l}), \tag{4}$$

where $L_a(u_l)$ is the a priori bit probability[1], $L_c$ is the channel reliability factor which is equal to $4E_s/N_0$, and $\mathbf{v_l}$ denotes the output vector consisting of data and parity signals for transition from state $s'$ to $s$. The dot product $(\mathbf{r_l} \cdot \mathbf{v_l})$ gives the correlation between the hypothesised transmitted and received vectors. Scaling this distance with $L_c$ means that the observations are more reliable when SNR is high and a priori values are trusted more when SNR is low.

The BCJR algorithm determines the bit likelihoods based on forward and backward recursions

$$\alpha_{l+1}(s) = \ln \sum_{s' \in \sigma_l} \mathrm{e}^{[\gamma_l(s',s) + \alpha_l(s')]}, \tag{5}$$

$$\beta_l(s') = \ln \sum_{s \in \sigma_{l+1}} \mathrm{e}^{[\gamma_l(s',s) + \beta_{l+1}(s)]}, \tag{6}$$

with initial conditions in case of trellis termination

$$\alpha_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}, \ \beta_N(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}. \tag{7}$$

[1]It must be noted that the $L_a$ values for the termination bits are always $0$.

The log-likelihood ratio in (1) can be written using the variables described above as [12]

$$\begin{aligned} LL(u_l) &= \ln\left\{ \sum_{(s',s)\in\Sigma_l^+} \mathrm{e}^{[\alpha_l(s')+\gamma_l(s',s)+\beta_{l+1}(s)]} \right\} \\ &- \ln\left\{ \sum_{(s',s)\in\Sigma_l^-} \mathrm{e}^{[\alpha_l(s')+\gamma_l(s',s)+\beta_{l+1}(s)]} \right\} \end{aligned} \tag{8}$$

where $\Sigma_l^+$ and $\Sigma_l^-$ are the sets of transitions with the information bit is 0 and 1, respectively.

It can easily be seen that calculations can be simplified more by defining a $max^*$ operation

$$max^*(x,y) = \ln(\mathrm{e}^x + \mathrm{e}^y) = \max(x,y) + \ln(1+\mathrm{e}^{-|x-y|}), \tag{9}$$

where the second term is usually called the *correction term*.

## III. PARALLEL DECODABLE TURBO CODES

Turbo codes were first introduced by Berrou et. al. [1] in 1993. A turbo code's (TC) encoder consists of two convolutional encoders operating in parallel. While the upper encoder encodes the information bits in the given order, the lower one operates on an interleaved version of the information bits. A TC decoder consists of two MAP decoders operating iteratively. Computed $LL$ values out of one decoder are used in the other, and vice versa. By subtracting the input $LL$ values from the computed ones, the *extrinsic information* is obtained, which is the actual estimation of that decoding step [1].

However, using an iterative decoding scheme may suffer from huge decoding latencies and degrade the transmission rate in real time operations. To overcome this drawback in a simple way, parallellisation is introduced on both the encoder and decoder sides [6] as shown in Figures 1 and 2. In the parallel decodable turbo code (PDTC) decoder structure, there exist decoder clusters instead of single component decoders. These two clusters operate iteratively, while the decoders in each cluster operate simultaneously and independently. This architecture decreases the decoding latency by $N$, where $N$ is the number of parallel decoders in a cluster[2]. While TC decoders operate on a block of $K$ information bits, each decoder in PDTC operates on $K/N$ information bits in parallel.

To implement this parallel structure, an architecture suitable for parallel processing is needed. For that reason, FPGA is chosen here to investigate the PDTC performance. In our studies, we have used 4 parallel decoders in each cluster without loss of generality which is a good degree of parallellisation and does not overload the FPGA platform used herein.

[2]The number of decoders in each cluster may be different in general.

## IV. PARALLEL DECODABLE REPEAT ACCUMULATE CODES

Repeat Accumulate (RA) codes are in the class of turbo-like codes. After the invention of turbo codes, Divsalar et. al [2] applied the random-like interleaver structure to a set of coding schemes. Repeat accumulate codes are obtained by the concatenation of a repetition code encoder and a $2$-state convolutional encoder. The decoding latency issue surfaces also in RA codes and hence the parallelisation idea may be applied. The parallel concatenated repeaters repeat the uncoded data bits $L$ times and forward the results to the rate-$1$ convolutional encoders after passing through an interleaver. Hence, the overall rate of the encoder becomes $1/L$ which is the same as that of the repetition encoder.

Although factor-graph based decoding is usually preferred for RA codes, we will utilise convolutional decoders along with repetition decoders so that similar architectures will be compared. In the decoding process, decoders operating in parallel are utilised in order to decrease latency as in PDTC decoders. The parallellised decoder structure is shown in Figure 4.

## V. FPGA IMPLEMENTATION

We have implemented our system on a Xilinx ML402 Virtex-4 SX Evaluation Platform and produced test results by working on it. This platform has a medium sized FPGA with number of available slices $15000$. Its architecture enables building parallel processing blocks. However, implementing a soft-in soft-out (SISO) decoder on an FPGA inherently faces some problems since it has limited resources which do not let one easily use floating-point arithmetic or large fixed-point arithmetic. Throughout this section, we will describe our solutions to the problems and optimisations we have applied. BPSK modulation is assumed throughout and other constellations can be similarly applied.

### *A. The Centre to Top Algorithm*

When the metric calculations in (5) and (6) are considered, it can be seen that the two operations are independent of each other. This gives the ability to calculate $\alpha$ and $\beta$ metrics simultaneously assuming that all of the received values are available for branch metric calculations. This assumption is valid for the iterative decoding schemes (like of turbo codes as in our case) since decoding process can begin after receiving the whole packet. By this algorithm, the decoding time can be halved. Consider a decoder running on $42$ bits ($40$ information bits and $2$ termination bits) as will be used throughout this study for each branch of the parallelised encoder/decoder. At time $0$ the metric values are initialised as defined in (7). $\alpha$ and $\beta$ values are calculated without computing any LL value up to time $21$. At time $21$, both of $\alpha_{21}$, $\beta_{22}$ and $\alpha_{20}$, $\beta_{21}$ values are available together with the branch metrics for the time, $\gamma_{21}$ and $\gamma_{20}$. So,

$LL(u_{21})$ and $LL(u_{20})$ are computed and given out. That process, starting from the centre of the frame, continues to the end and simultaneously to the beginning of the frame. That is why this algorithm is named as “centre to top” [3]. It must be noted that $\alpha$ and $\beta$ metric values do not have to be written to memory after the midpoint, since $LL$ values are calculated simultaneously. So, not only the decoding time but also the memory usage is halved by this algorithm.

## *B. Memory Collision-Free Interleavers*

Block RAMs are the most widely used storage elements on FPGA projects. Starting from that fact, we have used block RAMs to store observations and calculated $LL$ values. However, using RAMs introduces some difficulties because of their limited accessibility. Even in the dual port block RAMs, there are only two available ports to read or write at the same time.

As described in Section III, each decoder in a cluster operate in parallel to each other. This means that each receive the observation values and input $LL$ values at the same time instants. Since block RAMs storing these values can serve to only one decoder for the desired address, 4 parallel decoders must be assigned to work on different RAMs. If two (or more) decoders try to work on (especially for “writing” operation) the same RAM, there will occur a memory collision. To prevent this problem, memory collision-free interleavers of [10] have been used.

## *C. Observation Quantisation*

In the conventional mathematical model, a $+1$ or $-1$ is assumed to be transmitted for BPSK, appropriate noise is added and calculations are carried on with these assignments. An AWGN channel for BPSK modulation can be modeled as

$$y_k = h_k x_k + n_k, \tag{10}$$

for any time instant $k$ where $y_k$ is the received symbol, $h_k$ is the channel gain ($\sqrt{E_s}$ in an AWGN channel with $E_s$ being the signal energy), $x_k$ is the transmitted bit ($x_k = \mp 1$) and $n_k$ is a circularly symmetric complex Gaussian random variable with mean 0 and variance $N_0$.

The conditional probability of a received symbol $y_k$ can be expressed as

$$\begin{aligned} f(y_k|h_k,x) &= \frac{1}{\pi N_0} e^{-\frac{|y_k-h_kx|^2}{N_0}}, & (11)\\ \ln(f(y_k|h_k,x)) &= -\ln(\pi N_0) - \frac{|y_k|^2}{N_0} - \frac{|h_k|^2|x_k|^2}{N_0} \\ &\quad + \frac{2}{N_0}\Re\{y_k h_k^* x^*\}, & (12)\\ &= C + \frac{2}{N_0}\Re\{y_k h_k^* x^*\}, & (13) \end{aligned}$$

where $C$ is a constant and has no effect on the MAP calculations. Hence, the function can be redefined as

$$\ln(f(y_k|h_k,x)) \doteq \frac{2}{N_0}\Re\{y_k h_k^* x^*\}, \tag{14}$$

where $\doteq$ denotes equality with a constant.

As we use fixed-point arithmetic, the metric values in the BCJR algorithm are represented by a fixed number of bits, $K$. However, the decoder is not guaranteed to work properly with this representation unless the channel observations (input of the decoder) are carefully quantised. For that reason, we need to quantize observations by a quantisation factor, $q$, such that the represented observations lay in a set S smaller than the set of numbers represented by $K$ bits. After that, the quantised observation probability for $x = 1$ is used in decoding with

$$Q_k = Q(\ln(f(y_k|h_k, x = 1))) = \left\lfloor \frac{2/N_0 \Re(y_k h_k^*)}{q} \right\rfloor. \tag{15}$$

If we apply the AWGN channel model given in (10) on (15) for a BPSK modulation, we get

$$\begin{aligned} Q_k &= \left\lfloor \frac{2\sqrt{E_s}/N_0 \Re\{y_k\}}{q} \right\rfloor, & (16)\\ &= \left\lfloor \frac{2\sqrt{E_s}/N_0 \Re\{(\sqrt{E_s} + n_k)\}}{q} \right\rfloor, & (17)\\ &= \left\lfloor \frac{2E_s}{N_0 q} + \frac{2\sqrt{E_s}}{N_0 q} n_I \right\rfloor, & (18) \end{aligned}$$

where $n_I$ is the real part of the complex Gaussian noise with mean 0 and variance $N_0/2$.

Recalling that a finite number of bits are used in representing numbers, the question is how to choose $q$. If $q$ is chosen to be very small, $Q_k$'s will be large and the formulas such as (8) will not function properly due to overflow. If $q$ is chosen to be very large, then the difference in noise values of the observations will not be properly passed to the decoder and then soft decoding will suffer. We resolve the problem above by the compromise that the packet is normalised with respect to its absolute maximum symbol value,

$ObsMax$. If we represent that value with a predefined value, $NormMax$ (absolute maximum value after the quantisation is performed) then we get a set $S = \{-NormMax, -NormMax+1, \ldots, NormMax-1, NormMax\}$ for decoder's input sequence. This information can be combined with a well known property of the Gaussian distribution that, in a normally distributed set with mean $\mu$ and variance $\sigma^2$, obtaining a number $p$ such that $|p| > \mu + 3\sigma$ has a probability of about $1/1000$. To be able to apply that property, we need to identify the mean and variance of the random variable $A = \frac{2E_s}{N_0 q} + \frac{2\sqrt{E_s}}{N_0 q} n_I$.

$$E\{A\} = \frac{2E_s}{N_0 q} \tag{19}$$

$$\begin{aligned} \sigma_A &= \frac{2\sqrt{E_s}}{N_0 q}\sigma_{n_I} = \frac{2\sqrt{E_s}}{N_0 q}\frac{\sqrt{N_0}}{\sqrt{2}} \\ &= \sqrt{\frac{2Es}{N_0 q}}\frac{1}{\sqrt{q}} \\ &= \frac{\sqrt{E\{A\}}}{\sqrt{q}} \end{aligned} \tag{20}$$

After the quantisation of the packet, it is known that symbols greater than $+NormMax$ or smaller than $-NormMax$ can occur in the packet with a small probability. If we neglect the small probability of $1/1000$, we can define $NormMax$ as

$$NormMax = E\{A\} + 3\sigma_A \tag{21}$$

$$= E\{A\} + 3\frac{\sqrt{E\{A\}}}{\sqrt{q}} \tag{22}$$

By replacing (19) in (22), we get

$$NormMax = \frac{2E_s}{N_0 q} + 3\sqrt{\frac{2E_s}{N_0 q}}\frac{1}{\sqrt{q}}. \tag{23}$$

By solving this equation, $q$ can be calculated as

$$q = \frac{\frac{2E_s}{N_0} + 3\sqrt{\frac{2E_s}{N_0}}}{NormMax}. \tag{24}$$

As it is obvious in (24), $q$ is a function of the $SNR$ ($E_s/N_0$) for a selected $NormMax$ value. Instead of calculating the $q$ value for each packet, a look-up table (LUT) can be used. In our design, we have used a relatively large LUT that stores the $q$ values in $8$ bits, $3$ for integer part and $5$ for the decimal part. That gives a precision of $1/2^5$ and yields a satisfactory performance.

### *D. $max^*$ Approximation*

The correction term in $max^*$ operation causes problems when it is needed to be expressed in fixed-point arithmetic. It is not possible to easily realize the $\ln$ function fully in such a system. For that reason, some approximations must be made to implement the $max^*$ operation. There are basically two approximations in the literature. These two different approaches result in *log-MAP* with tables and *max-log-MAP*.

If the decoder is a log-MAP decoder then $max^*$ calculation is a more difficult subject, because the correction term, $\ln(1 + \mathrm{e}^{(-|x-y|)})$, should be calculated. Since the hardware implementation of such a function is complicated, this term is handled by construction of a LUT in practice. As described in the previous part, the observations are in quantised form, therefore LUT values also have to be quantised accordingly. That is, if the inputs to the $max^*$ function are in a quantised fashion, the other terms generated in the function also should be quantised in parallel with the inputs. The LUT construction function is,

$$LUT(i) = \left\lfloor \frac{\ln(1 + \mathrm{e}^{-iq})}{q} \right\rfloor \tag{25}$$

where $i$ is the absolute value of the difference of the inputs of the $max^*$ function. The LUT sizes are usually quite small (around 5-6 entries) with reasonable $NormMax$ values.

In a max-log-MAP decoder, the correction term is neglected, that is $max^*$ operation is the same with ordinary $\max$ operation. So, the quantisation term, $q$, is useless for this method. In other words, it can be said that decoder does not need an exact SNR estimation to operate properly. Studies in [13] and [14] have shown that max-log-MAP decoders work without any need on SNR estimation.

## *E. Fixed-Point Summation and Subtraction*

Using a restricted set ($[-(2^{K-1} - 1), 2^{K-1} - 1]$ where $K$ is the metric size) to represent metric values forces us to introduce new summation and subtraction operations with the closure property in the given set. The operation $clipsum$, denoted by $\oplus$, replaces with the regular summation. Under the assumption of $plus\_inf = 2^{K-1} - 1$ and $minus\_inf = -plus\_inf$,

$$a \oplus b = \begin{cases} plus\_inf, & a \geq plus\_inf \ or \ b \geq plus\_inf \\ minus\_inf, & a \leq minus\_inf \ or \ b \leq minus\_inf \\ plus\_inf, & a + b \geq plus\_inf \\ minus\_inf, & a + b \leq minus\_inf \\ a + b, & else. \end{cases} \tag{26}$$

Similarly, a new subtraction operation $clipsubtract$ ($\ominus$) is introduced as

$$a \ominus b = a \oplus (-b). \tag{27}$$

## *F. Node ($\alpha$, $\beta$) Metric Normalisation*

In (5) and (6) it has been shown that $\alpha$ and $\beta$ values are updated in a recursive manner. As the computations go further, these metric values may overflow ($> plus\_inf$) or underflow ($< minus\_inf$). To solve this problem, $\alpha$ and $\beta$ values are normalised at each trellis step. After each forward recursion, maximum of the newly generated forward metric values is subtracted from these values and $\alpha$ metrics are updated with these normalised values. The same is applied to the $\beta$ metrics. After the normalisation process, we get a maximum value of $0$ for $\alpha$ and $\beta$ metrics at each time instant and prevent underflow and overflow cases. Another approach to node metric normalisation can be found in [15].

## *G. Memory Complexity*

Before the decoding process, the observations have to be stored in different memories in order to use them in the parallel decoder structure. For that reason, a memory structure is defined. That structure differs for PDTC and PDRAC decoders.

*1) PDTC memory structure:* If there are $N$ decoders operating in parallel, then there must be $N$ independent memory blocks for data bit observations ($d$ in Figure 2). Accordingly, $N$ memory blocks are used for parity observations and $N$ memory blocks for interleaved parity observations ($p_1$ and $p_2$ in Figure 2, respectively). In addition to these, $N$ memory blocks are also defined for interleaver (memory collision-free interleavers) tables.

*2) PDRAC memory structure:* For $N$ parallel MAP decoders, the observations are stored into $N$ memory blocks. Different than the PDTC case, there are no data observations. Similar to the PDTC case, $N$ memory blocks are used to store the interleaver tables.

Log-likelihood values are stored in RAMs, too. Each decoder needs an a priori probability ($L_a$) and generates log-likelihood ratio ($LL$) and *extrinsic information* ($L_e$), where in our design $L_e$'s are calculated within the MAP decoder[3]. These $L_e$ and $L_a$ notations are eligible for the decoders running in the first cluster. In the second cluster, decoders use $L_e$ values as $L_a$ and generates the $L_e$ values which will be used as $L_a$ in the next iteration. The word "cluster" is used just for imaging which defines half of an iteration. In fact, decoders only change their state to switch the input and output log-likelihood ratios ($L_a$ and $L_e$). Since $LL$ values are final results, they are updated (overwritten) after each *cluster* run.

[3]The extrinsic information is generated inside the decoder to decrease the system complexity at the expense of maximum clock speed.

That structure brings out a memory usage of $3N$ memory blocks for log-likelihood ratio storage in both PDTC and PDRAC decoders.

Summing up all yields a usage of $7N$ number of memory blocks for PDTC decoder and $5N$ for PDRAC decoder.

## VI. Results

In this section, the effects of the implementation choices will be shown and the performances of parallellised decoder structures will be compared. Our results are produced with the choices given in the remainder of this paragraph. The parallel decoder architectures are constructed by using 4 parallel MAP decoders. This choice was a good compromise between a good degree of paralellisation and not overloading the FPGA so that problems arising from fitting a tight design into a limited FPGA are avoided. Each MAP decoder is responsible for decoding 42 bits. 40 bits of these corresponds to information and 2 corresponds to termination bits. Hence, the initial and final values of $\alpha$ and $\beta$ values are respectively known due to trellis termination. Since each decoder decodes 40 information bits, a packet consists of $4 \times 4 = 160$ information bits in total. In detail, 160 bits of information is embedded in a 512-bit packet with 160 information bits, 160 parity bits, 160 interleaved parity bits where the remaining 32 bits are the termination bits for $4$ parallel decoders. The overall code is slightly smaller than $1/3$. Furthermore, the number of iterations is chosen as 4 for PDTC and 8 for PDRAC which are close to average iteration numbers with early-stop procedures ([16] and references therein) for SNR of interest. In the memory collision-free interleavers of [10], 4 different S-random interleavers of size 40 are needed here. Without any optimisation of the interleavers, 4 different interleavers with the S value of $5$ are generated and used in all simulations.

The choice for the number $NormMax$ (explained in Section V-C) affects the performance of decoder such that choosing a high value results in saturation in metric calculations while choosing a small value causes a loss in representing observation values. This effect on performance has been shown in Figures 7, 8, and 9. The figures are obtained by observing $6000$ packets with each containing $160$ information bits for $E_b/N_0 = 2.6$ dB. The optimum $NormMax$ values for each $K$ and decoding algorithm can be obtained from these figures. Also, in the figures it can be seen that as the $K$ value increases, so does the number of available optimum $NormMax$ values.

Another parameter that affects the performance of the decoders is the selection of the $K$ value. Choosing a large $K$ results in a better performance while at the same time causes the decoder to consume more resources on FPGA and to work on a lower clock speed. So, there is a tradeoff between the performance, resource consumption, and the speed. The performance results are given in Figures 10, 11, and 12. It must be noted that these comparisons are made by using the best resulting $NormMax$ values for each decoder structure and $K$ value after $2000$ runs on a $160$ bit information packet (equally $512$-bit packet) with a fixed iteration number ($4$ for PDTCs and $8$ for PDRAC).

Figures 10-12 suggest that larger values of K enable better performance. That is why the selection of the $K$ value in a design must be done carefully to match the BER requirement. In these figures, it can be seen that the performance does not improve significantly beyond $K = 6$. So, the choice of $K$ as $6$ seems to be a good choice for these decoder structures. That brings a performance within $0.5$ dB from the floating-point for the PDTC decoders and $0.3$ dB for the PDRAC decoder.

In this study, we have used two different architectures for implementing PDTC decoders, namely *Architecture-A* and *Architecture-B*. The difference between these two architectures can be explained as follows. *Architecture-A* uses combinatorial logic operations heavily in the BCJR decoder implementation. All the numerical calculations are carried on in one clock cycle, i.e., $\alpha/\beta$ calculation, $\alpha/\beta$ normalisation, $\alpha$, $\beta$, $\gamma$ summation and $LL$ computations are done in one clock cycle. Decoding a $42$-bit length sequence ($42$ data bits, $42$ parity bits) takes $42$ cycles on *Architecture-A*. On the other hand, *Architecture-B* uses a pipelining structure in some calculations where independent computations are pipelined. The calculation and normalisation of $\alpha/\beta$ metrics are done in one cycle. The summations in (8) are done in a separate cycle (concurrent with the calculation and normalisation of alpha/beta metrics for the subsequent trellis stage) and $LL$ values are calculated in yet another clock cycle. The extrinsic values are determined in another cycle as well. Lastly the computed extrinsic information is forwarded to the output of the decoder in one cycle. This architecture aims to divide the combinatorial logic operations into smaller pipelined blocks to reduce the gate delays. Since independent operations are pipelined, all the operations continue still simultaneously except for an extra latency of $4$ clock cycles. By this architecture decoding a $42$-bit length sequence takes $46$ cycles. The resource consumptions for these two architectures are given in Tables I and II together with the speed considerations for different $K$ values. In Table III, the synthesis results for PDRAC decoder is given[4]. It can be observed that pipelining can enable significantly larger clock speeds at the expense of a small increase in percentage of slices used.

The distinct difference between log-MAP and max-log-MAP algorithms is that the former uses a LUT. As described in Section V-C, LUT is generated by using the $q$ value. In our design RA code uses the log-MAP algorithm. That is why, the $q$ calculating circuit is included in these two codes and it consumes

[4]Due to the speed advantage of *Architecture-B*, it is used on PDRAC decoder implementation.

TABLE I

SYNTHESIS RESULTS OF PDTC DECODER WITH MAX-LOG-MAP ALGORITHM

| metric size (bits) | Slices used | Slice usage (%) | Max. clock speed (MHz) |
|---|---|---|---|
| Architecture-A | | | |
| 4 | 6070 | 39 | 50.3 |
| 5 | 6104 | 39 | 49.8 |
| 6 | 6570 | 42 | 47.6 |
| 7 | 7174 | 46 | 43.8 |
| Architecture-B | | | |
| 4 | 6347 | 41 | 87.2 |
| 5 | 6501 | 42 | 86.9 |
| 6 | 6994 | 45 | 86.9 |
| 7 | 7537 | 49 | 85.7 |

some additional resources. These extra resources are included in the results given in Tables II and III. It is obvious that LUT insertion degrades the design performance in terms of both resource usage and maximum clock speed. The reason of that can be explained as follows. LUT can be thought as a large multiplexer which is controlled by the $q$ value and the inputs of the $max^*$ operation. Additionally the results of the LUT have to be added in the $max^*$ operation in order that $max^*$ result can be ready at the next clock cycle, that is, a combinatorially operating large multiplexer degrades the resource usage and combinatorial addition degrades the maximum operating frequency. Also, it must be noted that the slice usage increases almost linearly with $N$.

It must be noted that maximum achievable clock speeds given in Tables I,II and III are the predicted clock speeds of the Xilinx sysnthesis tool (XST). It is our observation that a carefully constrained design can often achieve even higher clock speeds when programmed onto an FPGA.

Large decoding latencies in turbo and turbo-like codes are often the drawback of these algorithms. By making them operate in parallel, a decrease in their decoding latencies is expected. To observe that decrease, decoding latencies are better to give in a formula. The decoding latency, $\tau$, for our parallel

TABLE II

SYNTHESIS RESULTS OF PDTC DECODER WITH LOG-MAP ALGORITHM

| metric size (bits) | Slices used | Slice usage (%) | Max. clock speed (MHz) |
|---|---|---|---|
| Architecture-A | | | |
| 5 | 8179 | 53 | 36.2 |
| 6 | 10628 | 69 | 31.5 |
| 7 | 11309 | 73 | 31.3 |
| Architecture-B | | | |
| 5 | 8663 | 56 | 65.5 |
| 6 | 10595 | 68 | 60.0 |
| 7 | 10807 | 70 | 55.4 |

TABLE III

SYNTHESIS RESULTS OF PDRAC DECODER

| metric size (bits) | Slices used | Slice usage (%) | Max. clock speed (MHz) |
|---|---|---|---|
| 5 | 5019 | 32 | 68.7 |
| 6 | 6046 | 39 | 68.1 |
| 7 | 6219 | 40 | 64.2 |

decodable turbo code decoder stucture (both log-MAP and max-log-MAP) is

$$\tau = \frac{D}{N}2I, \text{ for } \textit{Architecture-A} \text{ and} \tag{28}$$

$$\tau = \left(\frac{D}{N}+6\right)2I, \text{ for } \textit{Architecture-B}, \tag{29}$$

where $D$ is the number of information (data) bits in the packet, $N$ is the number of parallel decoders in a cluster and $I$ is the iteration number. The $\frac{D}{N}$ term is the decoding latency of a BCJR decoder operating with the CTT algorithm. The addition by $6$ in (29) is the result of the latency in BCJR ($4$) and interleaver structure ($2$) due to pipelining in *Architecture-B*. The reason of multiplying by $2I$ is that in each iteration the BCJR decoders run twice, one for the uninterleaved form of data and one for the interleaved.

Similarly for PDRAC decoders the decoding latency can be expressed as

$$\tau = \left(\frac{D}{NR} + \frac{D}{2NR} + 6\right) I \tag{30}$$

$$= \left(\frac{3D}{2NR} + 6\right) I \tag{31}$$

where $R$ is the code rate. The term $\frac{D}{NR}$ in (30) is the latency introduced by the BCJR decoders and the term $\frac{D}{2NR}$ is the latency introduced by the accumulate decoders. Although not performed here, different number of parallellised blocks for BCJR and accumulate decoders can be utilised for optimisation purposes.

During the decoding latency calculations, we have assumed that a *ping-pong* buffer structure is used in the receiver side. While a packet is being received, the observations are stored in a memory in the quantised form. After that, *ping* memories are filled as described in Section V-G for $d$, $p_1$ and $p_2$ observations and decoders begin to run. If another packet arrives during the decoding process, the $d$, $p_1$ and $p_2$ observations are stored in *pong* memories. In this case, the decoding process is not affected by the reception of the new packet. When the decoders finish their job, they operate on the *pong* memories and this time the *ping* memories are free for another packet storage. This structure doubles the memory usage in the system for storing observations.

At this point, we can make a final comparison between all the proposed structures in terms of maximum available data rates. If we denote the data rate by $\upsilon$, we can formulate it as,

$$\upsilon = \frac{D \text{ x } f}{\tau}, \tag{32}$$

where $f$ is the maximum available frequency and $\tau$ is the decoding latecy. To find the exact data rate, we need to decide on the architecture, number of data bits in a packet ($D$),metric representation width ($K$), iteration number ($I$), the number of constituent decoders in a cluster ($N$), and the code rate ($R$). In data rate calculation, the $f$ value can be obtained by checking the Tables I, II, and III for the selected $K$ value. Similarly, $\tau$ value can be obtained from (28), (29), or (31) for the decided structure and architecture. After observing the BER performances and FPGA resource usage, we have decided to use $K = 6$ for log-MAP and max-log-MAP PDTC decoders with both architectures and $K = 7$ for PDRAC decoder. Herein we used packets containing 160 data bits, that is $D = 160$. Using the Tables I, II, III and Figures 10, 11, 12 with the design choices listed above, we can generate Table IV that gives all the information and comparisons needed. In this table, the clock frequencies are adjusted in such a way that oscillators can be found in the market easily.

As seen in Table IV, $N$ values differ for each decoder structure. The idea of using different $N$ values can be explained as follows. It is remarked before that the resource consumption increases approximately

TABLE IV

COMPARISON OF THE PROPOSED DECODER STRUCTURES

| Decoder & Architecture | $N$ | $I$ | $K$ (bits) | Clock speed (MHz) | SNR for $BER = 10^{-3}$ | Bit Rate (Mbps) |
|---|---|---|---|---|---|---|
| PDTC with max-log-MAP in *Architecture-A* | 8 | 4 | 6 | 48 | ∼2.2 dB | 48 |
| PDTC with max-log-MAP in *Architecture-B* | 8 | 4 | 6 | 80 | ∼2.2 dB | 61.54 |
| PDTC with log-MAP in *Architecture-A* | 6 | 4 | 6 | 30 | ∼2.0 dB | 22.5 |
| PDTC with log-MAP in *Architecture-B* | 6 | 4 | 6 | 60 | ∼2.0 dB | 36.73 |
| PDRAC with log-MAP | 10 | 8 | 7 | 60 | ∼2.7 dB | 15.4 |

linearly as $N$ increases. Working on that assumption, maximum available $N$ value is calculated for the usage of the all FPGA resources. In [17], it is stated that increasing the $N$ value does not affect the performance of the PDTC decoder. $I$ stands for the number of iterations carried on in parallel decoder structure and $K$ shows the number of bits that the observations are represented by. By these provided results, it is not hard to say that a PDTC decoder with log-MAP architecture in *Architecture-B* is the best in terms of error performance. However, max-log-MAP decoder may be preferred in terms of resource consumption and data rate but with a $0.2$ dB performance degradation. On the other hand, PDRAC decoder structure with parallellised MAP decoders does not present a satisfactory performance among the proposed architectures.

## VII. Conclusion

Decoding latency is an important issue in iterative decoding algorithms which may be overcome with parallellisation. In this paper we implemented a parallellised encoder/decoder structure for turbo and repeat-accumulate codes. A new and simple metric quantisation scheme was proposed and used to avoid problems of decoding in fixed point arithmetic. Combinatorial logic and pipelining based architectures were separately implemented for comparison purposes. Comparisons were made in terms of error performance, FPGA resource usage, FPGA clock speed, and data rate. Max-log-MAP with pipelining provided the best result when all issues are taken into account. PDRAC does not perform well in comparison to any of the turbo decoders at the same data rate.

## REFERENCES

[1] Berrou,C., Glavieux,A., and Thitimajshima,P. :'Near shannon limit error-correcting coding and decoding: Turbo-codes', in Proc. ICC'93, May 1993, pp. 1064–1070.

[2] Divsalar,D., Jin,H., and McEliece,R.J. :'Coding theorems for 'turbo-like' codes', Proc. 36th Allerton Conf. on Communication, Control, and Computing, September 1998, pp. 201–210.

[3] Jung,J., Lee,I., Choi,D., Jeong,J., Kim,K., Choi,E., and Oh,D. :'Design and architecture of low-latency high-speed turbo decoders', ETRI Journal, October 2005, 27, (5), pp. 525–532.

[4] Salmela,P., Sorokin,H., and Takala,J. :'A programmable max-log-MAP turbo decoder implementation', Hindawi VLSI Design, Article ID 319095, 2008, 17 pages.

[5] Dobkin,R., Peleg,M., and Ginosar,R. :'Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders', IEEE Trans. on VLSI Systems, April 2005, pp. 427–438.

[6] Gazi,O., and Yılmaz,A.Ö. :'Fast decodable turbo codes', IEEE Communication Letters, April 2007, 11, (2).

[7] Ituero,P., Lopez-Vallejo,M., and Mujtaba,S. :'A configurable application specific processor for turbo decoding', in Conf. Record 39th Asilomar Conf. Signals, Systems and Computers, Nov. 2005, 4, pp. 1356–1360.

[8] Tarable,A., Dinoi,L., and Benedetto,S. :'Design of prunable interleavers for parallel turbo decoder architectures', IEEE Communications Letters, Feb. 2007, 11,(2), pp. 167–169.

[9] Prescher,G., Gemmeke,T., and Noll,T.G. :'A parametrizable low-power high-throughput turbo-decoder', in Proc. ICASSP 2005, March 2005, pp. V–25–28.

[10] Gazi,O., and Yılmaz,A.Ö. :'Collision free row column s-random interleaver', IEEE Communication Letters, April 2009, 13, (4).

[11] Bahl,L.R., Cocke,J., Jelinek,F., and Raviv,J. :'Optimal decoding of linear codes for minimizing symbol error rate', IEEE Transactions on Information Theory, 1974, IT-20, pp. 284–287.

[12] Lin,S., and Costello,D.J. :'Error control coding: Fundamentals and applications', (Prentice-Hall, 1983).

[13] Summers,T.A., and Wilson,S.G. :'Snr mismatch and online estimation in turbo decoding', IEEE Transactions on Communications, 46, April 1998, (4), pp. 421–423.

[14] Worm,A., and Hoeher,P. :'Turbo-decoding without snr estimation', IEEE Communications Letter, June 2000, 4, (6), pp. 193–195.

[15] Wu,P.H.-Y., and Pisuk,S.M. :'Implementation of a low complexity, low power, integer-based turbo decoder', IEEE GLOBECOM, 2001, pp. 946–951.

[16] Zhai,F., and Fair,I.J. :'Techniques for early stopping and error detection in turbo decoding', IEEE Trans. on Comm., Oct. 2003, 51, (10), pp. 1617–1623.

[17] O. Gazi :'Parallelized Architectures for Low Latency Turbo Structures', Ph.D. thesis, Middle East Technical University, 2007.

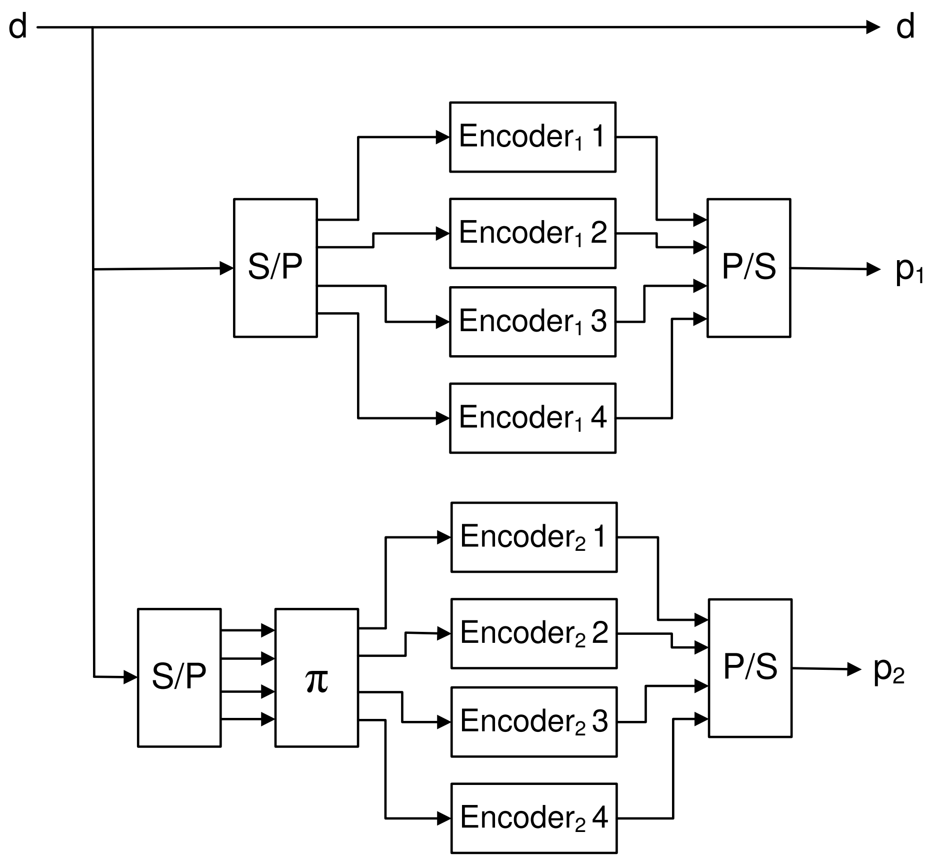

Fig. 1. PDTC encoder structure

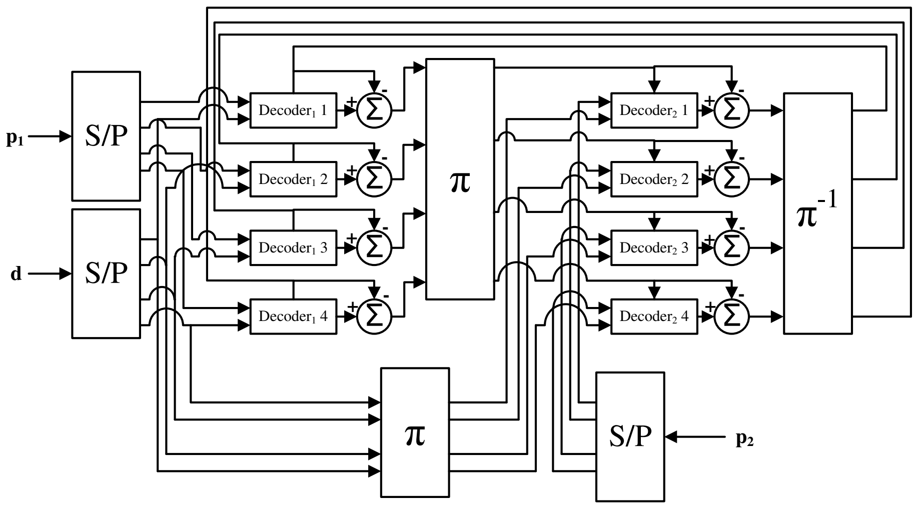

Fig. 2. PDTC decoder structure

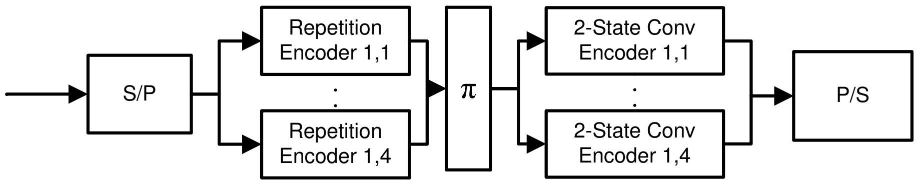

Fig. 3. Parallel Decodable Repeat Accumulate Code encoder structure

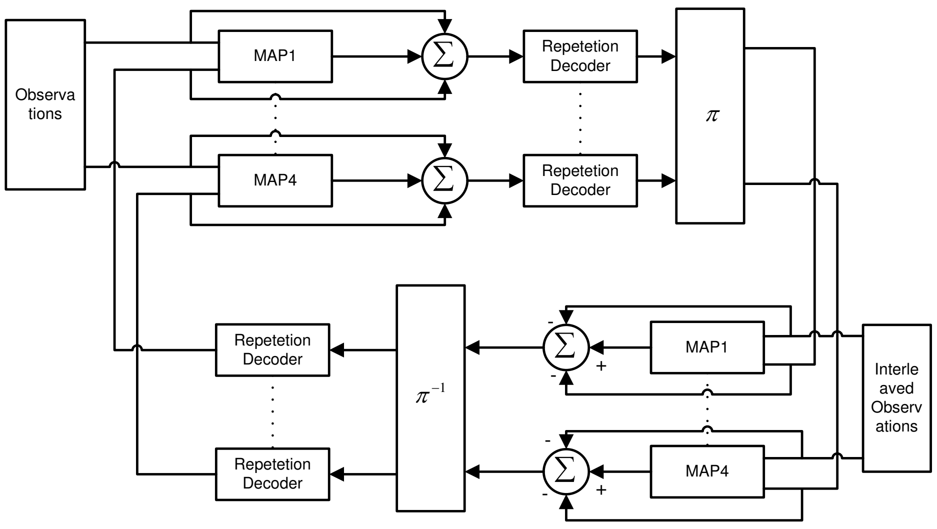

Fig. 4. Parallel Decodable Repeat Accumulate Code decoder structure

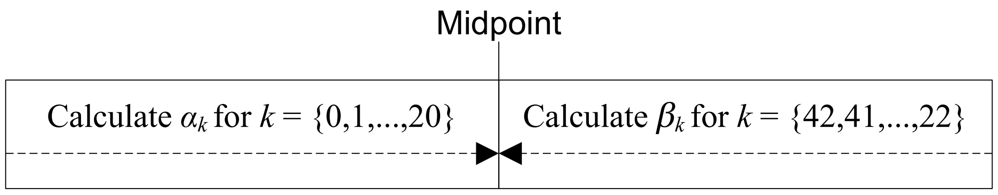
Midpoint
Calculate $\alpha_k$ for $k$ = {0,1,...,20}
Calculate $\beta_k$ for $k$ = {42,41,...,22}

Fig. 5. $\alpha$ and $\beta$ values are updated with no $LL$ computation upto the midpoint

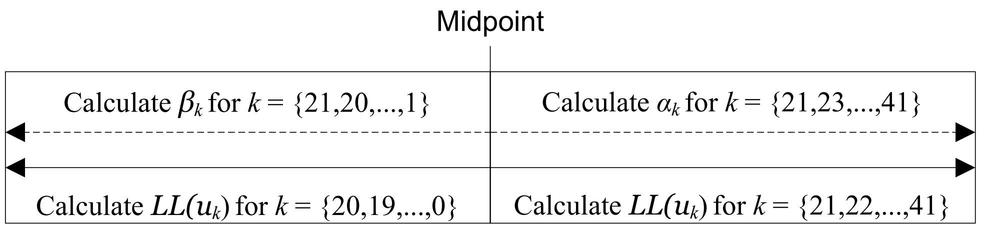
Midpoint
Calculate $\beta_k$ for $k$ = {21,20,...,1}
Calculate $\alpha_k$ for $k$ = {21,23,...,41}
Calculate $LL(u_k)$ for $k$ = {20,19,...,0}
Calculate $LL(u_k)$ for $k$ = {21,22,...,41}

Fig. 6. While $\alpha$ and $\beta$ values are updated, $LL$ values are being computed and given out at the same time

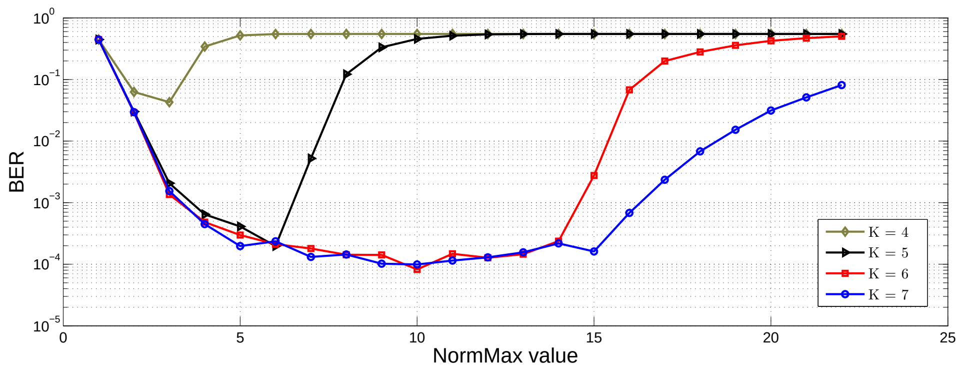

Fig. 7. The effect of $NormMax$ value on a rate-$1/3$ PDTC decoder using the max-log-MAP algorithm.

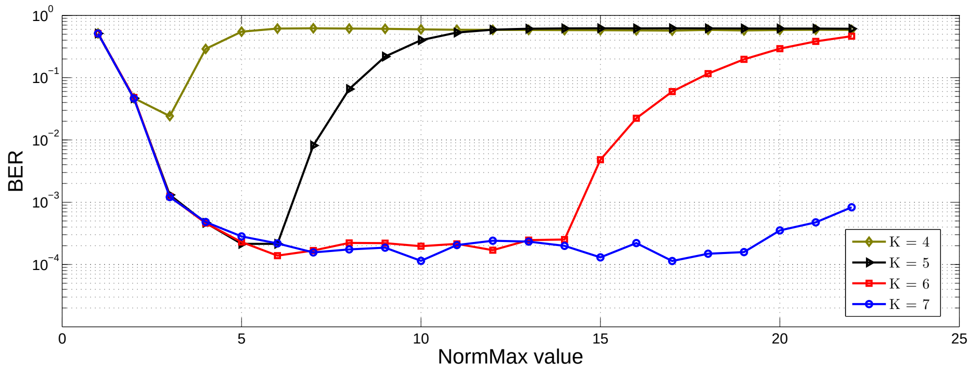

Fig. 8. The effect of $NormMax$ value on a rate-$1/3$ PDTC decoder using the log-MAP algorithm.

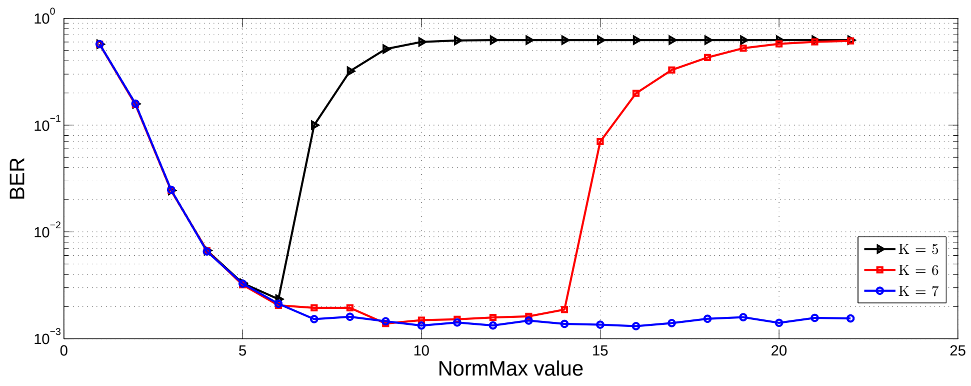

Fig. 9. The effect of $NormMax$ value on a rate-$1/3$ PDRAC decoder.

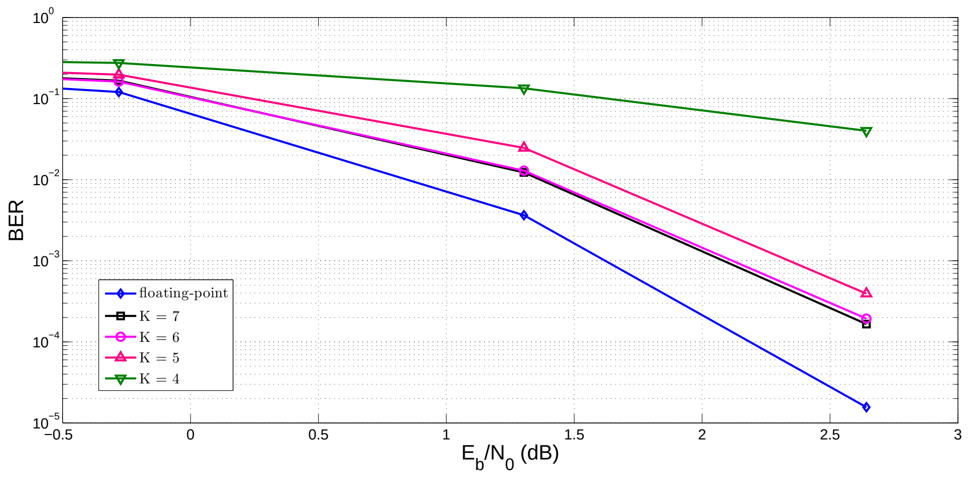

Fig. 10. Performance of PDTC decoder with max-log-MAP algorithm.

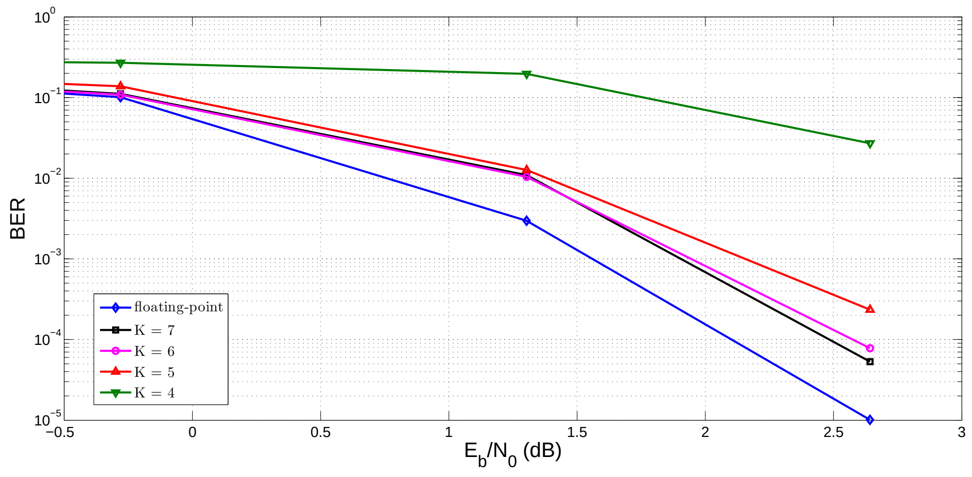

Fig. 11. Performance of PDTC decoder with log-MAP algorithm.

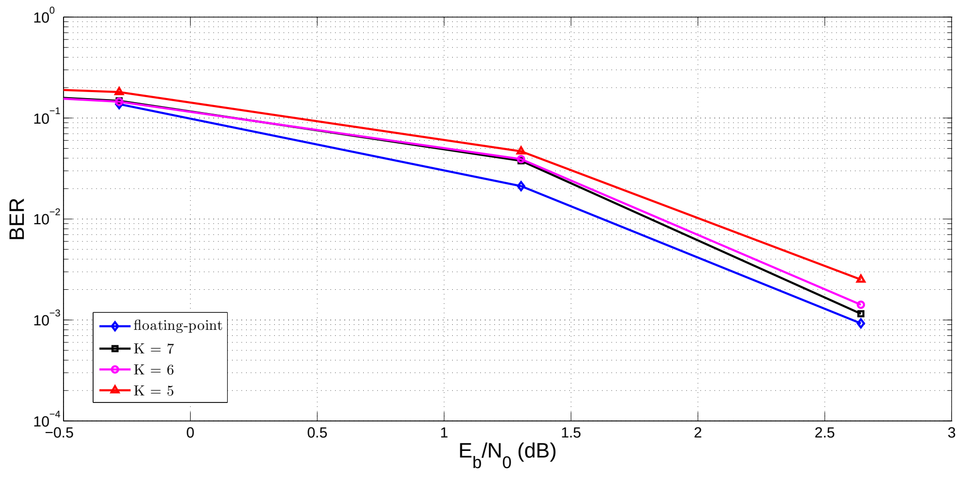

Fig. 12. Performance of PDRAC decoder.