CTC Turbo Decoding Architecture for LTE Systems Implemented on FPGA

Cristian Anghel, Valentin Stanciu, Cristian Stanciu, and Constantin Paleologu
Telecommunications Department
University Politehnica of Bucharest
Romania
canghel@comm.pub.ro, svl117@yahoo.com, cristian@comm.pub.ro, pale@comm.pub.ro

***Abstract*— This paper describes a turbo decoder for Long Term Evolution (LTE) standard, release 8, using a Max Log MAP algorithm. The Forward Error Correction (FEC) block dimensions, as indicated in the standard, are inside a range of 40 to 6144 bits. The coding rate is 1/3, the puncturing block not being taken into discussion here. The number of turbo iterations is variable, but in this study it was usually set to 3. The turbo decoder is implemented on a Xilinx Virtex-5 XC5VFX70T Field Programmable Gate Array (FPGA).**

Keywords- turbo codes; Max Log MAP decoder; FPGA implementation; LTE standard.

I. INTRODUCTION

The discussions around the channel coding theory were intense in the last decades, but even more interest around this topic was added once the turbo codes were found by Berrou, Glavieux, and Thitimajshima [1][2][3].

At the beginning of their life, after proving the obtained decoding performances, the turbo codes were introduced in different standards as recommendations, while convolutional codes were still mandatory. The reason behind this decision was especially the high complexity of turbo decoder implementation. But the turbo codes became more attractive once the supports for digital processing, like Digital Signal Processor (DSP) or Field Programmable Gate Array (FPGA), were extended more and more in terms of processing capacity. Today the chips include dedicated hardware accelerators for different types of turbo decoders, but this approach makes them standard dependent.

The Third-Generation Partnership Project (3GPP) [4] is an organization, which adopted early these advanced coding techniques. Turbo codes were standardized from the first version of Universal Mobile Telecommunications System (UMTS) technology, in 1999. The next UMTS releases (after High Speed Packet Access was introduced) added support for new and interesting features, while turbo coding remained still unchanged. Some modifications were introduced by the Long Term Evolution (LTE) standard [5][6], not significant as volume, but important as concept. While keeping exactly the same coding structure as in UMTS, 3GPP proposed for LTE a new interleaver scheme.

Valenti and Sun presented in [7] a UMTS dedicated turbo decoding scheme. Due to the new LTE interleaver, the decoding performances are improved compared with the ones corresponding to UMTS standard. Moreover, the new LTE interleaver provides support for the parallelization of the decoding process inside the algorithm, taking advantage on the main principle introduced by turbo decoding, i.e., the usage of extrinsic values from one turbo iteration to another.

This paper presents an efficient solution for the hardware implementation of a Convolutional Turbo Code (CTC) LTE decoder. The optimization indicators refer to the used logic area and to the obtained decoding speed. Also the level of performances degradation introduced by the finite precision representation is taken into account when selecting the final implementation solution.

The paper is organized as follows. Section II describes the LTE coding scheme with the new introduced interleaver. Section III presents the decoding algorithm. In Section IV, the implementation solutions and the proposed decoding scheme are discussed. Section V presents area and speed results obtained when targeting a XC5VFX70T [8] chip on Xilinx ML507 [9] board; it also provides simulation curves comparing the results obtained when varying the most important decoding parameters. Section VI presents the final conclusions and the future perspective of this study.

II. LTE CODING SCHEME

The coding scheme presented in 3GPP LTE specification is a classic turbo coding scheme, including two constituent encoders and one interleaver module. It is described in Fig. 1. One can observe at the input of the LTE turbo encoder the data block C_k. The K bits corresponding to this block are sent as systematic bits at the output in the steam X_k. In the same time, the data block is processed by the first constituent encoder resulting parity bits Z_k, while the interleaved data block C'_k is processed by the second constituent encoder resulting parity bits Z'_k. Combining the systematic bits and the two streams of parity bits, the following sequence is obtained at the output of the encoder: $X_1, Z_1, Z'_1, X_2, Z_2, Z'_2, \ldots, X_k, Z_k, Z'_k$.

At the end of the coding process, in order to drive back the constituent encoders to the initial state, the switches from Fig. 1 are moved from position A to B. Since the final states of the two constituent encoders are different, depending on the input data block, this switching procedure will generate tail bits for each encoder. These tail bits have to be transmitted together with the systematic and parity bits resulting the following final sequence: $X_{k+1}, Z_{k+1}, X_{k+2}, Z_{k+2}, X_{k+3}, Z_{k+3}, X'_{k+1}, Z'_{k+1}, X'_{k+2}, Z'_{k+2}, X'_{k+3}, Z'_{k+3}$.

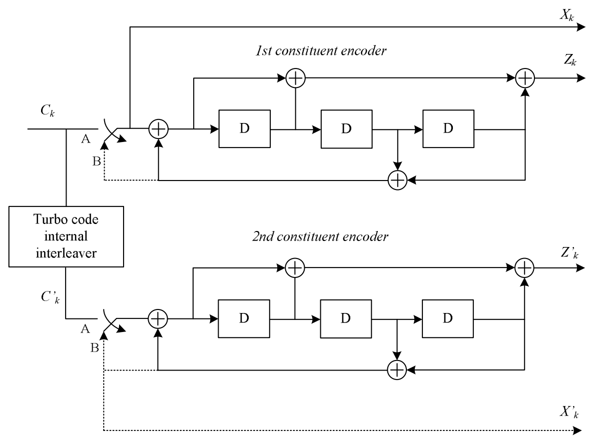

Figure 1. LTE CTC encoder.

As mentioned before, the novelty introduced by the LTE standard in terms of turbo coding is the interleaver module. The output bits are reorganized using

$$C_i^{'} = C_{\pi(i)}, \ i = 1, 2, ..., K, \tag{1}$$

where the interliving function π applied over the output index i is defined as

$$\pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \bmod K. \tag{2}$$

The length K of the input data block and the parameters f_1 and f_2 are provided in Table 5.1.3-3 in [5].

III. Decoding Algorithm

The LTE turbo decoding scheme is depicted in Fig. 2. The two Recursive Systematic Convolutional (RSC) decoders are using in theory the Maximum A Posteriori (MAP) algorithm. This classic algorithm provides the best decoding performances, but it suffers from very high implementation complexity and it can lead to large dynamic range for its variables. For these reasons the MAP algorithm is used as a reference for targeted decoding performances, while for real implementation new sub-optimal algorithms have been studied: Logarithmic MAP (Log MAP) [10], Maximum Log MAP (Max Log MAP), Constant Log MAP (Const Log MAP) [11], and Linear Log MAP (Lin Log MAP) [12].

For the proposed decoding scheme, the Max Log MAP algorithm is selected. This algorithm reduces the implementation complexity and controls the dynamic range problem with the cost of acceptable performances degradation, compared to classic MAP algorithm. The Max Log MAP algorithm keeps from Jacobi logarithm only the first term, i.e.,

$$\begin{aligned}&\max{}^*(x, y) = \ln(\mathrm{e}^x + \mathrm{e}^y) = \\ &\max(x, y) + \ln(1 + \mathrm{e}^{-|y-x|}) \approx \max(x, y).\end{aligned} \tag{3}$$

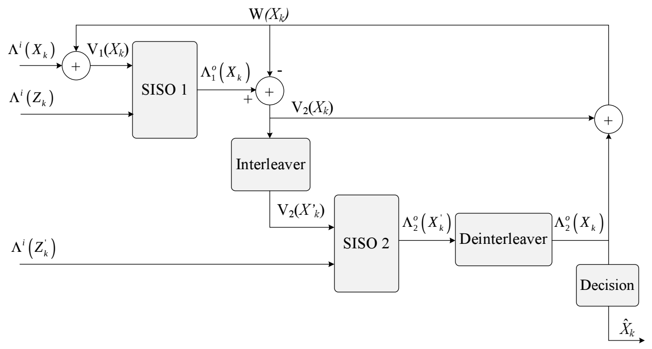

Figure 2. LTE turbo decoder.

The LTE turbo decoder trellis diagram contains 8 states. Each diagram state permits 2 inputs and 2 outputs. The branch metric between the states S_i and S_j is

$$\gamma_{ij} = \mathrm{V}(X_k)X(i,j) + \Lambda^i(Z_k)Z(i,j), \tag{4}$$

where $X(i,j)$ represents the data bit and $Z(i,j)$ is the parity bit, both associated to one branch. Also $\Lambda^i(Z_k)$ is the Log Likelihood Ratio (LLR) for the input parity bit. When Soft Input Soft Output (SISO) 1 decoder is taken into discussion this input LLR is $\Lambda^i(Z_k)$, while for SISO 2 it becomes $\Lambda^i(Z_k^{'})$; $V(X_k)$=$V_1(X_k)$ represents the sum between $\Lambda^i(X_k)$ and $W(X_k)$ for SISO 1 and $V(X_k)$=$V_2(X'_k)$ represents the interleaved version of the difference between $\Lambda_1^o(X_k)$ and $W(X_k)$ for SISO 2. In Fig. 2, $W(X_k)$ is the *extrinsic information* and $\Lambda_1^o(X_k)$ and $\Lambda_2^o(X_k^{'})$ are the output LLRs generated by the two SISOs.

In the LTE turbo encoder case, there are 4 possible values for the branch metrics between 2 states in the trellis:

$$\begin{aligned}\gamma_0 &= 0 \\ \gamma_1 &= \mathrm{V}(X_k) \\ \gamma_2 &= \Lambda^i(Z_k) \\ \gamma_2 &= \mathrm{V}(X_k) + \Lambda^i(Z_k).\end{aligned} \tag{5}$$

The decoding process is based on going forward and backward through the trellis.

A. Backward recursion

The trellis is covered backward and the computed metrics are stored in a normalized form at each node of the trellis. These stored values are used for the LLR computation at the trellis forward recursion. The backward metric for the S_i state at the k^{th} stage is $\beta_k(S_i)$, where $2 \le k \le K+3$ and $0 \le i \le 7$. The backward recursion is initialized with $\beta_{K+3}(S_0) = 0$ and $\beta_{K+3}(S_i) = 0, \forall i > 0$.

Starting from the stage k=K+2 and continuing through the trellis until stage k=2, the computed backward metrics are

$$\hat{\beta}_k(S_i) = \max\left\{(\beta_{k+1}(S_{j1}) + \gamma_{ij1}), (\beta_{k+1}(S_{j2}) + \gamma_{ij2})\right\}, \quad (6)$$

where $\hat{\beta}_k(S_i)$ represents the un-normalized metric and S_{j1} and S_{j2} are the two states from stage k+1 connected to the state S_i from stage k. After the computation of $\hat{\beta}_k(S_0)$ value, the rest of the backward metrics are normalized as

$$\beta_k(S_i) = \hat{\beta}_k(S_i) - \hat{\beta}_k(S_0) \quad (7)$$

and then stored in the dedicated memory.

B. *Forward recursion*

During the forward recursion, the trellis is covered in the normal direction, this process being similar with the one specific for Viterbi algorithm. Now only the forward metrics from the last stage (k-1) have to be stored, in order to allow the computation of the current stage (k) metrics. The forward metric for the state S_i at the stage k is $\alpha_k(S_i)$ with $0 \le k \le K-1$ and $0 \le i \le 7$. The forward recursion is initialized with $\alpha_0(S_0) = 0$ and $\alpha_0(S_i) = 0, \forall i > 0$. Starting from the stage k=1 and continuing through the trellis until the last stage k=K, the un-normalized forward metrics are given by

$$\hat{\alpha}_k(S_j) = \max\left\{(\alpha_{k-1}(S_{i1}) + \gamma_{i1j}), (\alpha_{k-1}(S_{i2}) + \gamma_{i2j})\right\}, \quad (8)$$

where S_{i1} and S_{i2} are the two states from stage k-1 connected to the state S_j from stage k. After the computation of $\hat{\alpha}_k(S_0)$ value, the rest of the forward metrics are normalized as

$$\alpha_k(S_i) = \hat{\alpha}_k(S_i) - \hat{\alpha}_k(S_0). \quad (9)$$

Because the forward metrics α are computed for the stage k, the decoding algorithm can obtain in the same time a LLR estimated for the data bits X_k. This LLR is found the first time by considering that the likelihood of the connection between the state S_i at k-1 stage and the state S_j at k stage is

$$\lambda_k(i,j) = \alpha_{k-1}(S_i) + \gamma_{ij} + \beta_k(S_j). \quad (10)$$

The likelihood of having a bit equal to 1 (or 0) is when the Jacobi logarithm of all the branch likelihoods corresponds to 1 (or 0) and thus:

$$\Lambda^o(X_k) = \max_{(S_i \to S_j):X_i=1}\{\lambda_k(i,j)\} - \max_{(S_i \to S_j):X_i=0}\{\lambda_k(i,j)\}, \quad (11)$$

where “max” operator is recursively computed over the branches, which have at the input a bit of 1 $\left\{(S_i \to S_j) : X_i = 1\right\}$ or a bit of 0 $\left\{(S_i \to S_j) : X_i = 0\right\}$.

IV. PROPOSED DECODING SCHEME

A. *Block Scheme*

Since one constituent decoder extrinsic outputs are inputs for the other, and because the interleaving or deinterleaving procedure is applied over data blocks, the operating periods for the two constituent decoders are not overlapped. Thus, the decoding scheme can use a single constituent decoder, which operates time-multiplexed. The proposed scheme is depicted in Fig. 3 and it is based on the previous work presented in [13] for a WiMAX CTC decoder. The memory blocks are used for storing data from one semi-iteration to another and from one iteration to another. SISO 1 reads the memory locations corresponding to V$_1$(X_k) and $\Lambda^i(Z_k)$ vectors. The reading process is performed forward and backward and it serves the first semi-iteration. At the end of this process, SISO 2 reads forward and backward from the memory blocks corresponding to V$_2$(X'_k) and $\Lambda^i(Z'_k)$ vectors in order to perform the second semi-iteration.

Vector V$_1$(X_k) is obtained by adding the input vector $\Lambda^i(X_k)$ with the extrinsic information vector W(X_k). After having the input data ready, SISO 1 starts the decoding process. At the output, the LLRs are available sequentially, at 8 clock periods distance. Performing the subtraction between these LLRs and the extrinsic values W(X_k), the vector V$_2$(X_k) is computed and then stored into its corresponding memory. The interleaving process is started and the re-ordered LLRs V$_2$(X'_k) are stored in their memory, where the corresponding values for the 3 tail bits X'_{k+1}, X'_{k+2}, X'_{k+3} are also added on the last memory locations. The second semi-iteration can start at this point. The same SISO unit is used, but reading this time data inputs from the other memory blocks. As one can see from Fig. 3, two switching mechanisms are included in the scheme. When in position 1, the memory blocks for V$_1$(X_k) and $\Lambda^i(Z_k)$ are used, while in position 2 the memory blocks for V$_2$(X'_k) and $\Lambda^i(Z'_k)$ become active.

At the output of the SISO unit, after each semi-iteration, K LLRs are obtained. The ones corresponding to the second semi-iteration are stored in the $\Lambda_2^o(X'_k)$ memory, then they are deinterleaved and finally they are stored in the $\Lambda_2^o(X_k)$ memory. Subtracting from these deinterleaved LLRs the values of V$_2$(X_k) vector, the extrinsic information W(X_k) is obtained. Also, if the decoder performs the last

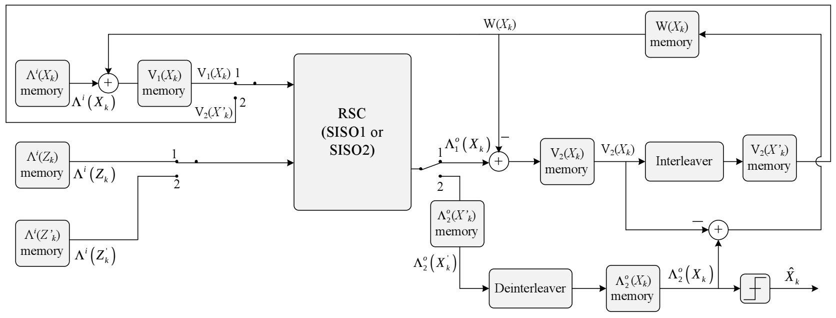

Figure 3. Proposed turbo decoder block scheme.

second semi-iteration, the hard decision is made over these deinterleaved LLRs, resulting this way the decoded bits.

In order to be able to handle all the data block dimensions, the used memory blocks have 6144 locations (this is the maximum data block length), except the ones storing the input data for RSCs, which have 6144 + 3 locations, including here also the tail bits. Each memory locations is 10 bits wide, the first bit being used for the sign, the next 6 bits representing the integer part and the last 3 bits indicating the fractional part. This format was decided studying the dynamic range of the variables (for the integer part) and the variations of the decoding performances (for the fractional part).

B. *The Interleaver*

The interleaver module is used both for interleaving and deinterleaving. The interleaved index is obtained based on a modified form of (2), i.e.,

$$\pi(i)=\{[(f_1+f_2\cdot i)\bmod K]\cdot i\}\bmod K. \qquad (12)$$

In order to obtain both functions, either the input data is stored in the memory in natural order and then it is read in interleaved order, either the input data is stored in the interleaved order and then it is read in natural order. Fig. 4 depicts the implementation solution for this module.

As one can observe from Fig. 4, the interleaved index computation is performed in three steps. First the value for $(f_1+f_2\cdot i)\bmod K$ is computed. This partial result is multiplied by natural order index *i* and then a new modulo *K* function is applied. In the first stage of this process, the remark that the formula is increased with f_2 for consecutive values of index *i* is used. This way, a register value is increased with f_2 at each new index *i*. If the resulted value is bigger than *K*, the value of *K* is subtracted from the register value. This processing is one clock period long, this being the reason why data is generated in a continuous manner.

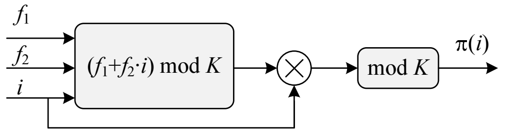

Figure 4. Proposed interleaver logic scheme.

In the second stage, a pipe-line multiplier is used for obtaining the result of the multiplication between index *i* and the first stage resulted value. The product result is obtained after 13 clock periods and it is 26 bits wide. In the third stage this result is compared with values 2^nK, with *n* between 13 and 0. Less subtraction for computing modulo *K* function are performed this way, the total number of clock periods being reduced from 6124 to 13. At the end of this third stage the interleaved indexes are obtained.

C. *The SISO module*

The internal SISO scheme is presented in Fig. 5. One can notice both the un-normalized metric computing blocks ALPHA (forward) and BETA (backward), and the transition metric computing block GAMMA, which in addition includes the normalization function (subtract the metrics for the first state from all the other metrics). The L block computes the output LLRs, which are normalized by the NORM block. The MUX-MAX block selects inputs corresponding to the forward or backward recursion and computes the maximum function. The MEM BETA block stores the backward metrics, which are computed before forward metrics. The metric normalization is required to preserve the dynamic range. Without normalization, the forward and backward metric width should be wider in order to avoid saturation, which means more memory blocks, more complex arithmetic (i.e., more used resources), and lower frequency (as an overall consequence). Hence, reducing the logic levels by eliminating the normalizing procedure does not increase the system performances.

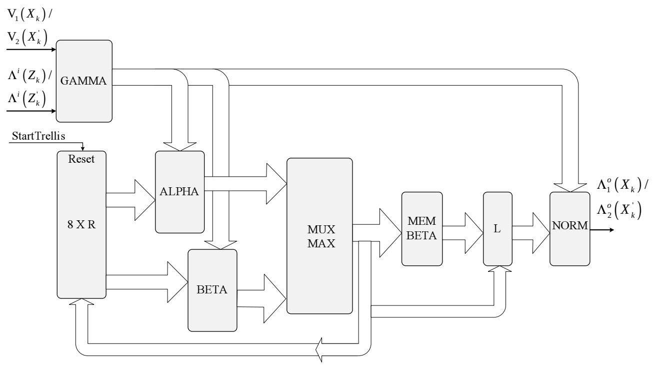
V$_1$(X_k) / V$_2$(X'_k)
$\Lambda^i(Z_k)$ / $\Lambda^i(Z'_k)$
GAMMA
StartTrellis
Reset
8 X R
ALPHA
BETA
MUX MAX
MEM BETA
L
NORM
$\Lambda_1^o(X_k)$ / $\Lambda_2^o(X'_k)$

Figure 5. Proposed SISO block scheme.

The ALPHA, BETA, and GAMMA blocks are implemented in a dedicated way. Each metric corresponding to each state is computed separately, not using the same function with different input parameters.

Consequently, 16 equations should be used for transition metric computation (2 possible transitions for each of the 8 states from a stage). In fact, only 4 equations are needed [as indicated in (5)]; moreover, from these 4 equations one of them leads to zero value, so that the computational effort is minimized for this implementation solution.

V. Implementation Results

A. Performances

The used hardware programming language is Very High Speed Hardware Description Language (VHDL). For the generation of RAM/ ROM memory blocks Xilinx Core Generator 11.1 was used. The simulations were performed with ModelSIM 6.5. The synthesis process was done using Xilinx XST from Xilinx ISE 11.1. Using these tools, the obtained system frequency when implementing the decoding structure on a Xilinx XC5VFX70T-FFG1136 chip is around 210 MHz. The occupied area is around 1000 (8.92%) slices from a total of 11200, while the used 18Kb memory blocks number is 32 from a total of 296.

B. Simulations

The following performance curves were obtained using a finite precision Matlab simulator. This approach was selected because the Matlab simulator produces exactly the same outputs as the ModelSIM simulator, while the simulation time is smaller.

All the simulation results are using the Max Log MAP algorithm, and the results are presented for different types of decoding parameters variations. All pictures describe the Bit Error Rate (BER) versus Signal-to-Noise Ratio (SNR) expressed as the ratio between the energy per bit and the noise power spectral density.

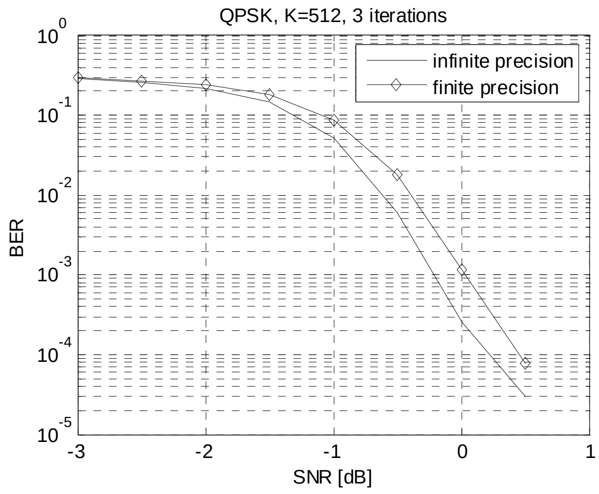

Figure 6. Finite precision vs. infinite precision.

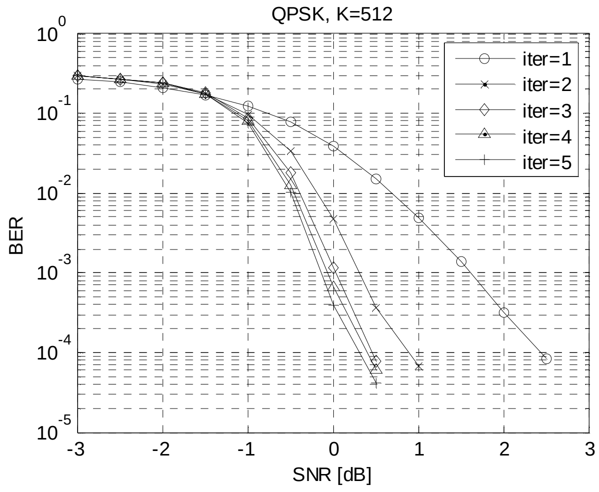

Figure 7. Decoding performances vs. number of iterations.

Fig. 6 depicts the obtained performances when executing the decoding process of the same input data, in infinite precision and in finite precision. For finite precision, as mentioned before, a 10 bit format was used, one bit for the sign, 6 bits for the integer part and 3 bits for the fractional part. In these simulations, *K*=512 bits, the used modulation is QPSK, and the number of turbo iterations is set to 3.

Fig. 7 depicts the performances improvement when the number of turbo iterations is increased. One can observe that after a certain number of turbo iterations the decoding improvement is not significant anymore and thus the added decoded latency is not justified. In these simulations, *K*=512 bits, the used modulation is QPSK, and the number of turbo iterations is increased from 1 to 5.

Finally, Fig. 8 describes the decoding performances improvement when the data block size increases. For these simulations the used modulation is QPSK, the number of turbo iterations is 3, and the data block lengths are *K*=40,

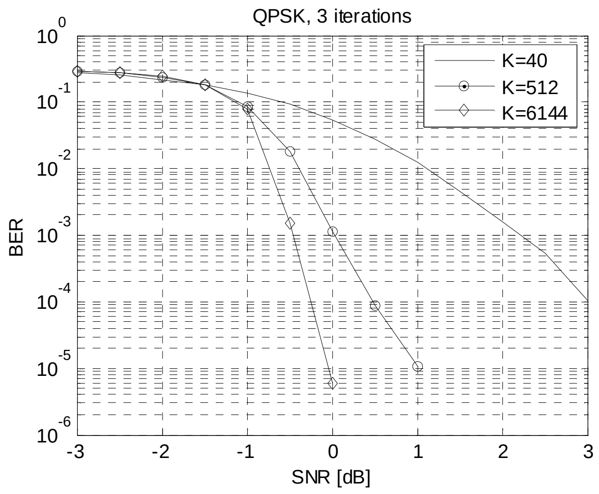

Figure 8. Decoding performances vs. block dimension.

K=512, and *K*=6144. One can observe an improvement of about 1.8 dB at BER = 10^{-2} between the smallest and the biggest block size defined by standard (*K*=40 and *K*=6144).

VI. Conclusions and Future Works

The most important aspects regarding the FPGA implementation of a CTC decoder for LTE systems were presented in this paper. Area and speed optimization solutions have been proposed based on the specific decoding scheme. A very efficient method of increasing the clock frequency was proposed, i.e., the normalization operation from the ALPHA/BETA updating loop was removed from that loop and distributed into the GAMMA block and also into the LLR computing block. Simulation and implementation results were given for different data block sizes and for different number of turbo iterations.

The perspective for a future work is to implement a stop criterion in order to reduce the decoding latency. A possible solution is the stop the decoding iterations when some indicators are not changing from one iteration to another.

Acknowledgments

This work was supported under the Grant UEFISCDI PN-II-RU-TE no. 7/5.08.2010.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes,” *IEEE Proceedings of the Int. Conf. on Communications*, Geneva, Switzerland, pp. 1064-1070, May 1993.

[2] C. Berrou and A. Glavieux, “Near Optimum Error Correcting Coding and Decoding: Turbo-Codes,” *IEEE Trans. Communications*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

[3] C. Berrou and M. Jézéquel, “Non binary convolutional codes for turbo coding,” *Electronics Letters,* vol. 35, no. 1, pp. 9-40, Jan. 1999.

[4] Third Generation Partnership Project. 3GPP home page. www.3gpp.org, last accessed on November 2011.

[5] 3GPP TS 36.212 V8.7.0 (2009-05) Technical Specification, “3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8).”

[6] F. Khan, *LTE for 4G Mobile Broadband*, Cambridge University Press, New York, 2009.

[7] M. C. Valenti and J. Sun, “The UMTS Turbo Code and an Efficient Decoder Implementation Suitable for Software-Defined Radios,” *International Journal of Wireless Information Networks,* Vol. 8, No. 4, pp. 203-216, October 2001.

[8] “Xilinx Virtex 5 family user guide,” retrieved from www.xilinx.com on January 2011.

[9] “Xilinx ML507 evaluation platform user guide,” retrieved from www.xilinx.com on January 2011.

[10] P. Robertson, E. Villebrun, and P. Hoeher, “A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain,” *Proc. IEEE International Conference on Communications* (ICC’95), Seattle, pp. 1009-1013, June 1995.

[11] S. Papaharalabos, P. Sweeney, and B. G. Evans, “Constant log-MAP decoding algorithm for duo-binary turbo codes,” *Electronics Letters Volume 42*, Issue 12, pp. 709 – 710, June 2006.

[12] J. F. Cheng and T. Ottosson, “Linearly approximated log-MAP algorithms for turbo decoding,” *Vehicular Technology Conference Proceedings,* 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st Volume 3, pp. 2252 – 2256, May 2000.

[13] C. Anghel, A. A. Enescu, C. Paleologu, and S. Ciochina, “CTC Turbo Decoding Architecture for H-ARQ Capable WiMAX Systems Implemented on FPGA,” “*Ninth International Conference on Networks*” ICN 2010, Menuires, France, April 2010.