Chapter 2

Efficient FPGA Implementation of a CTC Turbo Decoder for WiMAX/LTE Mobile Systems

Cristian Anghel, Cristian Stanciu and
Constantin Paleologu

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/67017

Abstract

This chapter describes the implementation on field programmable gate array (FPGA) of a turbo decoder for 3GPP long-term evolution (LTE) standard, respectively, for IEEE 802.16-based WiMAX systems. We initially present the serial decoding architectures for the two systems. The same approach is used; although for WiMAX the scheme implements a duo-binary code, while for LTE a binary code is included. The proposed LTE serial decoding scheme is adapted for parallel transformation. Then, considering the LTE high throughput requirements, a parallel decoding solution is proposed. Considering a parallelization with $N = 2^p$ levels, the parallel approach reduces the decoding latency N times versus the serial decoding one. For parallel approach the decoding performance suffers a small degradation, but we propose a solution that almost eliminates this degradation, by performing an overlapped data block split. Moreover, considering the native properties of the LTE quadratic permutation polynomial (QPP) interleaver, we propose a simplified parallel decoder architecture. The novelty of this scheme is that only one interleaver module is used, no matter the value of N, by introducing an even-odd merge sorting network. We propose for it a recursive approach that uses only comparators and subtractors.

Keywords: LTE, WiMAX, turbo decoder, single interleaver, Max LOG MAP, parallel architecture, FPGA

1. Introduction

The channel coding theory was intensively studied during the last decades, but the interest on this topic increased even more following the pioneering work of Berrou et al. on turbo codes [1–3].

In their early existence, the turbo codes proved to obtain great decoding performances, so that they were used in many standards as recommendations. They transformed into a more appealing solution once the processing capacity increased for the field programmable gate array (FPGA) and digital signal processor (DSP). Their implementation complexity was not prohibitive anymore, this allowing them to become mandatory.

In this context, the Third-Generation Partnership Project (3GPP) organization early proposed these novel coding techniques. It should be mentioned that turbo codes were introduced in standard by the first version of Universal Mobile Telecommunications System (UMTS) technology (in 1999). Moreover, the next UMTS releases (the following high-speed packet access) contributed with new and interesting features, while turbo coding remained still unchanged. Furthermore, several modifications were introduced by the long-term evolution (LTE) standard. Even if they were not significant as volume, their importance arose in terms of concept. In this framework, the 3GPP proposed for LTE a new interleaver scheme, while maintaining exactly the same coding structure as in UMTS. Also, the turbo codes were introduced by the Institute of Electrical and Electronics Engineers (IEEE) in 802.16 standards, known as the base for WiMAX systems.

In Ref. [4], an UMTS dedicated turbo decoding binary scheme is developed, whereas for WiMAX systems a similar duo-binary architecture is presented in Refs. [5] and [6]. Thanks to the new LTE/LTE-advanced (LTE-A) interleaver, the decoding performances are improved, as compared to the ones corresponding to the UMTS standard. In addition, the new LTE interleaver comes with native properties suited for a parallel decoding approach inside the algorithm, thus taking advantage on the main idea brought by turbo decoders (i.e., exchanging the extrinsic values between the two decoding units). In Ref. [7], a serial decoding scheme implemented on FPGA is presented. However, parallelization is still required when high throughput is required, as in the particular case of LTE systems using diversity techniques.

In the past years, many interesting parallel decoder schemes were studied by the researchers. In this context, the obtained results are measured on two directions. The direction number 1 is represented by the decoding performance degradation between the parallel and the serial solutions. The direction number 2 is the hardware resources occupied for such parallel decoder implementation. In Ref. [8], a first group of parallel decoding solutions is presented. It is based on the classical maximum a posteriori (MAP) algorithm. This method passes through the trellis twice, first time to compute the forward state metrics (FSM) and the second time to obtain the backward state metrics (BSM) and simultaneously the log likelihood ratios (LLR). Following this approach, several approaches were developed in order to reduce the theoretical latency of the decoding process of $2K$ clock periods for each semi-iteration (where K is the data block length).

In Refs. [9] and [10], a second set of parallel architectures that take advantage of the quadratic permutation polynomial (QPP) interleaver algebraic-geometric properties is described. In these works, efficient hardware implementations of the QPP interleaver are proposed. However, the parallelization factor N still represents the number of used interleavers in the developed architectures.

In Ref. [11], a third approach was reported, which consists in using a folded memory. All the data needed for parallel processing are stored on the same time. On the other hand, the main

challenge of this kind of implementation is to correctly distribute the data to each decoding unit, once a memory location containing all N values is read. In order to solve this issue, an architecture based on two Batcher sorting networks was proposed. However, even in this approach, N interleavers are still needed to generate all the interleaved addresses that input the master network.

In this chapter, we present the optimized implementations for serial architectures for WiMAX and LTE turbo decoding schemes. Then, for LTE systems, we describe a parallel decoding architecture introduced in Refs. [12] and [13], which also relies on a folded memory-based approach. Nevertheless, the main difference as compared to the already existing solutions presented above is that our proposed approach includes only one interleaver. Additionally, with an even-odd merge sorting unit [14, 15], the parallel architecture maintains the same structure as the serial one, the only difference being given by the fact that the soft-input soft-output (SISO) decoding unit is included N times in the scheme. The block memory number and dimensions remain unchanged between the two proposed decoding structures. In terms of decoding performance, the obtained results for the serial and parallel approaches are almost similar. We propose an overlapped data block split that reduces the small degradation introduced by the parallel architecture.

Finally, we present throughput and speed results obtained when targeting a XC5VFX70T [16] chip on Xilinx ML507 [17] board. Moreover, we provide simulation curves for the three considered cases, i.e., serial decoding, parallel decoding and parallel decoding with overlap.

2. The coding scheme

2.1. WiMAX systems

Section 8.4 from 802.16 standard [18] presents the coding scheme on the basis of which the proposed decoder is implemented. **Figure 1** shows the duo-binary encoder. The native coding rate is 1/3. In order to obtain other coding rates, a puncturing block must be used. Accordingly, a depuncturing block must be added to the receiver architecture.

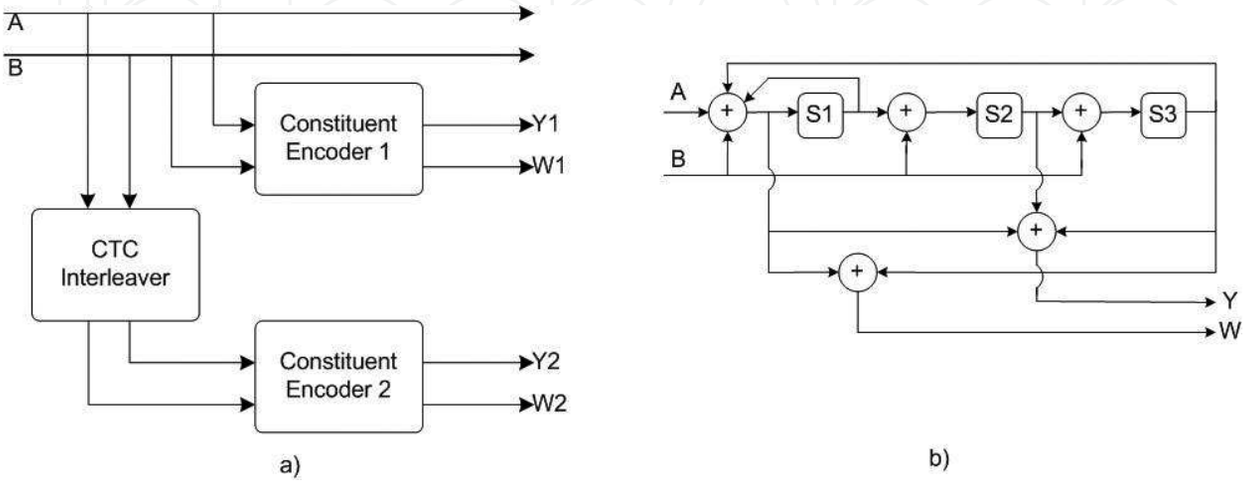

Figure 1. (a) 802-16e turbo coding scheme; (b) constituent encoder.

Let us define the following parameters: coding rate R; block dimension (in pairs of bits, i.e., di-bits) K, which is computed independent of a coding rate, as a function of the uncoded block size; the number of iterations L, i.e., the latency *Latency* (in clock periods); information bits rate R_b [Mbps]; and system clock frequency F_{clk} [MHz].

As mentioned in Ref. [6], the main problem of a convolutional turbo code (CTC) decoder implementation is represented by the amount of required hardware resources. Moreover, in order to reach the targeted high data rate, the system clock has to be fast. Equation (1) presents the decoding throughput.

$$R_b = \frac{2K}{\text{Latency } T_{clk}} \tag{1}$$

For a fixed latency algorithm, according to Eq. (1), the output throughput is improved when achieving a higher clock frequency. Another way is to reduce latency using a parallel architecture; however, this increases the occupied area and may lead to a smaller clock frequency due to longer routes. Moreover, another direct constraint is the significant memory needed for storing data. This issue also affects the frequency, since a large number of used memory blocks leads to a large resource spread on chip and, obviously, longer routes.

Taking into account the previously mentioned aspects, we can conclude that all the parameters presented above are related, so that a global optimization is not possible. Consequently, we have chosen to balance each direction in order to meet throughput requirements.

2.2. LTE systems

A classic turbo coding scheme is presented in the 3GPP LTE specification, including two constituent encoders and one interleaver module (**Figure 2**). The data block C_k can be observed at the input of the LTE turbo encoder. The K bits from this input data block are transferred at the output, as systematic bits, in the steam X_k. At the same time, the first constituent encoder processes the input data block, resulting the parity bits Z_k, whereas the second constituent encoder processes the interleaved data block C'_k, resulting the parity bits Z'_k. Combining the systematic bits and the two streams of parity bits, we obtain the following sequence (at the output of the encoder): X_1, Z_1, Z'_1, X_2, Z_2, Z'_2, …, X_K, Z_K, Z'_K.

In order to drive back the constituent encoders to the initial state (at the end of the coding process), the switches from **Figure 2** are moved from position A to position B. Since the final states of the two constituent encoders are not the same (different input data blocks produce different final state), this switching procedure generates tail bits for each encoder. These tail bits are sent together with the systematic and parity bits, thus 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}.

As it was previously mentioned and discussed in Ref. [7], the LTE turbo coding scheme introduces a new interleaving structure. Thus, the input sequence is rearranged at the output using:

http://dx.doi.org/10.5772/67017

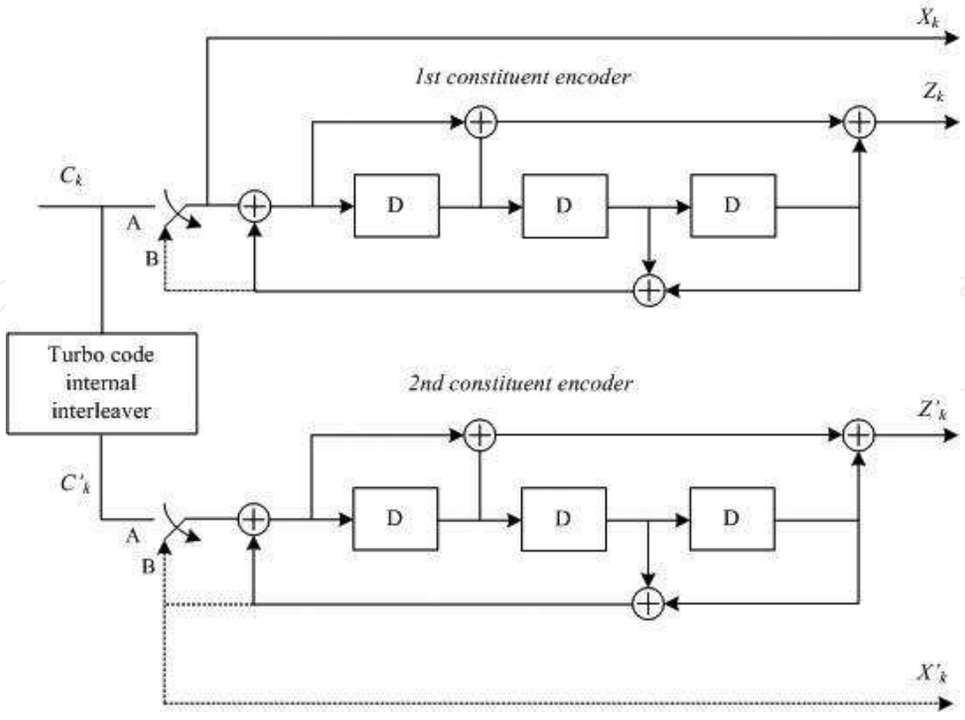

Figure 2. LTE turbo coding scheme.

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

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

$$\pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \mathrm{mod} K \tag{3}$$

The input block length K and the parameters f_1 and f_2 are provided in Table 5.1.3-3 in Ref. [19].

3. The decoding algorithm

3.1. WiMAX systems

The decoding architecture consists of two decoding units called constituent decoders. Each such unit receives systematic bits (in natural order or interleaved) and parity bits, as shown in **Figure 1**.

The block diagram implements a maximum-logarithmic-maximum A posteriori (Max-Log-MAP) algorithm. For the case of turbo binary codes, the decoder scheme will represent, in the log likelihood ratio (LLR) space, each binary symbol as a single likelihood ratio. But in the situation of turbo duo-binary codes, the decoding unit requires three likelihood ratios in the same space. If we consider the duo-binary pair A_k and B_k, the LLRs may be computed as:

$$\Lambda_{a,b} = (A_k, B_k) = \log \frac{P(A_k = a, B_k = b)}{P(A_k = 0, B_k = 0)} \tag{4}$$

where (*a*,*b*) are (0,1), (1,0), or (1,1). The ratio set is updated by each decoding unit (constituent decoder) for each input pair, using the corresponding LLRs and parity bits, also seen as LLRs. Then, the output LLRs minus the input LLRs provides the extrinsic values. The trellis for a duo-binary code contains eight states, each such state with four inputs and four outputs, as presented in **Figure 3**. Using the systematic and parity pairs LLRs, for each branch, the metric $\gamma_k(S_i \rightarrow S_j)$ is computed, i.e.,

$$\gamma_k(S_i \rightarrow S_j) = \Lambda^i_{a,b}(A_k, B_k) + w\Lambda(W_k) + y\Lambda(Y_k) \tag{5}$$

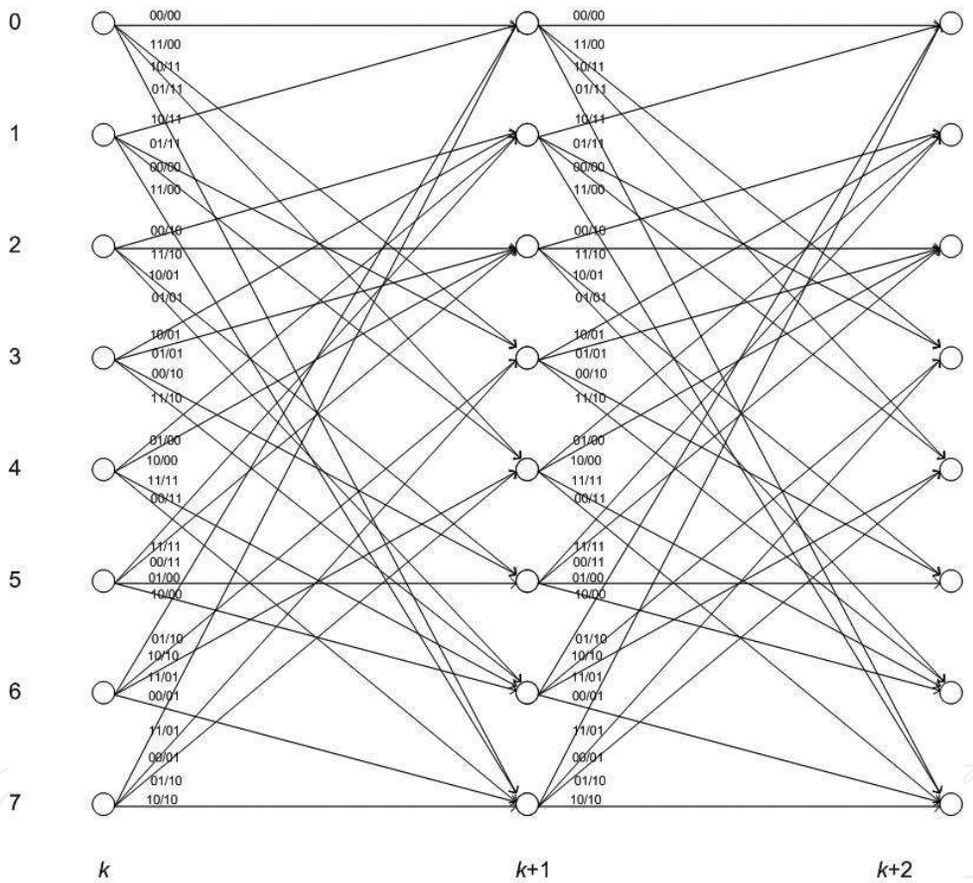

Figure 3. WiMAX decoder trellis.

The constituent decoder (**Figure 4**) performs the corresponding processing forward and backward over the trellis. When moving forward, the decoder computes the unnormalized metric $\alpha'_{k+1}(S_j)$ corresponding to each computed normalized metric $\alpha_k(S_i)$ associated with state S_i, using (**Figure 4**)

$$\alpha'_{k+1}(S_j) = \max_{S_i \rightarrow S_j} \{\alpha_k(S_i) + \gamma_k(S_i \rightarrow S_j)\} \tag{6}$$

http://dx.doi.org/10.5772/67017

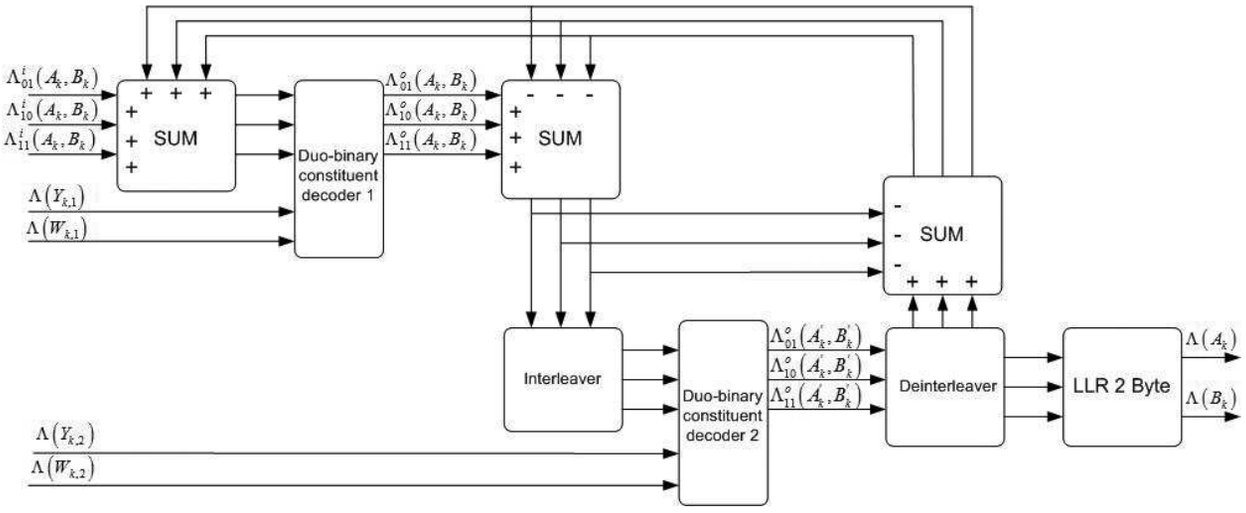

Figure 4. Decoder block scheme.

where the operator "maximum" is executed over all four branches entering the state S_j at the time stamp $k + 1$. Once the metrics for all states are updated at time stamp $k + 1$, the normalization versus the state S_0 value is made by the decoder. Analogously to forward processing, for backward moving, the decoder computes:

$$\beta_k^{'}(S_i) = \max_{S_i \to S_j} \{\beta_{k+1}(S_j) + \gamma_k(S_i \to S_j)\} \tag{7}$$

where the operator "maximum" and the normalization method are similar to Eq. (6).

The initialization with null values is carried out for all the forward and backward metrics at all states. Once the new values are computing and stored, the decoding unit executes the second step in the decoding procedure, i.e., the LLRs computing as in Eq. (4). The decoding unit starts by computing the likelihood ratio for each branch

$$Z_k(S_i \to S_j) = \alpha_k(S_i) + \gamma_k(S_i \to S_j) + \beta_{k+1}(S_j) \tag{8}$$

and continues with the value

$$t_k(a,b) = \max_{S_i \to S_j:(a,b)} \{Z_k\} \tag{9}$$

where the operator "maximum" is computed over all eight branches generated by the pair (a, b). At the end, the output LLR is computed as

$$\Lambda_{a,b}^{o}(A_k, B_k) = t_k(a,b) - t_k(0,0) \tag{10}$$

The decoding procedure is executed for a decided number of iterations or until a convergence criterion is reached. Then, a final decision is taken over the bits. This is achieved by computing for each bit from the pair (A_k, B_k) the corresponding LLR:

$$\Lambda(A_k) = \max\{\Lambda_{1,0}^{o}(A_k, B_k), \Lambda_{1,1}^{o}(A_k, B_k)\} - \max\{\Lambda_{0,0}^{o}(A_k, B_k), \Lambda_{0,1}^{o}(A_k, B_k)\} \tag{11}$$

$$\Lambda(B_k) = \max\{\Lambda_{0,1}^{o}(A_k, B_k), \Lambda_{1,1}^{o}(A_k, B_k)\} - \max\{\Lambda_{0,0}^{o}(A_k, B_k), \Lambda_{1,0}^{o}(A_k, B_k)\}, \tag{12}$$

where $\Lambda^o_{0,0}(A_k, B_k) = 0$. Finally, by comparing each LLR with a null threshold, i.e., looking at the sign, the hard decision is made.

3.2. LTE systems

The decoding architecture for the LTE systems is presented in **Figure 5**. The two decoding units called recursive systematic convolutional (RSC) use theoretically the MAP algorithm. The MAP solution, a classical one, ensures the best decoding performances. Unfortunately, at the same time, it is characterized by an increased implementation complexity and also it may include variables with a large dynamic range. These are the reasons why the classical solution with the MAP algorithm is used only as a reference for the expected decoding performance. When it comes to real implementation, new suboptimal algorithms have been studied: Logarithmic MAP (Log MAP) [20], Max Log MAP, Constant Log MAP (Const Log MAP) [21] and Linear Log MAP (Lin Log MAP) [22].

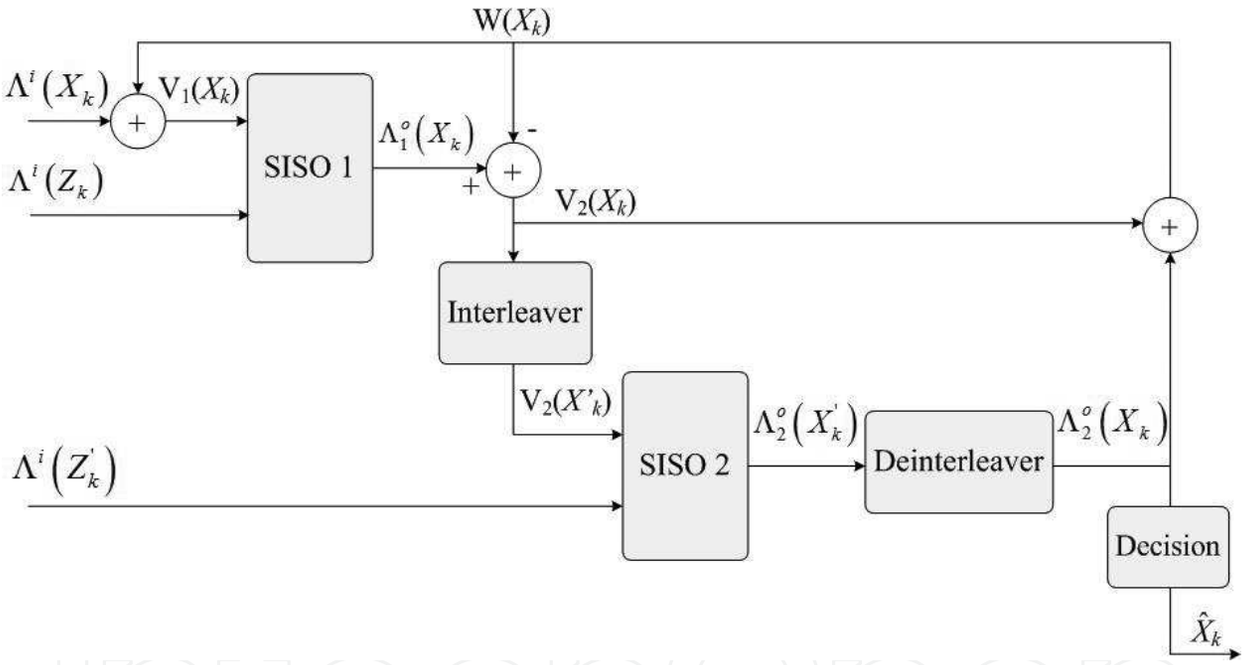

Figure 5. LTE turbo decoder.

For the LTE systems, we consider a decoding architecture based on the Max Log MAP algorithm. This suboptimal algorithm overcomes the problems of implementation complexity and dynamic range by paying the price of lower decoding performance when compared with the MAP algorithm. However, this degradation can be maintained inside some accepted limits. Starting from the Jacobi logarithm, only the first term is used by the Max Log MAP algorithm, i.e.,

$$\max^*(x, y) = \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|y-x|}) \approx \max(x, y) \ . \tag{13}$$

The trellis diagram for the turbo decoding architecture of the LTE systems contains eight states, as presented in **Figure 6**. Each state of the diagram has two inputs and two outputs. The branch metric between the states S_i and S_j is

http://dx.doi.org/10.5772/67017

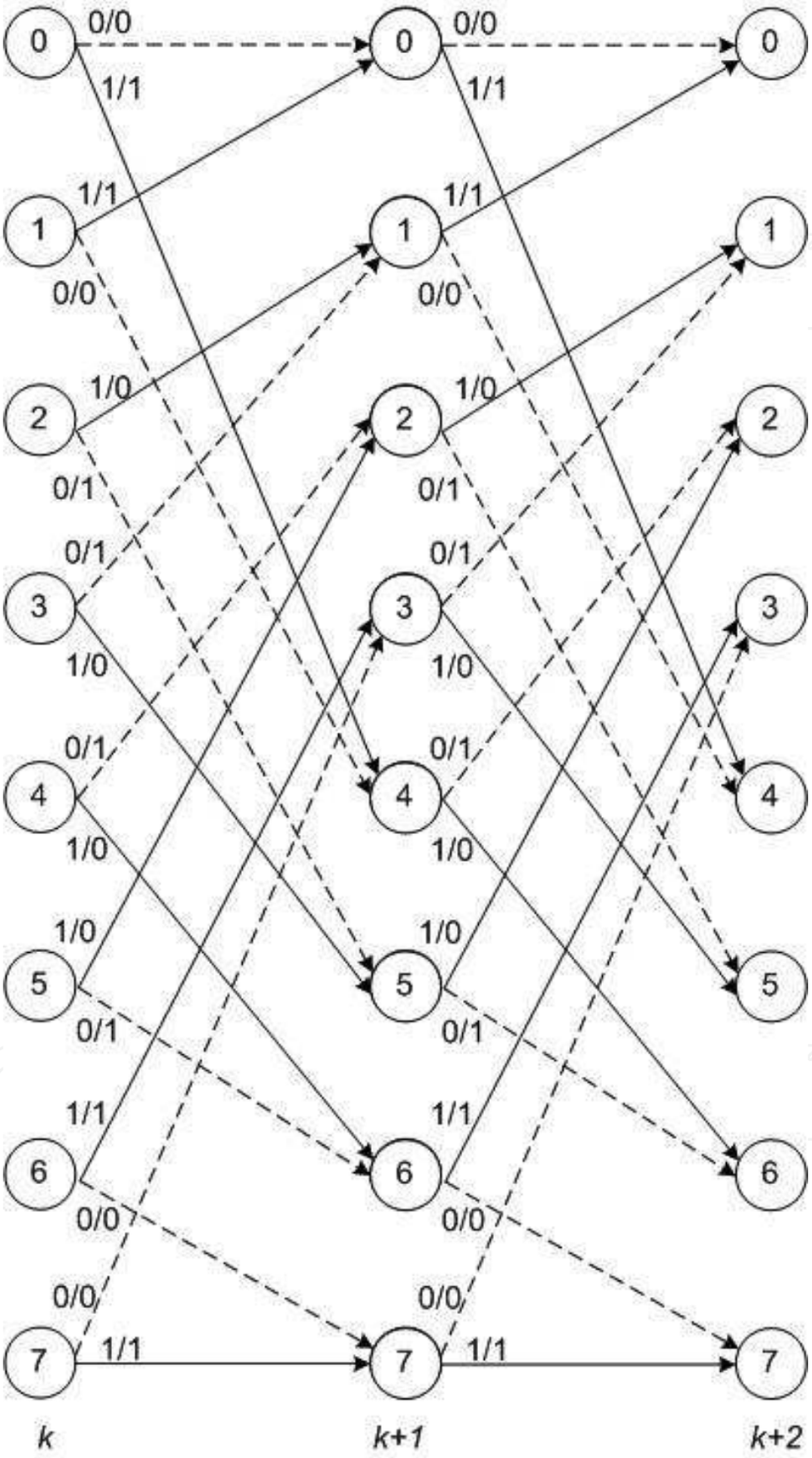

Figure 6. LTE turbo coder trellis.

$$\gamma_{ij} = \mathrm{V}(X_k)X(i,j) + \Lambda^i(Z_k)Z(i,j)\ , \tag{14}$$

where $X(i,j)$ and $Z(i,j)$ are the data, respectively, the parity bits, both associated with one branch and $\Lambda^i(Z_k)$ is the LLR for the input parity bit. For SISO 1 decoding unit, this input LLR is $\Lambda^i(Z_k)$, whereas for SISO 2 it becomes $\Lambda^i(Z_k^{'})$. For SISO 1, $\mathrm{V}(X_k) = \mathrm{V}_1(X_k) = \Lambda^i(X_k) + \mathrm{W}(X_k)$, whereas for SISO 2, $\mathrm{V}(X_k) = \mathrm{V}_2(X_k^{'}) = \mathrm{IL}\{\Lambda_1^o(X_k) + \mathrm{W}(X_k)\}$, where "IL" operator denotes the interleaving procedure. In **Figure 5**, $W(X_k)$ is the *extrinsic information*, whereas $\Lambda_1^o(X_k)$ and $\Lambda_2^o(X_k^{'})$ are the output LLRs generated by the two SISOs.

Looking at the LTE turbo encoder trellis, one can notice that between two states, there are four possible values for the branch metrics:

$$\begin{aligned} &\gamma_0 = 0 \\ &\gamma_1 = \mathrm{V}(X_k) \\ &\gamma_2 = \Lambda^i(\mathrm{Z}_k) \\ &\gamma_2 = \mathrm{V}(X_k) + \Lambda^i(Z_k)\ . \end{aligned} \tag{15}$$

The LTE decoding process follows a similar approach as for WiMAX systems, i.e., it moves forward and backward through the trellis.

3.2.1. Backward recursion

The algorithm moves backward over the trellis computing the metrics. The obtained values for each node are stored in a normalized manner. They will be used for the LLR computation once the algorithm will start moving forward through the trellis. We name $\beta_k(S_i)$ the backward metric computed at the kth stage, for the state S_i, where $2 \le k \le K+3$ and $0 \le i \le 7$. For the backward recursion, the initialization $\beta_{K+3}(S_i) = 0, 0 \le i \le 7$ is used at the stage $k = K + 3$. For the rest of the stages $2 \le k \le K + 2$, the computed backward metrics are

$$\hat{\beta}_k(S_i) = \max\{\left(\beta_{k+1}(S_{j1}) + \gamma_{ij1}\right),\ \left(\beta_{k+1}(S_{j2}) + \gamma_{ij2}\right)\}\ , \tag{16}$$

where S_{j1} and S_{j2} are the two states from stage $k + 1$ connected to the state S_i from stage k and $\hat{\beta}_k(S_i)$ represents the unnormalized metric. Once the unnormalized metric $\hat{\beta}_k(S_0)$ is computed for state S_0, all the backward metrics for states $S_1 \ldots S_7$ are normalized as

$$\beta_k(S_i) = \hat{\beta}_k(S_i) - \hat{\beta}_k(S_0) \tag{17}$$

and then stored in the dedicated memory.

3.2.2. Forward recursion

When the backward recursion is finished, the algorithm moves forward through the trellis in the normal direction. This specific phase of the decoding is similar to the one for Viterbi algorithm. In this case, the storing procedure is needed only for the previous stage metrics,

i.e., for computing the current stage k metrics, only the forward metrics from the last stage $k-1$ are needed. We will name $\alpha_k(S_i)$ the forward metric corresponding to state at the stage k, where $0 \le k \le K-1$ and $0 \le i \le 7$. For the forward recursion, the initialization $\alpha_0(S_i) = 0,\ 0 \le i \le 7$ is used at the stage k = 0. For the rest of the stages $1 \le k \le K$, the unnormalized forward metrics are computed as

$$\hat{\alpha}_k(S_j) = \max\left\{\left(\alpha_{k-1}(S_{i1}) + \gamma_{i1j}\right), \left(\alpha_{k-1}(S_{i2}) + \gamma_{i2j}\right)\right\}, \tag{18}$$

where S_{i1} and S_{i2} are the two states from stage $k-1$ connected to the state S_j from stage k. Once the unnormalized metric $\hat{\alpha}_k(S_0)$ is computed for state S_0, all the forward metrics for states $S_1 \ldots S_7$ are normalized as

$$\alpha_k(S_i) = \hat{\alpha}_k(S_i) - \hat{\alpha}_k(S_0)\ . \tag{19}$$

The decoding algorithm can obtain now an LLR estimated for the data bits X_k since it has for each stage k the forward metrics just computed and also the backward metrics stored in the memory. For the first time, this LLR is obtained by computing the likelihood of the connection between the state S_i at stage $k-1$ and the state S_j at stage k as

$$Z_k(S_i \to S_j) = \alpha_{k-1}(S_i) + \gamma_{ij} + \beta_k(S_j)\ . \tag{20}$$

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

$$\Lambda^o(X_k) = \max_{(S_i \to S_j):X_i=1} \{Z_k(S_i \to S_j)\} - \max_{(S_i \to S_j):X_i=0} \{Z_k(S_i \to S_j)\}, \tag{21}$$

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

4. Proposed serial decoding scheme

4.1. WiMAX systems

One important remark about the decoding algorithm is that the outputs of one constituent decoder represent the inputs for the other constituent decoder. At the same time, knowing that the interleaver and deinterleaver procedures apply over the data blocks (so the complete block is needed) in a nonoverlapping manner will allow the usage of a single constituent decoder. This decoding unit operates time multiplexed and the corresponding proposed scheme is presented in **Figure 7**.

In **Figure 7**, we can identify storing requirements: the memory blocks that store data from one semi-iteration to another and the memory blocks used from one iteration to another. IL stands for the interleaver/deinterleaver procedure, while CONTROL is the management unit, controlling the decoder functionalities. This module provides the addresses used for

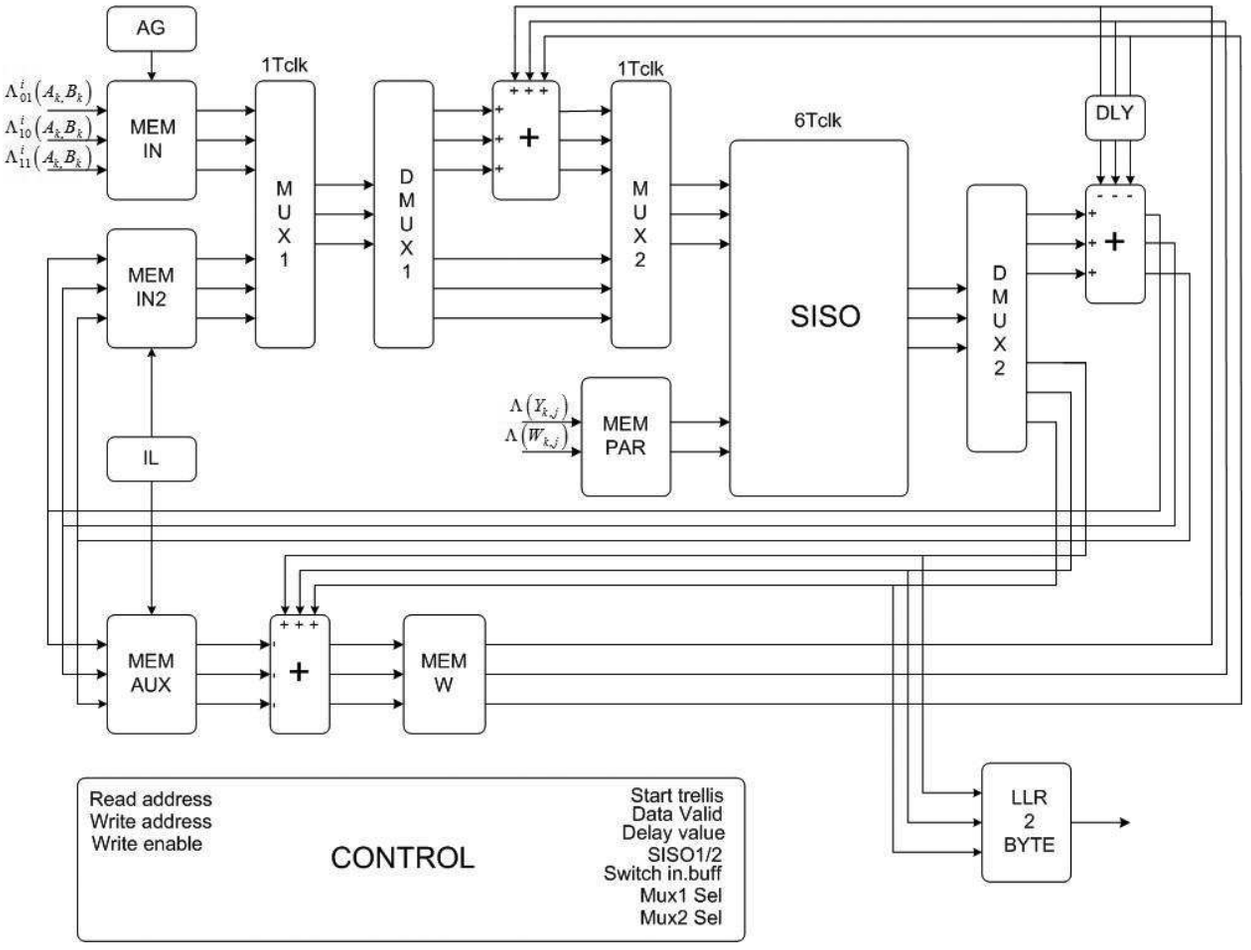

Figure 7. Proposed decoder scheme.

read and write, the signals used to trigger the forward and backward movements through the trellis, the selection for one of the two SISO units and also the control of MUX and DEMUX blocks. The input buffer is also selected since the decoding architecture can accept a new-encoded data block while still processing the previous one. The most important module shown in **Figure 7** is the SISO unit, which is the decoding structure. **Figure 8** depicts the block scheme of this decoding unit. One can observe the unnormalized metric computing modules BETA (backward) and ALPHA (forward) and the module GAMMA that computes the transition metric. This last one ensures also the normalization: the metrics values obtained for state S_0 are subtracted from the metrics values obtained for the states $S_1 \ldots S_7$. The output LLRs are computed inside the L module and normalized inside the NORM module. The MUX-MAX module provides the correct inputs when moving forward or backward through the trellis. It also computes the maximum function. The backward metrics are stored in MEM BETA memory during backwards recursion, their values being read when executing the forward recursion, in order to compute the estimated LLRs.

It is important to mention that some studies have been conducted regarding the normalization function. Trying to increase the system frequency (in order to reduce the decoding latency and

http://dx.doi.org/10.5772/67017

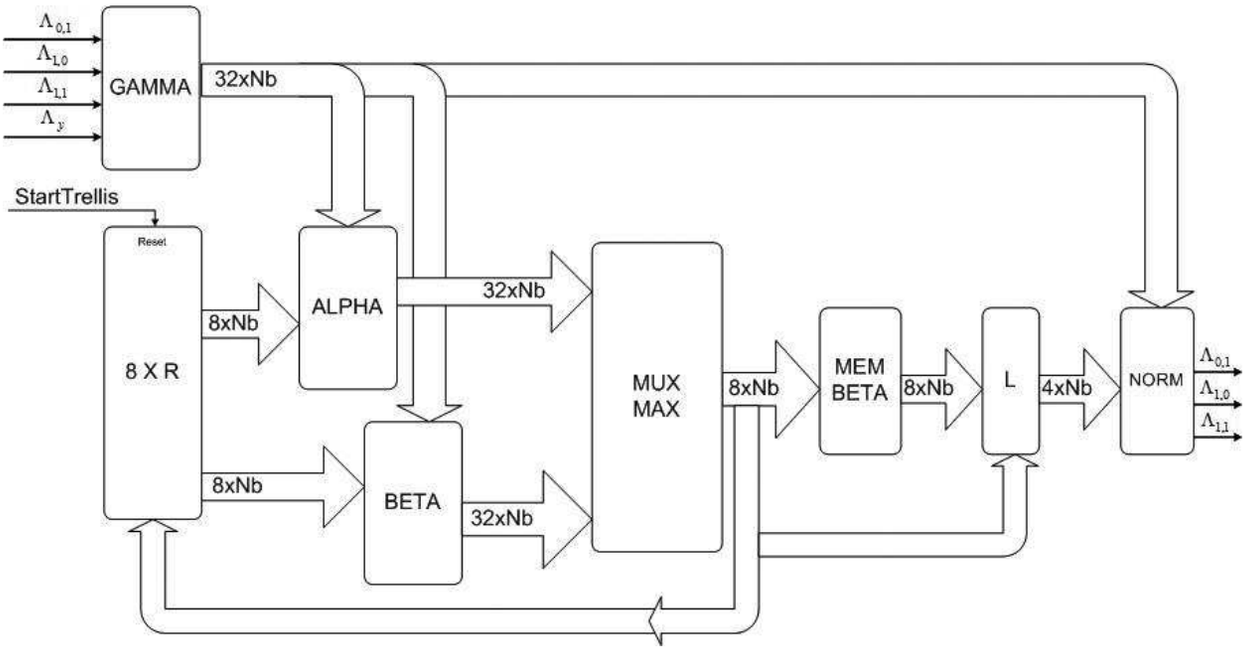

Figure 8. SISO block scheme.

so, to increase the decoded data throughput), one may think of removing the normalization and so to reduce the amount of logic on the critical path. This solution is not applicable because five extra bits would be needed for metrics values. From here more the memory blocks and more the complex arithmetic. Finally, all these will lead to a lower system frequency, so no benefit on this approach. On the other hand, we propose a dedicated approach to implement the metric computation blocks (ALPHA, BETA and GAMMA). Based on the trellis state, we identified the relations for each metric, 32 equations being used for transition metric computation (we remind that for each of the eight trellis states we have four possible transitions). Moreover, only 16 are distinct (the other 16 are the same) and from these 16, some are null. Using this approach, a complexity decrease is obtained.

Figure 9 depicts the timing diagram for the proposed SISO. This corresponds to the scenario with one SISO unit and some MUX and DEMUX blocks replacing the two SISO units from the theoretical decoding architecture (see **Figure 7**).

In **Figure 9**, R/W (K − 1:0) means reading/writing memory from addresses K − 1 to 0, R/W {IL (K − 1:0)} means reading/writing memory from interleaved addresses K − 1 to 0 and COMPUTE means that the block is processing the input data.

4.2. LTE systems

The same remark about the two SISO units from **Figure 5** working in a nonoverlapping manner applies for LTE systems as for WiMAX ones. The same approach is used, i.e., the proposed decoding architecture includes only one SISO unit and some MUX and DEMUX blocks. **Figure 10** depicts the block scheme of the proposed decoding architecture.

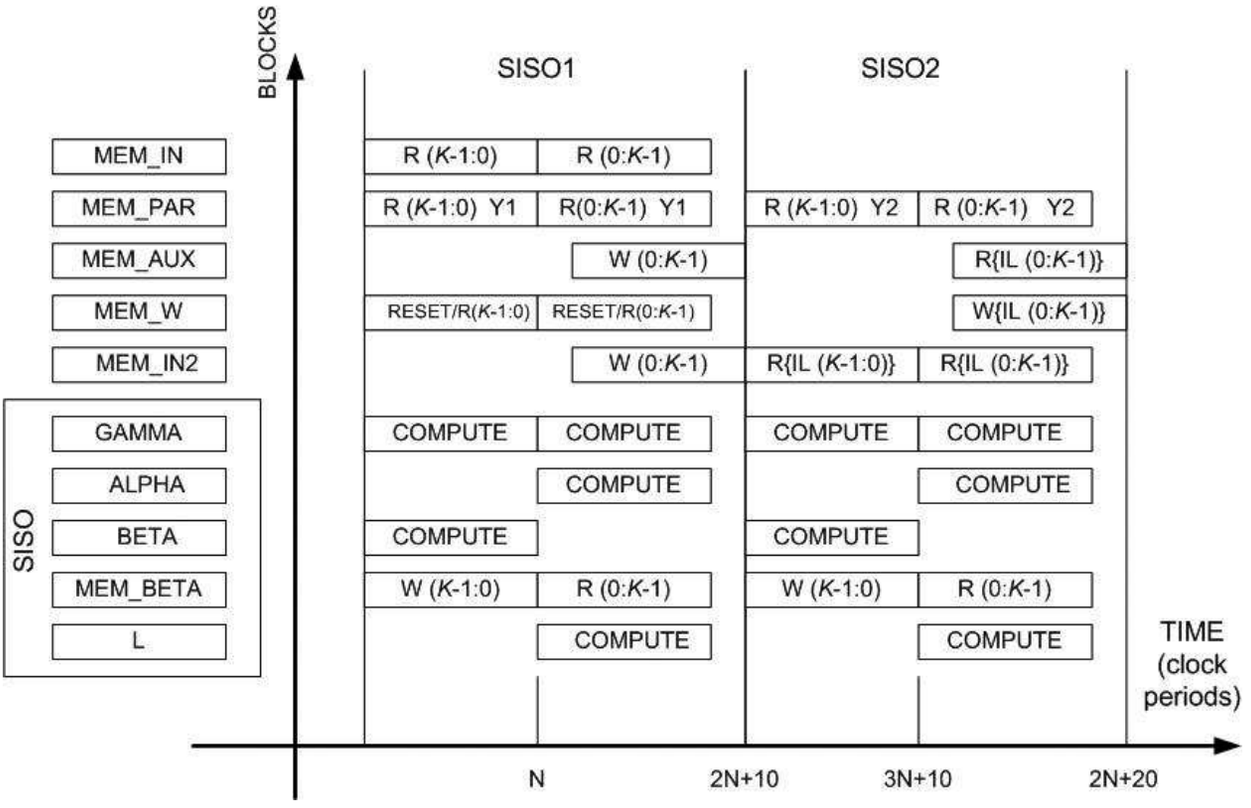

Figure 9. Time utilization for one turbo iteration.

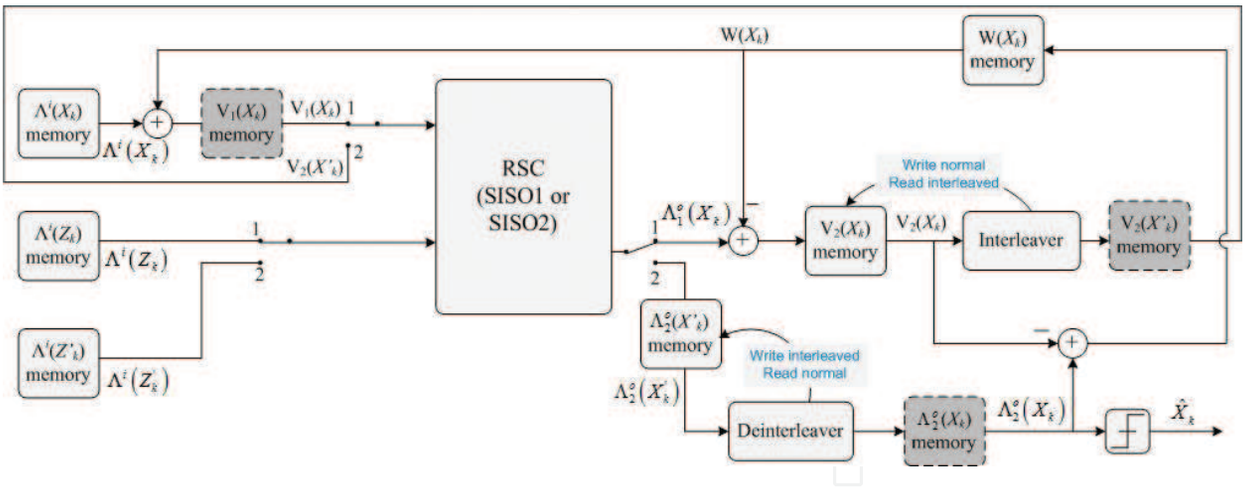

Figure 10. Proposed serial turbo decoder block scheme.

One can observe the memory blocks in **Figure 10**. Some are used to store data between two successive semi-iterations, respectively, between two successive iterations. Others, in dotted-line, are virtual memories used just to clarify the introduced notations. Moreover, the interleaver and deinterleaver modules are distinctively introduced in the scheme, but in fact they are the same. Both include a block memory called ILM (interleaver memory) and an interleaver. The novelty of this approach compared to the previous serial implementation proposed in Ref. [7] is the ILM. This memory will allow a fast transition to a parallel decoding architecture. The input data memories

(on the left side in **Figure 10**) and the ILM are switched buffers, allowing new data to be written while the previous block is still decoded. The ILM is filled with the interleaved addresses; at the same time, the new data are stored in the input memories. The saved addresses are then used as read addresses for the interleaver unit and as write addresses for the deinterleaver unit. Here, we detail the way the architecture from **Figure 10** works. The vectors $V_1(X_k) = \Lambda^i(X_k) + W(X_k)$ and $\Lambda^i(Z_k)$ are read from the corresponding memories by SISO 1. For the first semi-iteration, the memories are read in both directions, in order to ensure the forward and backward movements on the trellis. When this decoding phase is completed, the second semi-iteration starts, SISO 2 reads in both directions the memories storing the vectors $V_2(X_k^{'}) = IL\{V_2(X_k)\} = IL\{\Lambda_1^o(X_k) - W(X_k)\}$ and $\Lambda^i(Z_k^{'})$. IL stands again for the interleaver process.

In detail, SISO 1 reads the input memories and starts the decoding process, outputting the computed LLRs. Having the LLRs available and the extrinsic values, the vector $V_2(X_k)$ is computed and then stored in a normal order in the memory. The ILM content read in the normal order provides the reading addresses for $V_2(X_k)$ memory, emulating the interleaver process. The reordered LLRs $V_2(X'_k)$ are available, the corresponding values for the three tail bits X'_{K+1}, X'_{K+2} and X'_{K+3} being added at the end of this sequence. The same SISO unit acts now as SISO 2, this time reading data inputs from the other memory blocks. The two switching mechanisms from **Figure 10** change the position between these two semi-iterations (when in position 1, $V_1(X_k)$ and $\Lambda^i(Z_k)$ memories are active, while in position 2, $V_2(X'_k)$ and $\Lambda^i(Z_k^{'})$ memories are used).

The SISO unit provides at the end of each semi-iteration K values for the LLRs. The LLRs obtained after the second semi-iteration are stored in the $\Lambda_2^o(X_k^{'})$ memory (the content of ILM, already available for the $V_2(X_k)$ interleaver process, is used also as writing address for $\Lambda_2^o(X_k^{'})$ memory, after a delay is added).

The memories $\Lambda_2^o(X_k^{'})$ and $V_2(X_k)$ are read in the normal order to allow $W(X_k)$ computation; $W(X_k)$ is written in the corresponding memory and at the same time it is used for a new semi-iterations. In other words, the memory for $W(X_k)$ is updated during a semi-iteration. The time diagram for the proposed serial decoding architecture is presented in **Figure 11**, the intervals colored with gray indicating the writing periods for $W(X_k)$ memory. As mentioned in this chapter, the input memories and the ILM (the upper four memory blocks in the image) are switched buffers and they are filled with new data while the previous-coded block passes the last phase of its decoding process. The same notations as shown in **Figure 9** are used.

All the memory blocks in **Figure 10** have 6144 locations, this being the maximum coded data block length defined by the standard. Only the memory blocks with the input data for SISO units have 6144 + 3 locations because they store also the tail bits. All locations contain 10 bits. Using a Matlab simulator in finite precision, it has been observed that six bits are needed for the integer part, in order to cover the dynamic range of the variables and three bits are needed for the fractional part to maintain the decoding performance close to the theoretical one, with a certain accepted level of degradation. The 10th bit is for sign.

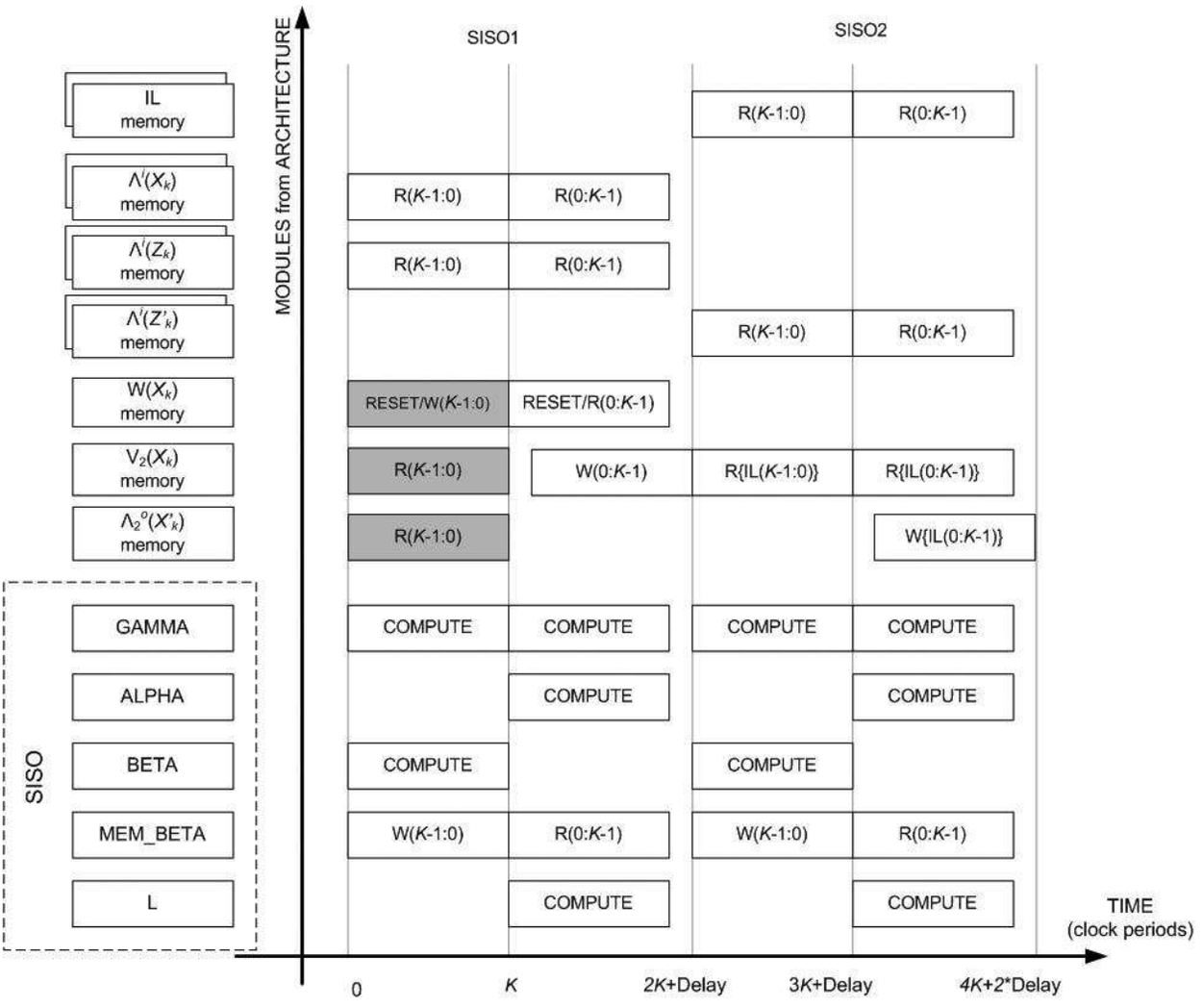

Figure 11. Time diagram for a serial turbo decoder.

The SISO decoding unit is similar to the one depicted in **Figure 8**. ALPHA and BETA modules compute the unnormalized forward metrics and the unnormalized backward metrics, respectively. The GAMMA module computes the transition metrics and executes also the normalization (the metrics for state S_0 are subtracted from the metrics corresponding to states S_1, …, S_7). The output LLRs are computed inside the L module and normalized by the NORM module. The selection of the inputs for forward and backward moving on the trellis and also the maximum function are executed by the MUX-MAX module. Finally, the MEM BETA module stores the backward metrics.

The L module produces the output log likelihood ratios. These are then normalized inside the NORM module. The MUX-MAX makes the inputs selection (for forward or backward trellis runs) and implements also the maximum operator. The MEM BETA module keeps the backward metrics corresponding values into the memory.

Using the same approach for both WiMAX and LTE proposed serial decoding architectures, the same remarks apply. So, for the LTE turbo decoder also, the normalization function allows

http://dx.doi.org/10.5772/67017

a reduced dynamic range for the variables. Trying to eliminate it, in order to reduce the number of logic levels on the critical path, will not lead to a higher system frequency because again, more memory blocks are required, more complex arithmetic (since variables are expressed on more bits) is used and finally, as an overall consequence, lower clock frequency is reported for the design.

And for ALPHA, BETA and GAMMA modules inside the SISO decoding unit, again the dedicated equations are used to compute the metrics. Sixteen such relations are implemented for transition metric computation (eight states in trellis with two possible transitions each). In fact, only four equations are distinct (as indicated in Eq. (15)]. And from these four equations, one of them is null. This way the computational effort is minimized for this proposed architecture.

The interleaving and deinterleaving procedures implement the same equation. The interleaved index is computed using a modified form of Eq. (3), i.e.,

$$\pi(i) = \{[(f_1 + f_2 \cdot i) \bmod K] \cdot i\} \bmod K \tag{22}$$

For the interleaving process, the data are written in the memory block in the natural order and then it is read in the interleaved order, while for the deinterleaver process the data are written in the interleaved order and then it is read in the natural order.

The computation in Eq. (22) is executed in three phases. First, the value for $(f_1 + f_2 \cdot i) \bmod K$ is obtained. The index i (describing the natural order) multiplies this partial result and the obtained value is passed once again through modulo K block. And as a remark for this computation: the formula is increased with f_2 for consecutive values of index i. So a register adds f_2 for each new index. If the register current value is higher than K, K is subtracted and the result is placed back in the register. This processing requires one system clock period, the results being generated in a continuous manner.

5. Proposed parallel decoding scheme

The serial architecture described in **Figure 10** for LTE systems can be reorganized in a parallel setup, by instantiating the RSC SISO module N times in the structure. We propose a configuration that concatenates the N values associated with the N RSCs and employs a single memory location for all the memories in the scheme. The K locations with 10 bits per location (corresponding to the serial architecture) are replaced by K/N positions with $10N$ bits per position (working for the parallel format).

The most important benefit brought by the proposed serial decoding scheme is the single usage of the interleaver module before the decoding stage. The ILM is updated, each time a new data block enters the decoder, while the previous block is still being decoded. This approach prepares a fast and simple transition to the parallel scheme. Considering that the factor N is known, the ILM will have K/N locations, with N values being written at each location (i.e., the ILM can be prepared for the parallel processing that follows). As

mentioned in Ref. [16], a Virtex 5 block memory can be organized from a configuration of 32k locations × 1 bit to a setup of 512 locations × 72 bits. In the costliest scenario (i.e., K = 6144), based on the N values and representing the stored values on 10 bits, the parallel ILM can be employed as:

- 768 locations × 80 bits
- 1536 locations × 40 bits
- 3072 locations × 20 bits
- 6144 locations × 10 bits

Only two BRAMs are used, the same as in the case of serial ILM.

Figure 12 shows the ILM working principle. As one can observe, during the writing procedure, each index i from 0 to $K - 1$ generates a corresponding interleaved value. All the computed values are stored in the ILM, in the same order. We will consider the ILM as a matrix, the rows being the memory locations and the columns being the positions on each location. The first K/N interleaved values are placed on the first column. The second set of K/N values is stored on the second column and the procedure continues. In order to perform the described method, a true dual port BRAM is selected. In **Figure 12**, each time a new value is added on row WA at column WP (near the already existing content at columns till WP-1), the content of row WA + 1 is also read from the memory. In the next clock period, a new value is added at row WA + 1 at column WP (near the already existing content at columns till WP − 1), while reading also the content of row WA + 2. And so on. When the interleaver function is used, the ILM is read in a normal way and the N interleaved values from a row are employed as reading addresses for the $V_2(X_k)$ memory. Furthermore, the new LTE interleaver module (with the QPP algebraic properties) will always place at the same row the N values that should be read in the interleaved order from ILM. The only additional task is a reordering process needed to match the corresponding RSCs. An example is presented in **Figure 13** for the values K = 40 and N = 8. On the left side, the content of the $V_2(X_k)$ memory is shown. Each column is composed of the outputs generated by one of the N RSC SISOs. On the right side, the content of ILM memory is described. Each minimum value from a line of the ILM represents the line address for the $V_2(X_k)$ memory (see the gray color circle in the illustration). By using a reordering module, each position from the outputted line is directed to its corresponding SISO. For example, position c from the first read line (index 10) is sent to SISO g, whereas position c from the second read line (index 13) is sent to SISO a. The same procedure applies also for the deinterleaving process, only that the write addresses are extracted from ILM, while the reading ones are used in the natural order.

For the reordering module, an even-odd merge sorting network is applied. The corresponding method was introduced by Batcher in Ref. [14] and is part of the sorting network group that includes several sorting approaches. One such example is the bubble sorting, which sorts in a repeated manner the adjacent pairs of elements. Another example is the shell sorting, which groups the input data into an array and then performs the array's column sorting (also in a repeating manner). After each associated iteration, the array becomes one column smaller. A third example is the even-odd transposition sorting, which sorts alternatively the odd-indexed

http://dx.doi.org/10.5772/67017

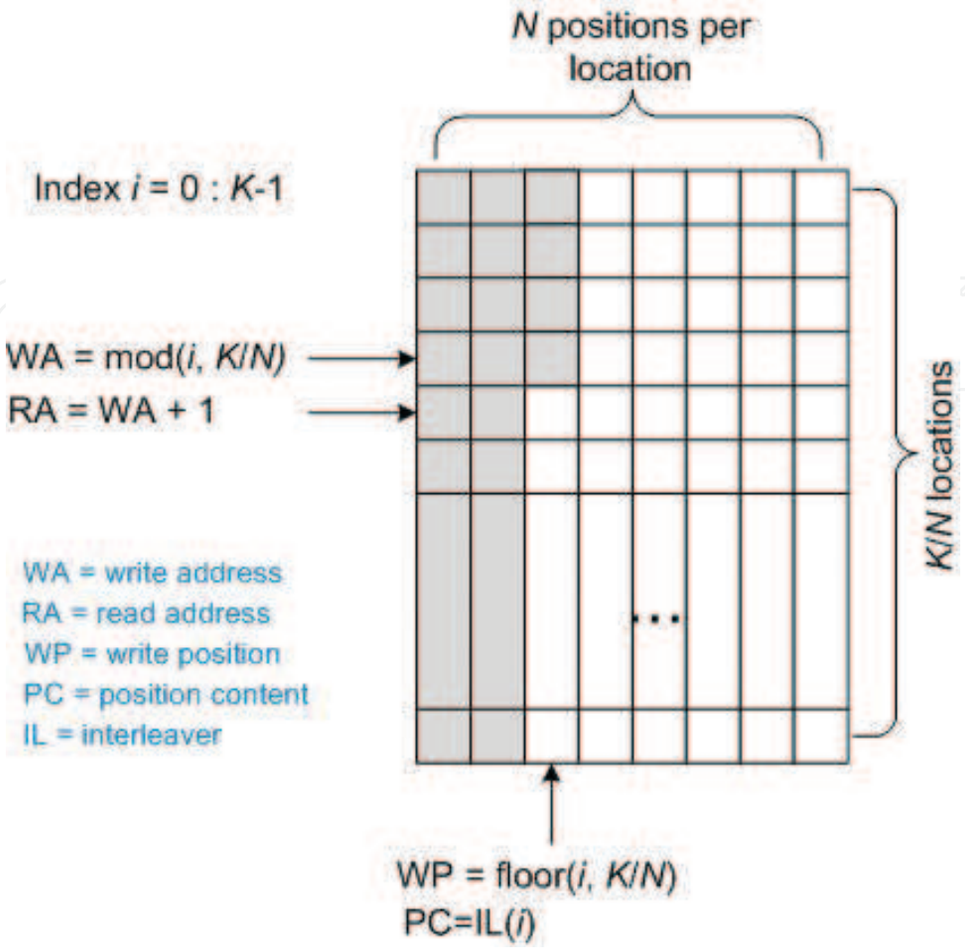

Figure 12. ILM memory writing procedure.

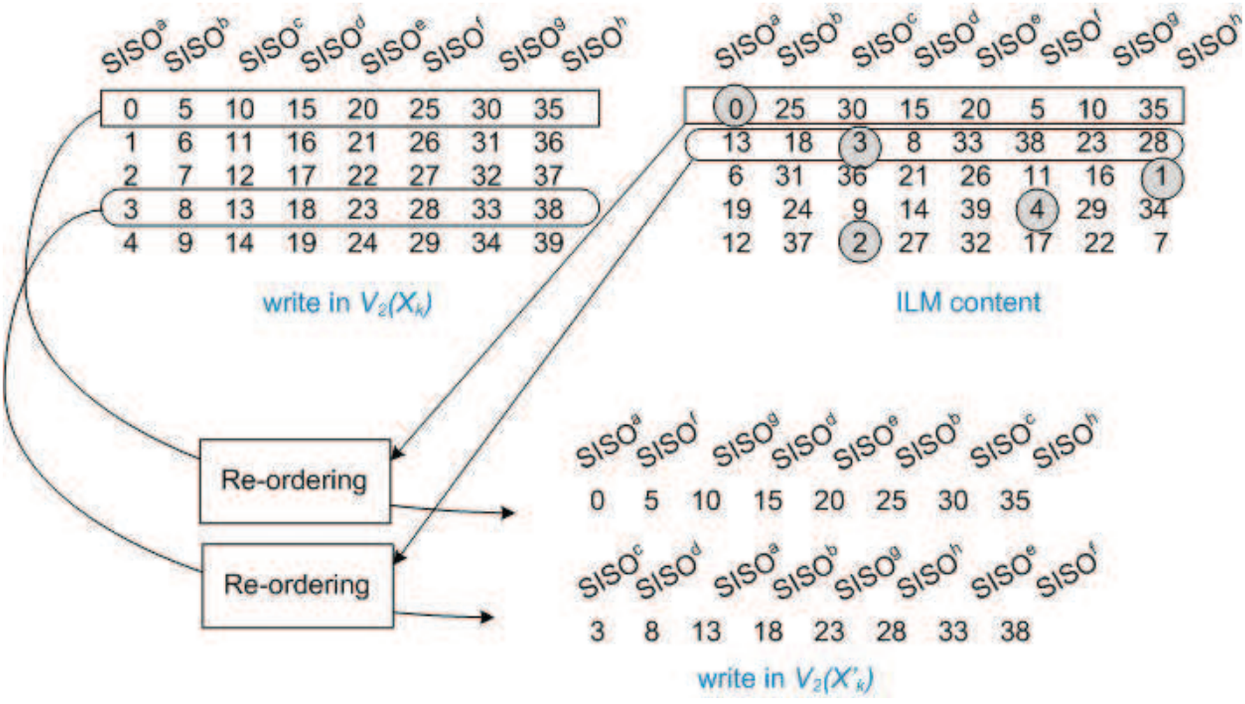

Figure 13. Virtual parallel interleaver.

and the adjacent even-indexed elements, respectively, the even-indexed elements and the adjacent odd-indexed values. The fourth example is the bitonic sorting. The two halves of the input data are sorted in opposite directions and then jointly processed to produce one complete sorted sequence.

The even-odd merge sorting method is based on a theorem saying that any list of $a = 4b$ (b natural) elements can be sorted if the following steps are applied: first, separate sorting is executed over the two halves of the list. After this step, the elements with odd index and the ones with even index are sorted separately. The last step consists in a comparing and switching procedure executed over all the elements $2n$ and $2n + 1$ ($n = 1,\ldots, a/2 - 1$). The demonstration of this theorem is available in Ref. [23]. An example for $N = 8$ is depicted in a graphical format shown in **Figure 14**. From a timing point of view, **Figure 15** depicts the case when $N = 2$ is used. Same comments as the ones for **Figure 11** apply.

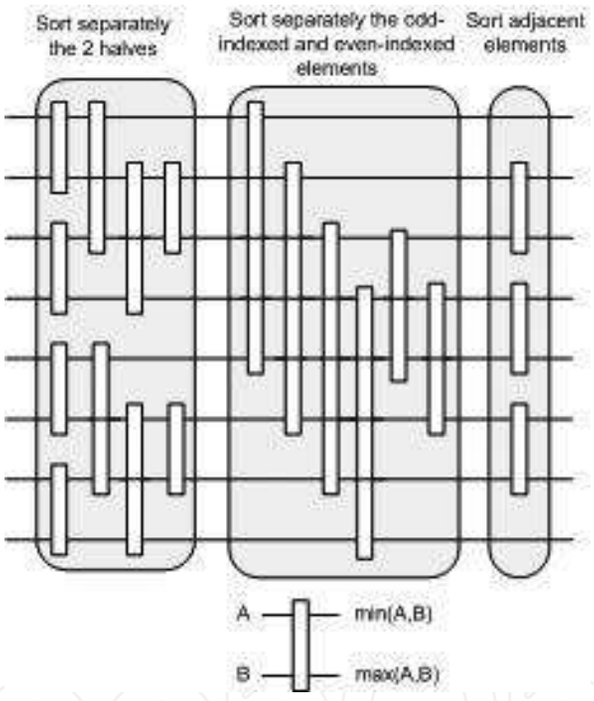

Figure 14. Even-odd merge sorting for $N = 8$.

In combination with the presented parallel decoding architecture, we also propose a simplified implementation for the interleaver block. As seen from Eq. (3), the arithmetic requirements for the computation of the memory addresses $\pi(i)$ consist of three multipliers, one adder and one divider (used for the extraction of the remainder associated with the *modulo* operation). For all possible K values associated with the division, the quotients range is very large, since the numerator and the denominator can have very big values (and often situated in different numerical ranges—up to billions). We propose an efficient method to reduce the arithmetic complexity associated with Eq. (3).

By introducing the notation

http://dx.doi.org/10.5772/67017

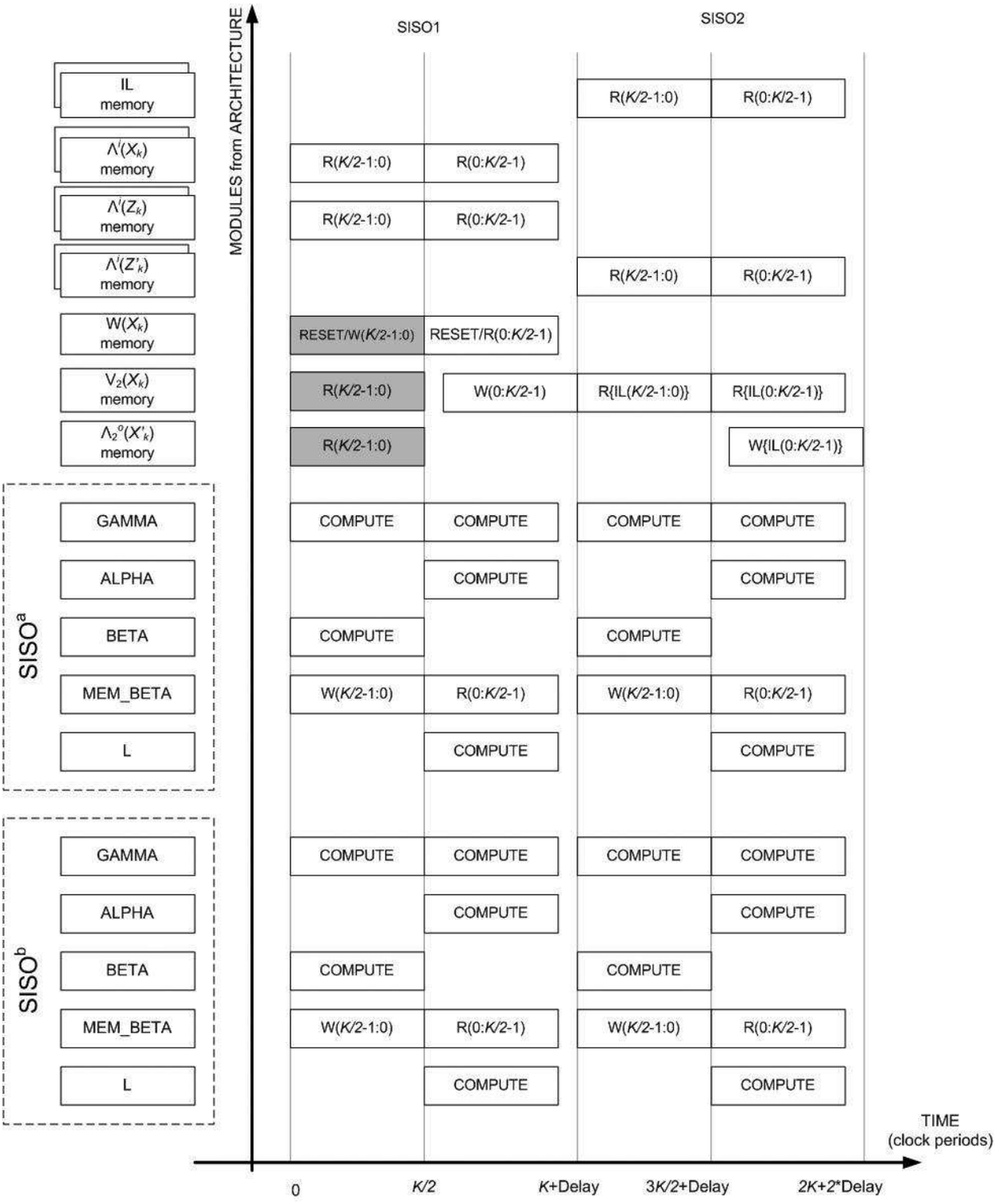

Figure 15. Time diagram for parallel turbo decoder ($N = 2$).

$$p(i) = f_1 i + f_2 i^2 \tag{23}$$

it can be observed that

$$\begin{aligned} &p(0) = 0, \\ &p(i) = p(i\text{–}1) \ + \ s_1 \ + \ s_2(i), \quad i > 0, \end{aligned} \tag{24}$$

where

$$\begin{aligned} &s_1 = f_1 \quad \text{and} \\ &s_2(i) = \begin{cases} 0, & i = 0, \\ f_2, & i = 1, \\ s_2(i\text{–}1) \ + \ 2f_2, & i > 1 \end{cases} \end{aligned} \tag{25}$$

We can rewrite Eq. (3) using Eqs. (23) and (24)

$$\pi(i) = p(i) \ \bmod \ K = [p(i\text{–}1) \ + \ s_1 \ + \ s_2(i)] \ \bmod \ K \tag{26}$$

The multiplications are replaced by additions, which require less hardware resources. Nevertheless, the division is still necessary for the *modulo* operation. If we consider the *modulo* operator applied to a sum of elements expressed as

$$\left[\sum_k c_k\right] \ \bmod \ K = \ \left[\sum_k c_k \ \bmod \ K\right] \ \bmod \ K \tag{27}$$

we can decrease the arithmetic effort needed to obtain $\pi(i)$ in Eq. (26). The number of *modulo* operations becomes bigger, but the overall complexity of the corresponding divisions is reduced since smaller quotients are used. Consequently, using Eqs. (25)–(27), one can write:

$$s_3(i) = s_1 + s_2(i) = \begin{cases} 0, & i = 0, \\ f_1 + f_2, & i = 1, \\ s_3(i\text{–}1) \ + 2f_2, & i > 1 \end{cases} \tag{28}$$

Using Eq. (29) in Eq. (26), the result is

$$\begin{aligned} \pi(i) = p(i) \ \bmod \ K &= [p(i\text{–}1) + s_3(i)] \ \bmod \ K \\ &= [p(i\text{–}1) + s_3(i\text{–}1) \ + 2f_2)] \ \bmod \ K \\ &= [\pi(i\text{–}1) + s_3(i\text{–}1) \bmod \ K + 2f_2 \bmod \ K] \ \bmod \ K \end{aligned} \tag{29}$$

All of the numerical values added in the last stage of Eq. (29) are lower than K and available recursively (during the processing of a distinct frame), such as $\pi(i\text{–}1)$ and $s_3(i\text{–}1) \bmod \ K$ or they can be predetermined and stored, like the case of $2f_2 \ \bmod \ K$.. The overall arithmetic complexity is reduced to $2K$ additions and $2K$ simplified modulo operations (i.e., each is resolvable using a comparison and a subtraction) for the address generation module. The method improves the solutions presented in [24, 25], by eliminating any multiplications or divisions. Additionally, the lower numerical range of the operators (with values lower than $2K$; i.e., values in the range of thousands) allows the usage of minimal resources for the representation of binary values.

http://dx.doi.org/10.5772/67017

6. Implementation results

6.1. WiMAX systems

The estimated system frequency when implementing the decoding structure on a Xilinx XC4VLX80-11FF1148 chip using the Xilinx ISE 11.1 tool is 125 MHz. The reserved chip area is around 3000 (8.37%) slices from a total of 35,840. The results are comparable with the assessments presented in [26].

The decoding latency and decoding rate corresponding to the above-mentioned clock frequency (see **Table 1**) are

$$Latency = 2L(2K + 10) \tag{30}$$

$$R_b = \frac{2K}{2L(2K + 10)T_{clk}} \tag{31}$$

		Latency [μs]			Rb [Mbps]		
Fclk [MHz]	***K* [di-bits]**	***L* = 3**	***L* = 4**	***L* = 5**	***L* = 3**	***L* = 4**	***L* = 5**
125	24	2.78	3.71	4.64	17.24	12.93	10.34
125	240	23.52	31.36	39.2	20.41	15.31	12.24
125	2400	230.9	307.8	384.8	20.79	15.59	12.47

Table 1. Latency and throughput.

The implementation delay is represented by 10 clock periods per iteration and is added to the theoretical latency of the MAP algorithm (which is $4KN$ clock periods).

In **Figure 16**, the decoding performances are presented for a quadrature phase shift keying (QPSK) modulation, ½ rate, 1–4 iterations, a block size of 6 bytes (the smallest possible) and a transmission simulated through an additive white Gaussian noise (AWGN) channel. The results are depicted for the worst case scenarios, considering that the test was performed for the smallest block size.

6.2. LTE systems

Figures 11 and **15** show that the decoding latency is reduced in the case of parallel decoding with a factor almost equal to N. The presented implementation has an 11 clock period *Delay,* which is added for each forward trellis run (when the LLRs are computed). As a consequence, two such values must be considered during each iteration.

For serial decoding, the native latency is computed as follows: at the first semi-iterations, K clock periods required for the backward trellis run and another (K + *Delay*) clock periods for the forward trellis run and LLR computation. The value is then considered twice in order to take into account the second semi-iteration. By denoting L the number of executed iterations,

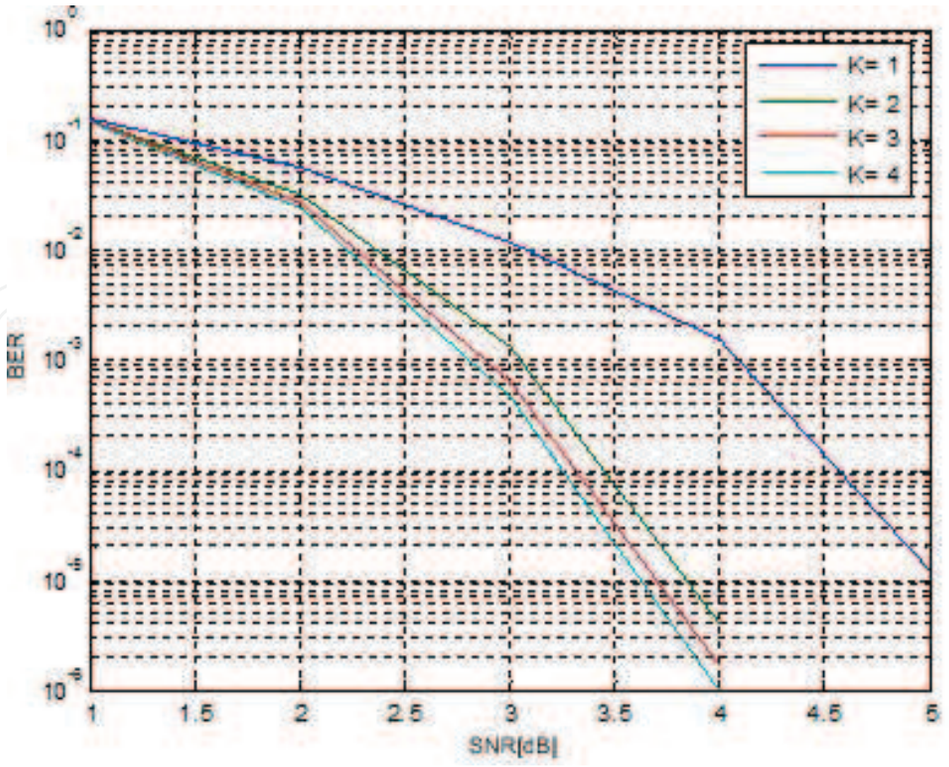

Figure 16. The impact of the number of iterations on decoding performances.

the numbers of clock periods required for a serial, respectively, a parallel block decoding operation result as:

$$Latency\_s = (4K + 2Delay)L \tag{32}$$

$$Latency\_p = (4K/N + 2Delay)L \tag{33}$$

When performing tests for the parallel decoding performances, a certain level of degradation was observed, since the forward and backward metrics are altered at the data block boundaries. In order to have similar performance as in the serial decoding case, a small overhead is accepted. By introducing an overlap at each parallel block boarder, the metrics computation gains a training phase. The minimum overlap window length is selected to cover the minimum standard defined data block (in this case $K_{\min} = 40$ bits).

Figure 17 shows this situation, for the $N = 2$ setup. If we consider $N > 2$, which leads to blocks with $K_{\min}$ at both the left and right sides, the corresponding latency can be expressed as:

$$Latency\_po = \left(4(K/N + 2K_{\min}) + 2Delay\right)L \tag{34}$$

For even-odd merge sorting network implementation, we can study the configuration $K = 40$ bits and $N = 8$. The input of the ILM content is represented by the 40 interleaved addresses organized in five memory locations and eight addresses for each location. The minimum-detected value for each ILM location (i.e., the natural-order memory location that will be

http://dx.doi.org/10.5772/67017

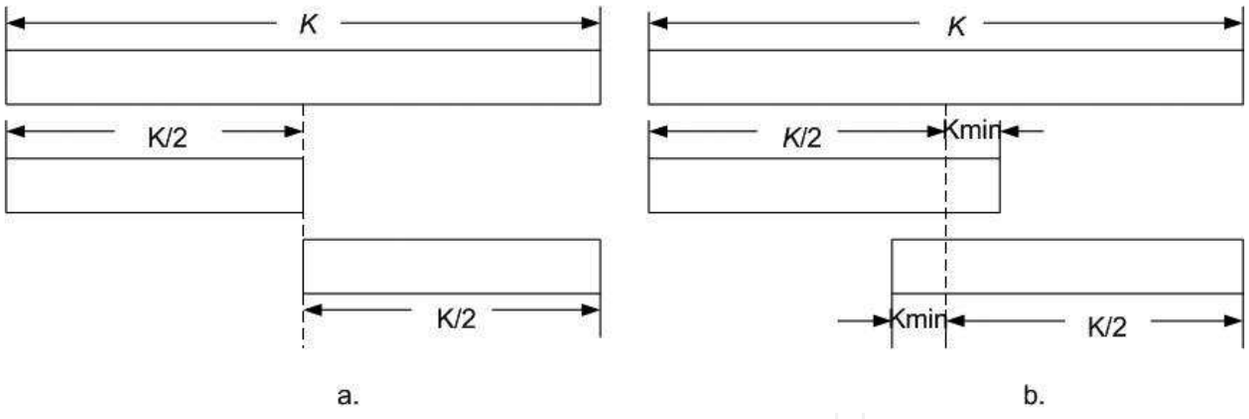

Figure 17. (a) Non overlapping split; (b) overlapping split.

accessed) is contained in the output of the sorting unit. Also, the module provides the order which will be used to send data read from natural-order memory location to the *N* decoding units. In this example, at the third clock period, the second ILM location is read, i.e., the addresses 6, 31, 36, 21, 26, 11, 16 and 1. The sorting module labels these addresses with an index, obtaining the pairs: (6, 0), (31, 1), (36, 2), (21, 3), (26, 4), (11, 5), (16, 6) and (1, 7). Then the addresses are arranges in an increasing order: (1, 7), (6, 0), (11, 5), (16, 6), (21, 3), (26, 4), (31, 1) and (36, 2). At the same time, the minimum address found at this location is sent at the output, 1 in this example. In conclusion, location number 1 is read from the natural-order data memory. The eight samples from the location 1 are distributed to the eight decoding units as indicated by the output index. The first sample from this location is sent to decoder unit 7, the second sample to decoder unit 0, the third one to decoder unit 5 and so on. As **Figure 18** shows that at the register transfer level (RTL), besides flip flops, the sorting unit includes only basic selection elements.

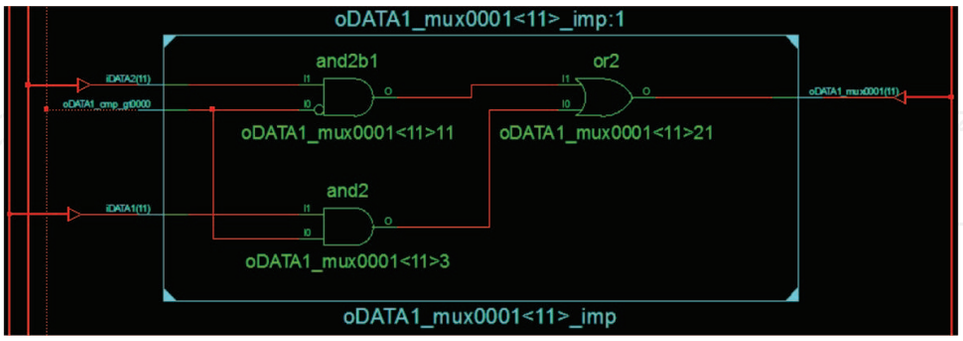

Figure 18. Basic selection element for binary inputs.

It can be seen in **Figure 19** that the sorting unit allows a pipeline data processing. Consequently, with a certain implementation delay (7 clock periods in the proposed scheme), the module provides a value belonging to the set of sorted indexes at each clock cycle.

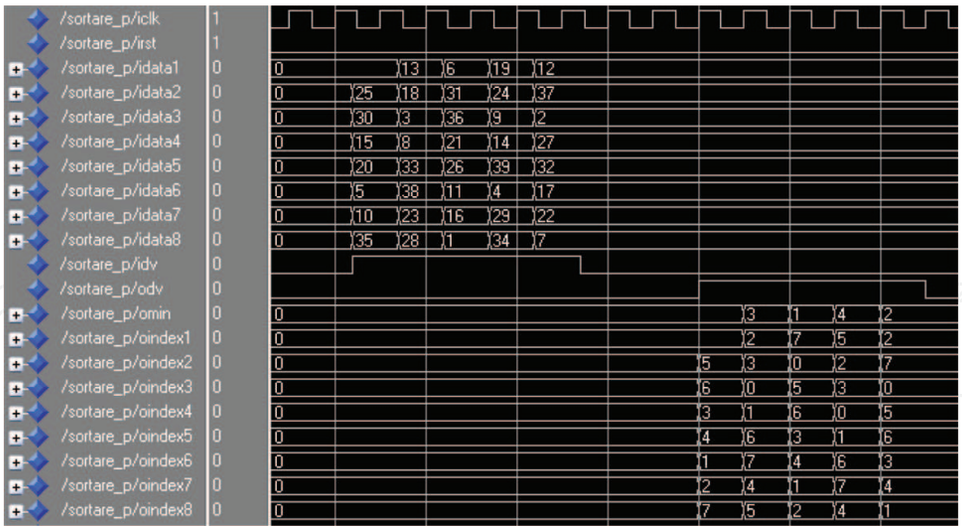

Figure 19. Even-odd merge sort – ModelSim simulation.

It is important to mention that the even-odd merge sorting was selected because it allows a pipeline functioning, consuming also lower resources than the other listed methods. Some comparative results were provided in [11, 27] in terms of used resources for the application-specific integrated circuit (ASIC).

In order to evaluate the performances, we used the very high speed hardware description language (VHDL), programming language. The code was tested using ModelSIM 6.5. For the generation of RAM/ROM memory blocks, Xilinx Core Generator 14.7 was employed and the synthesis process was accomplished using Xilinx XST from Xilinx ISE 14.7. Using the above-mentioned tools, the resulted values for the decoding structure when implemented on a Xilinx XC5VFX70T-FFG1136 are the following [28]: frequency of 310 MHz and 664 flip flops and 568 LUTs for the sorting unit, respectively, a frequency of 300 MHz, 1578 flip flop registers and 1708 LUTs for the interleaver.

The values listed in **Table 2** are obtained using Eqs. (32)–(34), when $N = 8$ is considered. One can observe that the overhead introduced by the overlapping split method is less important for bigger values of K, this being the scenario when a parallel approach is usually applied. The achieved overall system frequency is 210 MHz, with the longest signal propagation time required for the SISO unit.

Table 3 provides the corresponding throughput rate when the values from **Table 2** are used.

	Latency_s [μs]		*Latency_p* [μs]		*Latency_po* [μs]	
K	*L*					
	3	4	3	4	3	4
1536	88.08	117.4	11.28	15.04	15.85	21.14
4096	234.3	312.5	29.57	39.42	34.14	45.52
6144	351.4	468.5	44.2	58.9	48.7	56.02

Table 2. Latency values for $N = 8$, $L = 3$ or 4 and $K = 1536$, 4096 or 6144.

	Tput_s [Mbps]		*Tput_p* [Mbps]		*Tput_po* [Mbps]	
K	L					
	3	4	3	4	3	4
1536	17.43	13.07	136.1	102.0	96.86	72.64
4096	17.47	13.10	138.5	103.8	119.9	89.9
6144	17.48	13.11	139	104.2	125.9	94.4

Table 3. Throughput values for $N = 8$, $L = 3$ or 4 and $K = 1536$, 4096 or 6144.

As one can observe from **Table 3**, the serial decoding performance is similar to the theoretical one. Let us consider, for example, the case $L = 3$ and $K = 6144$. Considering the theoretical latency of $4KL$ clock periods, the theoretical throughput is 17.5 Mbps. After implementation, the obtained result for the proposed serial architecture is 17.48 Mbps.

The following performance graphs were obtained using a finite precision Matlab simulator. This approach was selected because the same outputs as the ModelSIM simulator are obtained in Matlab, while the testing time is considerably smaller.

All the simulation results were generated for the Max Log MAP algorithm. The illustrations present 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 20 presents the attained performances for the case of $K = 512$, $N = 2$, $L = 3$ and QPSK modulation, using the three discussed decoding methods, i.e., the serial one, the parallel without overlapped split one and the parallel with overlapped split one. **Figure 21** depicts the same performance comparison, this time for $K = 1024$ and $N = 4$.

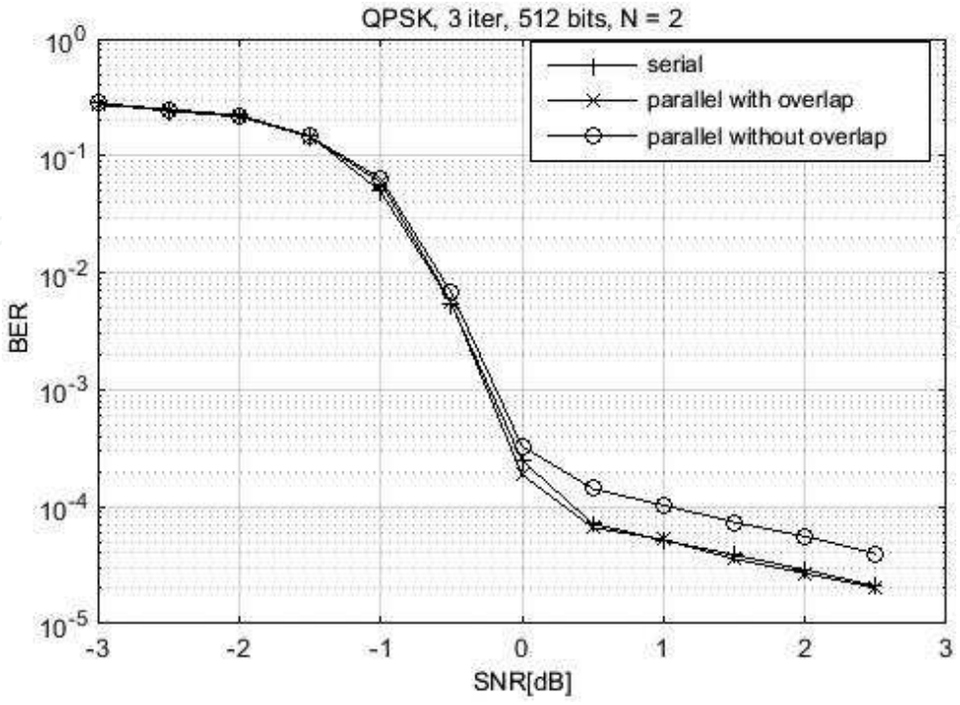

Figure 20. Comparative decoding results for QPSK, $L = 3$, $K = 512$, $N = 2$.

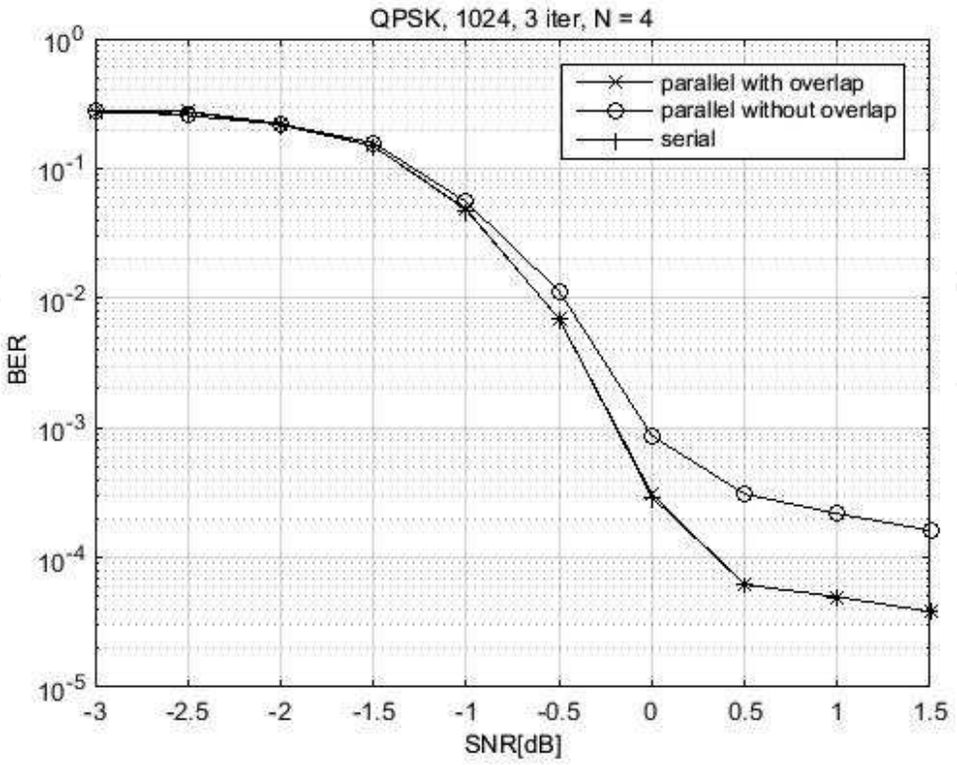

Figure 21. Comparative decoding results for QPSK, $L = 3$, $K = 1024$, $N = 4$.

Analyzing the results presented in **Figures 20** and **21**, one can conclude that the decoding performance obtained, when parallel decoding with the overlapped split method is used, is almost similar to the one for serial decoding. In contrast, the parallel decoding without the overlapped split method generates some loss in performance when compared to the serial decoding. This degradation is dependent on the parallelization factor N.

7. Conclusions

This chapter presented the most important aspects related to the FPGA implementation of a turbo decoder for WiMAX and LTE systems. The serial turbo decoder architectures for the two systems have been developed and efficiently implemented, important results being obtained especially for the proposed architectures of the interleaver/deinterleaver. For LTE systems, the interleaver memory ILM has been introduced. In this manner, the interleaver process effectively works only outside the decoding process itself.

The ILM has been written together with the input data, while the previous block was still under decoding. It should be outlined that this solution allows the transition from the serial to the parallel decoder in an efficient manner, involving only values that are concatenated at same memory locations. The parallel approach requires the same storing capacity (the number of BRAMs) and a single interleaver, thus adding only an even-odd merge sorting network. This unique interleaver has been implemented in an efficient configuration that uses only comparators and subtractors and no multipliers and dividers

The parallel decoding performances have been compared with the serial ones. In this context, certain degradation has been observed. In order to eliminate this degradation, a small overhead is accepted by the overlapping split that is applied to the parallel data blocks.

Acknowledgements

This work was supported by the UEFISCDI under Grant PN-II-RU-TE-2014-4-1880.

Author details

Cristian Anghel*, Cristian Stanciu and Constantin Paleologu

*Address all correspondence to: canghel@comm.pub.ro

Politehnica University of Bucharest, Romania

References

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

[2] C. Berrou and A. Glavieux, Near optimum error correcting coding and decoding: Turbo-Codes, *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, 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] 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. 203–215, Oct. 2001.

[5] 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.

[6] C. Anghel, A. A. Enescu, et al., FPGA implementation of a CTC Decoder for H-ARQ compliant WiMAX systems, *Proceedings of International Conference on Design & Technology of Integrated Systems*, DTIS 2007, Morocco, pp. 82–86.

[7] C. Anghel, V. Stanciu, C. Stanciu and C. Paleologu, CTC Turbo decoding architecture for LTE systems implemented on FPGA, *IARIA ICN 2012*, Reunion, France, 2012.

[8] S. Chae, A low complexity parallel architecture of turbo decoder based on QPP interleaver for 3GPP-LTE/LTE-A, http://www.design-reuse.com /articles/31907/turbo-decoder-architecture-qpp-interleaver-3gpp-lte-lte-a.html

[9] Y. Sun and J. R. Cavallaro, Efficient hardware implementation of a highly-parallel 3GPP LTE/ LTE-advance turbo decoder, *Integration, the VLSI Journal*, vol. 44, no. 4, pp. 305–315, Sept. 2011.

[10] D. Wu, R. Asghar, Y. Huang and D. Liu, Implementation of a high-speed parallel turbo decoder for 3GPP LTE terminals, ASICON '09, *IEEE 8th International Conference on ASIC*, pp. 481–484, 2009.

[11] C. Studer, C. Benkeser, S. Belfanti and Q. Huang, Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE, *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 8–17, Jan. 2011.

[12] C. Anghel and C. Paleologu, Simplified parallel architecture for LTE-A turbo decoder implemented on FPGA, *Proceedings of the 9th International conference on Circuit, Systems, Signal and Telecommunications CCST 2015*, Dubai, pp. 102–111.

[13] C. Stanciu, C. Anghel and C. Paleologu, Efficient recursive implementation of a quadratic permutation polynomial interleaver for LTE systems, *Revue Roumaine, des Sciences Techniques - Serie Électrotechnique et Énergétique*, ISSN: 0035-4066, vol. 61, pp. 53–57.

[14] K. E. Batcher, Sorting networks and their applications," in *Proceeding of AFIPS Spring Joint Computer Conference*, vol. 32, 1968.

[15] C. Anghel, C. Stanciu and C. Paleologu, Sorting methods used in parallel turbo decoding for LTE systems, *2015 International Symposium on Signals, Circuits and Systems (ISSCS)*, 9–10 July, 4 p.

[16] Xilinx Virtex 5 family user guide, https://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[17] Xilinx ML507 evaluation platform user guide, https://www.xilinx.com/products/boards/ml507/docs.htm

[18] https://standards.ieee.org/about/get/802/802.16.html

[19] 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)."

[20] P. Robertson, E. Villebrun and P. Hoeher, A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain, *Proceeding of IEEE International Conference on Communications* (ICC'95), Seattle, pp. 1009–1013, June 1995.

[21] S. Papaharalabos, P. Sweeney and B. G. Evans, Constant log-MAP decoding algorithm for duo-binary turbo codes, *Electronics Letters*, vol. 42, no. 12, pp. 709–710, June 2006.

[22] 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 vol. 3, pp. 2252–2256, 2000.

[23] Massachusetts Institute of Technology, Mathematics, last access date: November 2014, *math.mit.edu/~shor/18.310/batcher.pdf*

[24] R. Asghar, D. Wu, J. Eilert and D. Liu, Memory conflict analysis and a re-configurable interleaver architecture supporting unified parallel turbo decoding, *Journal of Signal Processing Systems*, vol. 60, no. 1, pp. 15–19, July 2010.

[25] S. Wang, L. Liu and Z. Wen, High speed QPP generator with optimized parallel architecture for 4G LTE-A system, *International Journal of Advancements in Computing Technology*, vol. 4, no. 23, pp. 355–364, July 2010.

[26] Xilinx, IEEE 802.16e CTC decoder core, DS137 (v2.3), July 11, 2006.

[27] E. Mumolo, G. Capello and M. Nolich, VHDL design of a scalable VLSI sorting device based on pipelined computation," *Journal of Computing and Information Technology - CIT 12*, vol. 12, no. 1, pp. 1–14, 2004.

[28] C. Anghel, C. Stanciu and C. Paleologu, LTE turbo decoding parallel architecture with single interleaver implemented on FPGA, Springer Verlag *Circuits, Systems and Signal Processing*, ISSN: 0278-081X, DOI 10.1007/s00034-016-0362-z, 2016.

IntechOpen