Abstract: - This paper describes a turbo decoder for 3GPP Long Term Evolution Advanced (LTE-A) standard, using a Max LOG MAP algorithm, implemented on Field Programmable Gate Array (FPGA). Taking advantage of the quadratic permutation polynomial (QPP) interleaver proprieties and considering some FPGA block memory characteristics, a simplified parallel decoding architecture is proposed. It should be used especially for large data blocks, when high decoding latency is introduced by the serial decoding. The parallelization factor $N$ is usually a power of 2, the maximum considered value being 8. The obtained parallel decoding latency is $N$ times lower than the serial decoding latency. With the cost of very low latency added to this value, the parallel decoding performances are similar with the serial decoding ones. The novelty of the proposed parallel architecture is that only one interleaver is used, independently of the $N$ value.

Key-Words: - LTE-A, turbo decoder, Max LOG MAP, parallel architecture, FPGA

1 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 existence, 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. Nowadays 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.

An UMTS dedicated turbo decoding scheme is presented in [7]. Due to the new LTE/ LTE-A 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. The parallel decoding represents one software adaptation requested by the high data rates, while additional hardware changes are also proposed [8].

There are many parallel decoding architectures proposed in the literature in the last years. The obtained results are evaluated on 2 axes. The first one is the decoding performances degradation
introduced by the parallel method compared with
the serial decoding scheme and the second one is the
amount of resources needed for such parallel
architecture implementation. A first set of parallel
architectures is described in [9]. Starting from the
classical method of implementing the Maximum A
Posteriori (MAP) algorithm, i.e., going to trellis
once to compute the Forward State Metrics (FSM)
and then twice to compute the Backward State
Metrics (BSM) and also the Log Likelihood Ratios
(LLR), several solutions to reduce the decoding
latency of 2K clock periods per semi-iteration,
where K is the data block length, are introduced.
The first one reduces the decoding time to half (only
K) by starting simultaneously the BSM and FSM
computation. After computing half of these values,
2 LLR blocks start working in parallel, the
interleaver block being also doubled. Another
proposed scheme eliminates the need for the second
interleaver but increases the decoding time with K/2
compared with the previous one, a total decoding
latency of 3K/2 clock periods being obtained.

A second set of parallel architectures takes
advantage of the Quadratic Permutation Polynomial
(QPP) interleaver algebraic-geometric properties, as
described in [10][11]. Here efficient hardware
implementations of the QPP interleaver are
proposed, but the parallelization factor N represents
also the number of used interleavers in the proposed
architectures.

A third approach consist in using a folded
memory to store simultaneously all the values
needed for parallel processing [12]. But for this kind
of implementation the main challenge is to correctly
distribute the data to each decoding unit once a
memory location containing all N values was read.
More precisely, the N decoding units working in
parallel were writing their data in a concatenated
order to the same location, but when the interleaved
reading is taking place, these values are not going in
the same order to the same decoding unit, but
instead they should be redistributed. To solve this,
an architecture based on 2 Batcher sorting networks
is proposed. But also in this approach, N interleavers
are needed to generate all the interleaved addresses
that input the master network.

In this paper, we introduce also a folded memory
based approach, but the main difference comparing
with the already existent solutions described above
is that our proposed solution uses only one
interleaver. Additionally, with some multiplexing
and demultiplexing blocks, the parallel architecture
remains close to the serial one, only the Soft Input
Soft Output (SISO) decoding unit being instantiated
N times. The block memories numbers and
dimensions are unchanged between the two block
schemes. In terms of decoding performances, with
the cost of a small overhead added, the performances of the serial and parallel decoding
architectures are kept similar.

The paper is organized as follows. Section 2
describes the LTE coding scheme with the new
introduced QPP interleaver. Section 3 presents the
decoding algorithm. In Section 4, there are
discussed the implementation solutions and the
proposed decoding schemes, for both serial and
parallel decoding. Section 5 presents throughput and
speed results obtained when targeting a
XC5VFX70T [13] chip on Xilinx ML507 [14]
board; it also provides simulation curves comparing
the results obtained when using serial decoding,
parallel decoding, and parallel decoding with
overlap. Section 6 contains the conclusions of this
work.

2 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_i. The K bits corresponding to this
block are sent as systematic bits at the output in the
stream X_i. In the same time, the data block is
processed by the first constituent encoder resulting
parity bits Z_e, while the interleaved data block C_i
is processed by the second constituent encoder
resulting parity bits Z_i. Combining the systematic
bits and the two streams of parity bits, the following
sequence is obtained at the output of the encoder:
X_i, Z_i, Z'_i, X_k, Z_k, Z'_k, ..., 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+4},
Z'_{k+2}, X'_{k+3}, Z'_{k+3}.

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:
where the interliving function $\pi$ applied over the output index $i$ is defined as

$$\pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \mod K.$$  

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

### 3 Decoding Algorithm

The LTE turbo decoding scheme is depicted in Fig. 2. The two Recursive Systematic Convolutional (RSC) decoders use in theory the MAP algorithm. This classic algorithm provides the best decoding performance, 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) [15], Maximum Log MAP (Max Log MAP), Constant Log MAP (Const Log MAP) [16], and Linear Log MAP (Lin Log MAP) [17].

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.,

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

The LTE turbo decoder trellis diagram contains 8 states, as depicted in Fig. 3. Each diagram state permits 2 inputs and 2 outputs. The branch metric between the states $S_i$ and $S_j$ is

$$\gamma_0 = V(X_k)X(i,j) + \Lambda'(Z_k)Z(i,j).$$  

where $X(i,j)$ represents the data bit and $Z(i,j)$ is the parity bit, both associated to one branch. Also $\Lambda'(Z_k)$ is the LLR for the input parity bit. When SISO 1 decoder is taken into discussion this input LLR is $\Lambda'(Z_k)$, while for SISO 2 it becomes $\Lambda'(Z_k)$; $V(X_k)=V_1(X_k)$ represents the sum between $\Lambda'(X_k)$ and $W(X_k)$ for SISO 1 and $V(X_k)=V_2(X_k)$ represents the interleaved version of the difference between $\Lambda'(X_k)$ and $W(X_k)$ for SISO 2. In Fig. 2, $W(X_k)$ is the extrinsic information, while $\Lambda'_1(X_k)$ and $\Lambda'_2(X_k)$ 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:

$$\gamma_0 = 0$$

$$\gamma_1 = V(X_k)$$

$$\gamma_2 = \Lambda'(Z_k)$$

$$\gamma_2 = V(X_k) + \Lambda'(Z_k).$$

The decoding process is based on going forward and backward through the trellis.
3.1 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 \( 2 \leq k \leq K+3 \) and \( 0 \leq i \leq 7 \). 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

\[
\hat{\beta}_k(S_i) = \max\{\beta_{k+1}(S_{i1}) + \gamma_{i1}, \beta_{k+1}(S_{i2}) + \gamma_{i2}\},
\]

where \( \hat{\beta}_k(S_i) \) represents the un-normalized metric and \( S_{i1} \) and \( S_{i2} \) 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

\[
\beta_k(S_i) = \hat{\beta}_k(S_i) - \hat{\beta}_k(S_0)
\]

and then stored in the dedicated memory.

3.2 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. In order to allow the computation of the current stage \( (k) \) metrics, only the forward metrics from the last stage \( (k-1) \) have to be stored. The forward metric for the state \( S_i \) at the stage \( k \) is \( \alpha_k(S_i) \) with \( 0 \leq k \leq K-1 \) and \( 0 \leq i \leq 7 \). 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

\[
\hat{\alpha}_k(S_j) = \max\{\alpha_{k-1}(S_i) + \gamma_{ij}, \alpha_{k-1}(S_{i1}) + \gamma_{i1}, \alpha_{k-1}(S_{i2}) + \gamma_{i2}\},
\]

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

\[
\alpha_k(S_i) = \hat{\alpha}_k(S_i) - \hat{\alpha}_k(S_0).
\]

Because the forward metrics \( \alpha \) 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 stage \( k-1 \) and the state \( S_j \) at stage \( k \) is

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

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:

\[
\Lambda^* (X_k) = \max_{(S_i \rightarrow S_j)} \{\lambda_k(i, j)\} - \max_{(S_i \rightarrow S_j)} \{\lambda_k(i, j)\},
\]

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

4 Proposed Decoding Scheme

4.1 Serial Decoder Block Scheme

From the theoretical decoding scheme depicted in Fig. 2 it can be noticed that SISO 2 decoder starts working only after SISO 1 decoder finishes its job and vice-versa, the usage of previously obtained extrinsic values being the main principle of the turbo decoding.
Also, all the processing is based on complete data blocks since the interleaver or deinterleaver procedures should be applied in between. It results that the 2 SISOs are decoding data in non-overlapped time windows, so only one SISO unit can be used to process in a time-multiplexed manner, as one can observe in Fig. 4, where a serial decoder block scheme based on the previous work presented in [18] for a WiMAX CTC decoder is described.

The memory blocks are used for storing data from one semi-iteration to another and from one iteration to another. The dotted-line memory blocks are virtual memories added only to ease the understanding of the introduced notations. Also, it should be mentioned that the Interleaver and Deinterleaver blocks are in fact the same, including a block memory called ILM (Interleaver Memory) and an interleaver. The ILM is the new approach introduced by the author compared with the previous serial implementation presented in [19] and the goal is to prepare the architecture for parallel decoding also. The memory is written with the interleaved addresses each time a new data block is received. The values are then used as read addresses (when interleaver process is ongoing) or as write addresses (when deinterleaver process is ongoing). This ILM, together with the 3 memories from the left side of the picture (for the input data) are switched-buffers, allowing new data to be written while the previous one is still under decoding process.

The scheme depicted in Fig. 4 works as follows. SISO 1 reads the memory locations corresponding to V_i(X_i) and Δ_i(Z_i) 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_i') and Δ_i(Z_i') vectors in order to perform the second semi-iteration.

The vector V_i(X_i) is obtained by adding the input vector Δ_i'(X_i') with the extrinsic information vector W(X_i). While reading these 2 memories, SISO 1 starts the decoding process. At the output, the LLRs are available and performing the subtraction between them and the delayed extrinsic values already read from W(X_i) memory, the vector V_2(X_i) is computed and then stored into its corresponding memory in a normal order. The interleaving process is started (the initially written ILM is read now in normal order, so that interleaved read address for V_2(X_i) are obtained) and the re-ordered LLRs V_2(X_i') are available, the corresponding values for the 3 tail bits X_k+i, X_k+i+2, X_k+i+3 being added at the end of this sequence. The second semi-iteration is ongoing. The same SISO unit is used, but reading this time data inputs from the other memory blocks. As one can see from Fig. 4, two switching mechanisms are included in the scheme. When in position 1, the memory blocks for V_i(X_i) and Δ_i'(Z_i) are used, while in position 2 the memory blocks for V_2(X_i') and Δ_i'(Z_i') 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 Δ_i'(X_i') memory (the ILM output, which was already available for the V_2(X_i) interleaver process, is used as writing address for Δ_i'(X_i') memory, after a delay is added).

Reading in a normal order Δ_i'(X_i') memory and also V_2(X_i) memory provides inputs for W(X_i) memory and on the same time allows a new semi-iterations to start for SISO 1. So the W(X_i) memory update is made on the same time with a new semi-iteration start. Fig. 5 depicts a time diagram for the serial turbo decoding and the gray colored intervals.
describe $W(X_k)$ memory writing. One can observe that the upper 4 memories in the picture are switched-buffers, so they are written while the previous data block is still processed. In the picture $R$ stands for Read, $W$ represents Write, $(K-1:0)$ is the backward trellis run, $(0:K-1)$ is the forward trellis run and $IL$ means interleaved read (for interleaver process) or write (for deinterleaver process).

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).

The constituent modules of the SISO block are the ones presented in Fig. 6. 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.

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.

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

$$\pi(i) = \lfloor (f_1 + f_2 \cdot i) \mod K \rfloor \cdot i \mod K \ (12)$$

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.

The interleaved index computation is performed in three steps. First the value for $(f_1 + f_2 \cdot i) \mod K$ 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.

4.2 Parallel Decoder Block Scheme

The proposed parallel architecture is similar to the serial one described in Fig. 4, only that the RSC SISO module is instantiated \(N\) times in the scheme. We propose an architecture that concatenates the \(N\) values from the \(N\) RSCs and points always at the same memory location, for all the memories in the scheme. So instead of having \(K\) locations with 10 bits per location as in the serial architecture, in the parallel one each memory contains \(K/N\) locations with 10\(N\) bits per locations.

The main advantage introduced by the proposed serial architecture is the fact that the interleaver block works only once, before the decoding itself taking place. The ILM memory is written when a new data block is received, while the previous one is still under decoding. This approach allows a simplified parallel scheme way of work. Knowing the parallelization factor \(N\), the ILM memory can be prepared for the parallel processing that follows. More precisely, the ILM memory will have \(K/N\) locations, \(N\) values being written at each location.

As mentioned in [20], a Virtex 5 block memory can be configured from (32k locations x 1 bit) to (512 locations x 72 bits). In the worst case scenario when \(K=6144\), based on the \(N\) values and keeping the stored values on 10 bits as previously mentioned, the parallel ILM memory can be (768 locations x 80 bits), (1536 locations x 40 bits), (3072 locations x 20 bits), or (6144 locations x 10 bits), so still only 2 BRAMs are used, as in the case of serial ILM.

Fig. 7 describes the way ILM works. As one can observe, while writing procedure, each index \(i\) from 0 to \(K-1\) generates a corresponding interleaved values. These interleaved values are written in a normal order in ILM. The first \(K/N\) corresponding interleaved values occupy the first position on each memory locations. The second \(K/N\) values are placed on the second position of each location, and so on. In order to perform this procedure, a true dual port BRAM is used. Each time a new position in location \(n\) is written, the content of location \(n+1\) is also read from the memory, so that the next clock period the next interleaved value to be added to the already existing content at that location. When the interleaver function is needed during a semi-iteration, the ILM is read in a normal way, so that the \(N\) interleaved values from one location to represent the reading addresses for \(V_2(X_k)\) memory. But the QPP proprieties guarantee that the \(N\) values that should be read in the interleaved way from the memory are placed at the same memory location, only that their positions should be re-arranged before being sent to the corresponding RSCs. For simplifying the representation, the case of \(K=40\) and \(N=8\) is exemplified in Fig. 8. On the left one can see the content of \(V_2(X_k)\) memory. Each column represents the outupts of one of the \(N\) RSC SISOs. On the right there is described the content of ILM memory. The minimum values from each line of ILM (grey colored circle in figure) represents the line address for \(V_2(X_k)\) memory. Then, using a re-ordering module implemented with multiplexers and de-multiplexers, each position from the read line is sent to its corresponding SISO. For example, position \(b\) from the first read line (index 5) is sent to SISO \(f\), while position \(b\) from the second read line (index 8) is sent to SISO \(d\). The same procedure applies also for deinterleaver process, only that the
write addresses are extracted from ILM, while the read ones are in normal order.

From timing point of view, Fig. 9 depicts the case when $N=2$ is used. Same comments as the ones for Fig. 5 apply.

5 Implementation Results

From Fig. 5 and 9 it can be observed that the decoding latency is reduced in the case of parallel decoding with almost a factor equal to $N$. There is a certain $\text{Delay}$, which in this implementation case is 11 clock periods that adds at each forward trellis run, when the LLRs are computed, so 2 such values are introduced at each iteration.

The native latency for serial decoding is computed as follows: $K$ clock periods needed for the backward trellis run at the first semi-iterations, another $K$ clock periods plus $\text{Delay}$ for the forward trellis run and LLR computation, and multiplied by 2 for the second semi-iteration. Considering $L$ the number of executed iterations, it results a total latency in clock periods for each block serial decoding of:

$$\text{Latency\_s} = (4K + 2\text{Delay})L,$$  \hspace{1cm} (13)

while for the parallel decoding the needed number of clock periods is:

$$\text{Latency\_p} = (4K / N + 2\text{Delay})L.$$  \hspace{1cm} (14)

Testing the parallel decoding performances, a certain level of degradation was observed, since the forward and backward metrics are altered at the data block extremities. In order to obtain similar results as in the serial decoding case, a small overhead is accepted. If at each parallel block border an overlap is added, the metrics computation will have a training phase. The minimum overlap window may be as long as the minimum standard defined data block, in this case $K_{\min}=40$ bits. Fig. 10 describes this situation, for $N=2$ case.

The corresponding latency is in this case, considering $N>2$, which leads to blocks with $K_{\min}$ at both left and right sides:

$$\text{Latency\_po} = (4(K / N + 2K_{\min}) + 2\text{Delay})L.$$  \hspace{1cm} (15)

In order to evaluate the 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 values included in Table 1 are computed based on (13), (14), and (15) for the $N=8$ case. It can be noticed that the overhead introduced by the overlapped split is less significant once the value of $K$ increases, which represents the scenario when parallel decoding is usually used.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$L$</th>
<th>$\text{Latency_s\ [\mu s]}$</th>
<th>$\text{Latency_p\ [\mu s]}$</th>
<th>$\text{Latency_po\ [\mu s]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1536</td>
<td>3</td>
<td>88.08</td>
<td>117.4</td>
<td>11.28</td>
</tr>
<tr>
<td>4096</td>
<td>3</td>
<td>234.3</td>
<td>312.5</td>
<td>29.57</td>
</tr>
<tr>
<td>6144</td>
<td>4</td>
<td>351.4</td>
<td>468.5</td>
<td>44.2</td>
</tr>
</tbody>
</table>
Table 2 provides the corresponding throughput rate when the values from Table I are used.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1536</td>
<td>3</td>
<td>17.43</td>
<td>13.07</td>
<td>102.0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>136.1</td>
<td>107.8</td>
<td>72.64</td>
</tr>
<tr>
<td>4096</td>
<td>3</td>
<td>14.77</td>
<td>13.10</td>
<td>103.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>138.5</td>
<td>119.9</td>
<td>89.9</td>
</tr>
<tr>
<td>6144</td>
<td>3</td>
<td>14.78</td>
<td>13.11</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>139</td>
<td>104.2</td>
<td>94.4</td>
</tr>
</tbody>
</table>

As one can observe from Table 2, the serial decoding performance is close to the theoretical one. Let us consider for example the case $K=6144$ and $L=3$. The native theoretical latency is $4KL$ clock periods, which leads to a theoretical throughput of 17.5 Mbps, while the obtained results for the proposed serial implementation is 17.48 Mbps.

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. 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.

Fig. 11 depicts the obtained results when a block of length $K = 512$ was decoded in a serial manner, in a parallel without overlapping manner and in a parallel with overlapping manner. For this scenario $K = 512$, QPSK modulation was used and $L = 3$. Fig. 12 presents the same type of results, for the case of $K = 1024$.

As one can observe from Fig. 11 and 12, the parallel decoding with overlap is producing same results as the serial decoding.

On the other hand, the parallel decoding without overlap introduces a certain level of degradation compared with the serial decoding, the loss in terms of performances being dependent on the value of $N$.

### 6 Conclusions

The most important aspects regarding the FPGA implementation of a turbo decoder for LTE-A systems were presented in this paper. The serial turbo decoder architecture was developed and implemented in an efficient manner, especially from the interleaver/deinterleaver processes point of view. The interleaver memory ILM was introduced so that the interleaver process to work effectively only outside the decoding process itself. The ILM was written together with the input data, while the previous block was still under decoding. This approach allowed the transfer to the parallel architecture in a simplified way, using only concatenated values at same memory locations. The parallel architecture used the same number of block memories and only one interleaver, with the cost of some multiplexing/demultiplexing structures.

The parallel decoding performances were compared with the serial ones and certain degradation was observed. To eliminate this degradation, a small overhead was accepted by the overlapping split that was applied to the parallel data blocks.
Acknowledgment

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/134398.

References:


