U.P.B. Sci. Bull., Series C, Vol. 79, Iss. 1, 2017 ISSN 2286-3540

NOVEL PARALLEL CTC TURBO DECODER ARCHITECTURE FOR LTE SYSTEMS

Cristian ANGHEL[1], Cristian STANCIU[2], Constantin PALEOLOGU[3]

This paper describes a novel architecture for parallel turbo decoding used in Long Term Evolution (LTE) systems. The decoding scheme contains only one Quadratic Permutation Polynomial (QPP) interleaver, independently on the parallelization factor. Also, we propose an efficient implementation for the QPP interleaver. The arithmetic properties of the corresponding interleaver equations are exploited to perform the address computation in a recursive manner. The proposed decoding structure is evaluated in different radio environments and with several connection settings (modulation, data block length.)

Keywords: LTE, turbo coding, Max Log MAP, parallel architecture, decoding performance

1. Introduction

The channel coding is a physical layer procedure used by the wireless communications systems to increase the transmission robustness on the radio link. There are several methods to execute the error correcting coding: Reed-Solomon codes, BCH codes, cyclic codes, polynomial codes, and linear block codes. Turbo coding is one channel coding method, firstly introduced by Berrou, Glavieux, and Thitimajshima [1, 2, 3]. It uses two recursive systematic convolutional (RSC) constituent encoders connected in a parallel concatenation scheme.

Turbo codes faced two major problems along their existence. The first one was related to their reported performance close to Shannon limit. It was difficult for the technical community to accept such new theory. Once the controversy was clarified by the authors, turbo codes started being included in several communications standards. The Third-Generation Partnership Project (3GPP) [4] is one organization that adopted earlier turbo codes, while Universal Mobile Telecommunications System (UMTS) [5] and Long Term Evolution (LTE) [6, 7] are the most famous standards including them. The Institute of Electrical and Electronics Engineers (IEEE) group also made turbo codes part of most of the communication standards, one example being 802.16e [8] based on which

[1] Lecturer, Dept. of Telecommunications, University POLITEHNICA of Bucharest, Romania, e-mail: canghel@comm.pub.ro
[2] Lecturer, Dept. of Telecommunications, University POLITEHNICA of Bucharest, Romania, e-mail: cristian@comm.pub.ro
[3] Professor, Dept. of Telecommunications, University POLITEHNICA of Bucharest, Romania, e-mail: pale@comm.pub.ro

Worldwide Interoperability for Microwave Access (WiMAX) systems were developed. But due to their implementation complexity, especially on decoding part, the second issue appeared together with the discussion if the costs are covered by the benefits. The answer was brought and given by the technology evolution. New more powerful Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs) allowed efficient implementation for turbo decoders, while simplified decoding algorithms were proposed with decoding performance close to the classical reference. Starting from classic maximum a posteriori probability (MAP) algorithm [9], the new Logarithmic MAP (Log MAP) [9], Maximum Log MAP (Max Log MAP), Constant Log MAP (Const Log MAP) [10], and Linear Log MAP (Lin Log MAP) [11] algorithms were introduced.

For evolved wireless communications standards, such as LTE Advanced (LTE-A), a new challenge appeared for turbo codes. In order to support high data rates, lower decoding latency was requested. This demand applies especially for the biggest data blocks defined in [6]. The solution is the parallelization of the turbo decoding process.

There are many parallel decoding architectures proposed in the literature during 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 [12]. Starting from the classical method of implementing the 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 also being doubled. Another proposed scheme eliminates the need for the second interleaver but increases the decoding time with $K/2$ as compared to 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 [13, 14]. Here, efficient hardware implementations of the QPP interleaver are proposed. But the parallelization factor N represents also the number of used interleavers in these proposed architectures.

A third approach consists in using a folded memory to store simultaneously all the values needed for parallel processing [15]. 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 are 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. However, in this approach, *N* interleavers are also needed to generate all the interleaved addresses that input the master network.

In this paper, we also introduce a folded memory based approach, but the main difference as compared to the already existent solutions described above is that our proposed solution uses only one interleaver. Additionally, with an even-odd merge sorting unit [16, 17], 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 performance, with the cost of a small overhead added, the decoding results of the serial and parallel decoding architectures are kept similar. Moreover, for the single interleaver, we propose a solution that exploits key arithmetic properties of the corresponding equation to perform the address computation in a recursive manner. The proposed method replaces divisions and multiplications by comparisons and subtractions.

The paper is organized as follows. Section 2 describes the serial turbo decoding scheme and the proposed parallel turbo decoding architecture, with a single interleaver and with the re-ordering unit. Section 3 presents the proposed solution for interleaver implementation. Section 4 presents area and speed results obtained when targeting a XC5VFX70T [18] chip on Xilinx ML507 [19] board; it also provides simulation curves comparing the results obtained when using serial decoding, parallel decoding, and parallel decoding with overlap. Section 5 contains the conclusions of this work.

2. LTE turbo decoding architecture

2.1 Serial turbo decoding scheme

According to the theoretical decoding scheme, 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. This can be noticed in Fig. 1, where a serial decoder block scheme based on the previous work presented in [20] (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 we introduced compared with the previous serial implementation presented in [21] 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. 1 works as follows: SISO 1 reads the memory locations corresponding to $V_1(X_k)$ and $\Lambda^i\left(Z_k\right)$ vectors. The reading process is performed forward and backward and it serves the first semi-iteration. At the end of this process, SISO 2 reads forward and backward from the memory blocks corresponding to $V_2(X'_k)$ and $\Lambda^i\left(Z_k^{'}\right)$ vectors in order to perform the second semi-iteration.

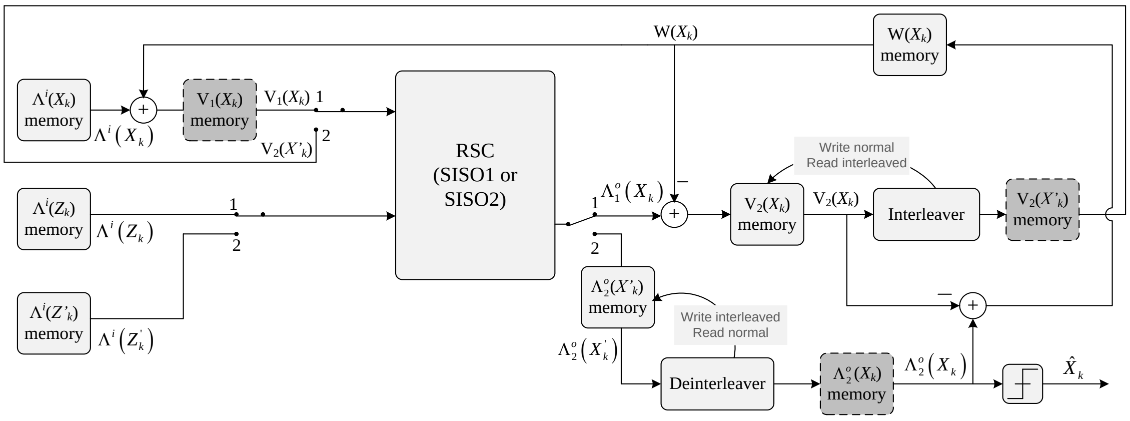


Fig. 1. Proposed serial turbo decoder block scheme

The vector $V_1(X_k)$ is obtained by adding the input vector $\Lambda^i\left(X_k\right)$ with the extrinsic information vector $W(X_k)$. 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 $W(X_k)$ memory; also the vector $V_2(X_k)$ 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 addresses for $V_2(X_k)$ are obtained) and the re-ordered LLRs $V_2(X'_k)$ are available, the corresponding values for the 3 tail bits X'_{K+1}, X'_{K+2}, X'_{K+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 in Fig. 1, two switching mechanisms are included in the scheme. When in position 1, the memory blocks for $V_1(X_k)$ and $\Lambda^i\left(Z_k\right)$ are used, while in position 2 the memory blocks for $V_2(X'_k)$ and $\Lambda^i\left(Z_k^{'}\right)$ become active.

At the output of the SISO unit, after each semi-iteration, K LLRs are obtained. The ones corresponding to the second semi-iteration are stored in the $\Lambda_2^o\left(X_k^{'}\right)$ memory [the ILM output, which was already available for the $V_2(X_k)$ interleaver process, is used as writing address for $\Lambda_2^o\left(X_k^{'}\right)$ memory, after a delay is added]. Reading in a normal order $\Lambda_2^o\left(X_k^{'}\right)$ memory and also $V_2(X_k)$ memory provides inputs for $W(X_k)$ memory and on the same time allows a new semi-iterations to start for SISO 1. Hence the $W(X_k)$ memory update is made on the same time with a new semi-iteration start.

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).

2.2 Parallel turbo decoding scheme

The proposed parallel architecture is similar to the serial one described in Fig. 1, 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 scheme, in the parallel one each memory contains K/N locations with $10N$ bits per location.

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 [18], 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. 2 describes the way ILM works.

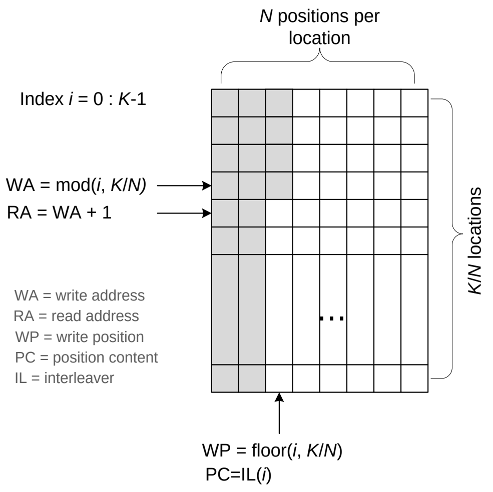


Fig. 2. ILM memory writing procedure

As one can observe, while writing procedure, each index *i* from 0 to $K-1$ generates 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 in 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 properties 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. 3.

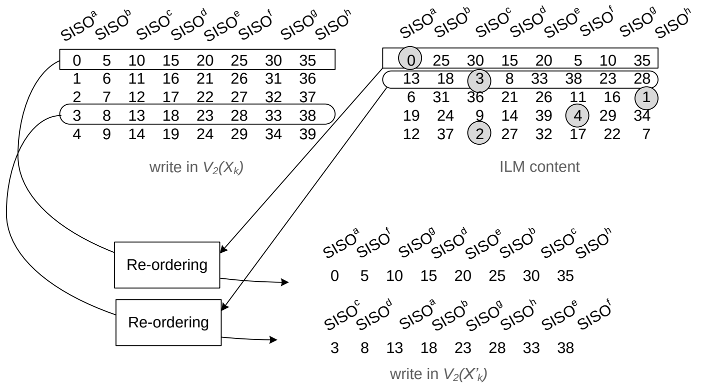


Fig. 3. Virtual parallel interleaver.

On the left side, 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 side, the content of ILM is described. The minimum values from each line of ILM (grey colour circle in figure) represents the line address for $V_2(X_k)$ memory. Then, using a re-ordering module, 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 also applies for deinterleaver process, but the write addresses are extracted from ILM, while the read ones are in normal order.

For the re-ordering module, an even-odd merge sorting network is considered. The even-odd merge sorting method is part of the sorting networks group that includes many other sorting types. One example of sorting network is the bubble sorting, which sorts in a repeated manner the adjacent pair of elements. Another example is the shell sorting, which uses an array to group the input data and then sorts the column of the array, in a repeated manner too, after each such iteration the array becoming one column smaller. A third example is the even-odd transposition sorting, which sorts alternatively odd-indexed and the adjacent even-indexed elements, respectively even-indexed and the adjacent odd-indexed elements. Finally, another example that should be mentioned in the category of network sorting is the bitonic sorting. This method starts sorting the 2 halves of the input data in opposite direction, then jointly sorting the 2 halves to produce one complete sorted sequence.

The even-odd merge sorting method was introduced by Batcher in [16]. It is based on a theorem saying that any list of $p=4k$ (k is a positive integer) elements can be sorted if the following steps are applied: first, the two halves of the list are sorted separately, then the odd-indexed elements and the even-indexed elements are sorted separately, and finally a comparing and switching procedure is executed over all the elements $2m$ and $2m+1$ ($m=1,..,p/2-1$). The proof of this theorem can be found in [17]. The example for $N=8$ is depicted in Fig. 4.

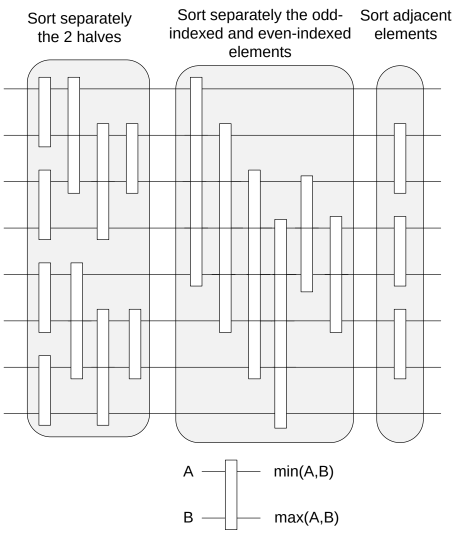
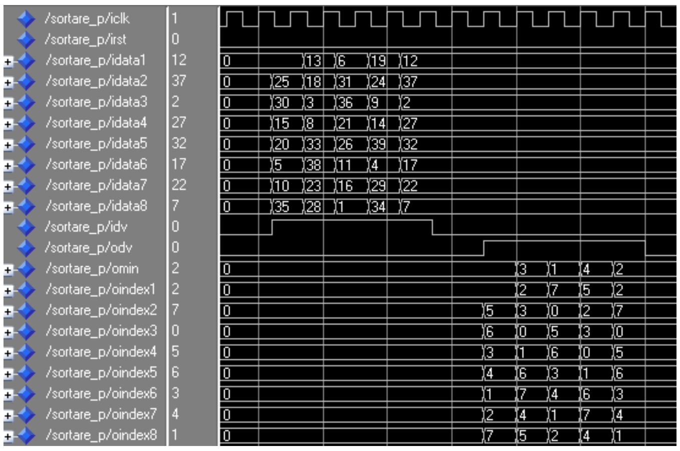

Sort separately the 2 halves
Sort separately the odd-indexed and even-indexed elements
Sort adjacent elements
A
B
min(A,B)
max(A,B)

Fig. 4. Even-odd merge sorting for *N*=8

Fig. 5 depicts a ModelSim capture for the case *K*=40 bits and *N*=8. One can observe at the input the ILM content (the 40 interleaved addresses organized in 5 memory locations, 8 addresses on each location), and at the output the minimum detected value for each ILM location (i.e., the normal-order memory location that shall be read) and the order data read from normal-order memory location shall be sent to the *N* decoding units.



```
/sortare_p/iclk     1
/sortare_p/irst     0
/sortare_p/idata1   12   0      13  6   19  12
/sortare_p/idata2   37   0  25  18  31  24  37
/sortare_p/idata3   2    0  30  3   36  9   2
/sortare_p/idata4   27   0  15  8   21  14  27
/sortare_p/idata5   32   0  20  33  26  39  32
/sortare_p/idata6   17   0  5   38  11  4   17
/sortare_p/idata7   22   0  10  23  16  29  22
/sortare_p/idata8   7    0  35  28  1   34  7
/sortare_p/idv      0
/sortare_p/odv      0
/sortare_p/omin     2    0      3  1  4  2
/sortare_p/oindex1  2    0      2  7  5  2
/sortare_p/oindex2  7    0  5   3  0  2  7
/sortare_p/oindex3  0    0  6   0  5  3  0
/sortare_p/oindex4  5    0  3   1  6  0  5
/sortare_p/oindex5  6    0  4   6  3  1  6
/sortare_p/oindex6  3    0  1   7  4  6  3
/sortare_p/oindex7  4    0  2   4  1  7  4
/sortare_p/oindex8  1    0  7   5  2  4  1
```

Fig. 5. Odd-even merge sort – Modelsim simulation

For example, at the second clock period, the second ILM location is read, i.e., the addresses 13, 18, 3, 8, 33, 38, 23, 28. The sorting module will allocate an index from 0 to 7 to these addresses and then will arrange them in an increasing order. Also, the minimum value is provided at the output, in this case 3. After this sorting procedure, it results that location number 3 is to be read from the normal-order data memory. Once the location content is available, the 8 samples from the location will be distributed to the 8 decoding units, as indicated by the output index. The first sample from location will be sent to SISO number 2, the second sample to SISO number 3, the third one to SISO number 0, and so on.

3. Proposed interleaver implementation

The inteleaver reorders the input sequence C_k as

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

where $\pi(i)$ is an address computed as

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

The parameters f_1, f_2, and the block length K are standardized in 188 possible sets of values and can be found in Table 5.1.3-3 in [6].

The apparent arithmetic requirements for the computation of the memory addresses $\pi(i)$ consist of one addition, three multiplications, and one division (which is used for the extraction of the remainder associated with the *modulo* operation). The associated denominators are the values of K and the remainders (the results of the modulo computations) have smaller values than the corresponding K lengths. Fig. 6 illustrates, for each of the possible data block lengths K (i.e., each of the intervals i=0,…,K-1), the maximum values of the *dividends* and *quotients* associated with (2). It can be noticed that the minimum hardware resources necessary for finite numeric formats must account for representations of values up to billions for the *dividents* and millions for the *quotients.*

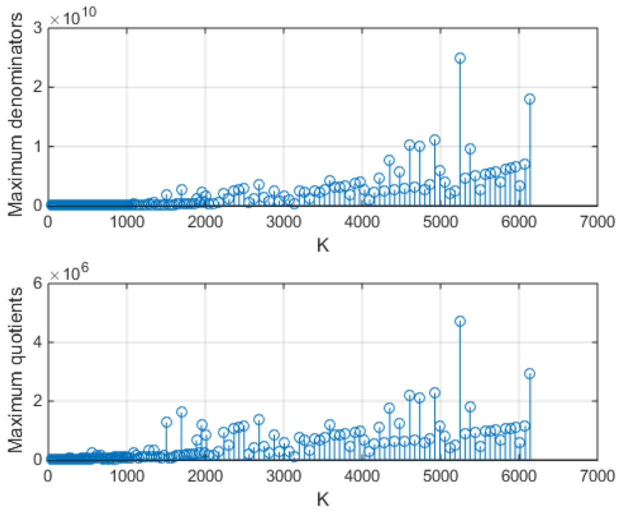


Fig. 6. Maximum dividents and quotients for the interleaver address generator

The values illustrated in Fig. 6 require up to 35 bits, respectively 23, for unsigned integer representations. Furthermore, the hardware implementations must generate the values $\pi(i)$ with a minimum delay, requiring a *pipe-line* arithmetic. The large numeric ranges and the pipe-line system occupy large chip areas.

By introducing the notation:

$$p(i) = f_1 i + f_2 i^2 \tag{3}$$

it can be observed that

$$\begin{aligned} &p(0) = 0, \\ &p(i) = p(i-1) + s_1 + s_2(i),\ i > 0 \end{aligned}, \tag{4}$$

where

$$s_1 = f_1 \text{ and}$$

$$s_2(i) = \begin{cases} 0,\ i = 0, \\ f_2,\ i = 1, \\ s_2(i-1) + 2f_2, i > 1. \end{cases} \tag{5}$$

We can re-write (2) using (3) and (4)

$$\pi(i) = p(i) \bmod K = \left[p(i-1) + s_1 + s_2(i)\right] \bmod K. \tag{6}$$

The multiplications are replaced by additions and the arithmetic complexity is reduced. Nevertheless, the division is still required for the *modulo* operation. Considering that the *modulo* operator applied to a sum of elements can be expressed as

$$\left[\sum_k c_k\right] \bmod K = \left[\sum_k c_k \bmod K\right] \bmod K, \tag{7}$$

we propose to modify the computation of $\pi(i)$ in (6) to considerably reduce the arithmetic complexity. The number of modulo operations increases, but the complexity of the corresponding divisions is reduced as a consequence of having smaller quotients. Consequently, using (5), (6), and (7), we obtain:

$$\begin{aligned} \pi(i) &= \left[\ p(i-1) \bmod K + s_1 \bmod K + s_2(i) \bmod K\right] \bmod K \\ &= \left[\pi(i-1) + f_1 \bmod K + \left(s_2(i-1) + 2f_2\right) \bmod K\right] \bmod K \\ &= \left[\pi(i-1) + f_1 + s_2(i-1) \bmod K + 2f_2 \bmod K\right] \bmod K. \end{aligned} \tag{8}$$

By taking into account in (8) that f_1 and f_2 are constant values for a given frame length *K*, their contribution to the function *p*(*i*) can be included into a single pre-computed value. Therefore, the functions s_1 and $s_2(i)$ can be combined into a single step function:

$$s_3(i) = s_1 + s_2(i) = \begin{cases} 0, \; i = 0, \\ f_1 + f_2, \; i = 1, \\ s_3(i-1) + 2f_2, i > 1. \end{cases} \tag{9}$$

By using (9) in (6), it results in

$$\begin{aligned} \pi(i) = p(i) \bmod K \; &= \left[p(i-1) + s_3(i) \right] \bmod K \\ &= \left[p(i-1) + s_3(i-1) + 2f_2) \right] \bmod K \\ &= \left[\pi(i-1) + s_3(i-1) \bmod K + 2f_2 \bmod K \right] \bmod K. \end{aligned} \tag{10}$$

All of the values in the last stage of (10) are lower than the value K, and available recursively, such as $\pi(i-1)$ and $s_3(i-1) \bmod K$, or they can be predetermined and stored, like the case of $2f_2 \bmod K$.

The overall arithmetic complexity of the address generation module is reduced from 3K additions and 3K simplified modulo operations corresponding to (8), to 2K additions and 2K simplified modulo procedures associated with (10).

The method improves the solutions presented in [22, 23, 24], by eliminating any multiplications or divisions. Additionally, the low numerical range of the operators (with numbers lower than 2K) allows the usage of minimal resources for the representation of binary values.

4. Experimental results

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 the above-mentioned tools, the obtained results when implementing the decoding structure on a Xilinx XC5VFX70T-FFG1136 are the following: frequency of 310 MHz and 664 Flip Flops and 568 LUTs for sorting unit.

For the interleaver, the proposed hardware design requires 125 Slice Registers and 229 LUTs (Look-Up-Tables), with a maximum clock frequency of 224.459 MHz (equivalent to a minimum clock period of 4.455 ns). Some comparative results in terms of used resources were provided in [15] [25], but for Application-Specific Integrated Circuit (ASIC).

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. The figures 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. 7 depicts the 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. In this scenario, N = 2, QPSK modulation was used and L = 3. Fig. 8 presents the same type of results, for the case of K = 1024 and N = 4. As one can observe from

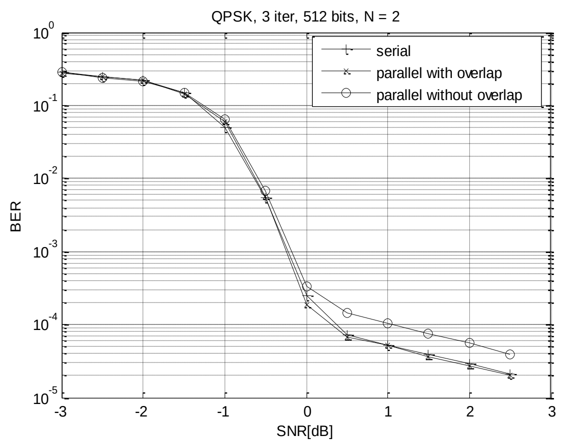


Fig. 7. Comparative decoding results for QPSK, L = 3, K = 512, N = 2

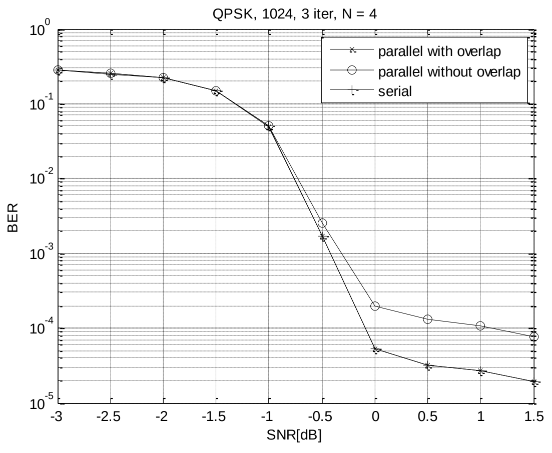


Fig. 8. Comparative decoding results for QPSK, L = 3, K = 1024, N = 4

Figs. 7 and 8, the parallel decoding with overlap is producing the same results as the serial decoding.On the other hand, the parallel decoding without overlap introduces a certain level of degradation as compared to the serial decoding, the loss in terms of performance being dependent on the value of N.

6. Conclusions

The most important aspects regarding the implementation of a turbo decoder for LTE 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, adding only an even-odd merge sorting network. The single interleaver was implemented in an efficient recursive manner, replacing divisions and multiplications by comparisons and subtractions. 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.

Acknowledgement

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.

R E F E R E N C E S

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

[2] *C. Berrou and A. Glavieux*, Near optimum error correcting coding and decoding: Turbo-Codes, *IEEE Trans. Communications*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

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

[4] Third Generation Partnership Project. 3GPP home page. www.3gpp.org.

[5] *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, Oct. 2001.

[6] 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).”

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

[8] IEEE organization, https://standards.ieee.org/about/get/802/802.16.html

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

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

[11] *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.

[12] *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

[13] *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, issue 4, pp. 305-315, Sept. 2011.

[14] *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.

[15] *C. Studer, C. Benkeser, S. Belfanti, and Quiting Huang*, Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE, *IEEE Journal of Solid-State Circuits*, vol. 46, issue 1, pp 8-17, Jan. 2011.

[16] *K. E. Batcher*, Sorting Networks and their Applications, in Proc. *AFIPS Spring Joint Comput. Conf.*, Vol. 32, 1968.

[17] Massachusetts Institute of Technology, Mathematics, *math.mit.edu/~shor/18.310/batcher.pdf*

[18] Xilinx Virtex 5 family user guide, www.xilinx.com.

[19] Xilinx ML507 evaluation platform user guide, www.xilinx.com.

[20] *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), Menuires, France, April 2010.

[21] *C. Anghel, V. Stanciu, C. Stanciu, and C. Paleologu*, CTC Turbo Decoding Architecture for LTE Systems Implemented on FPGA, *Eleventh International Conference on Networks* (ICN), Reunion, France, February 2012.

[22] *Di Wu, R. Asghar, Yulin 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.

[23] *R. Asghar, Di 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, issue 1, pp. 15-19, July 2010.

[24] *S. Wang, L. Liu, and Z. Wen*, High Speed QPP Generator with Optimized Parallel Architecture for 4G LTE-A System, *Int. Journal of Advancements in Computing Technology*, vol. 4, issue 23, pp. 355-364, July 2010.

[25] *E. Mumolo, G. Capello, and M. Nolich*, “VHDL design of a scalable VLSI sorting device based on pipelined comutation,” in Journal of Computing and Information Technology - CIT 12, 2004, Vol 12, No. 1, pp. 1–14.