Implementation Techniques for low latency turbo decoders

Técnicas para la implementación de turbo decodificadores con baja latencia

YLLESCAS-CALDERON, Lennin Conrado†*, PARRA-MICHEL, Ramón and GONZALEZ-PEREZ, Luis Fernando

CINVESTAV del IPN, Unidad Guadalajara, Laboratorio de Telecomunicaciones
A2e Technologies, Engeenering Department

ID 1st Author: Lennin Conrado, Yllescas-Calderon / ORC ID: 0000-0002-8620-5263, CVU CONACYT ID: 267118

ID 1st Coauthor: Ramón, Parra-Michel / ORC ID: 0000-0003-2327-2482, CVU CONACYT ID: 123034

ID 2nd Coauthor: Luis F. Gonzalez-Pérez / ORC ID: 0000-0002-9097-361X, CVU CONACYT ID: 25582

DOI: 10.35429/JSTA.2019.15.5.29.41

Received April 01, 2019; Accepted June 30, 2019

Abstract

Turbo coding is a channel coding technique that increases the capacity of communications systems, especially wireless and mobile. Due to its high correction capability, this technique is used in modern wireless communication standards such as 3GPP and LTE/LTE-Advanced. One of the features of these systems is the increase in data processing capacity, where transmission rates of up to 1 Gbps are specified. However, the turbo coding technique inherently presents a limited performance as a consequence of the turbo decoding process at the reception stage. The turbo decoder presents a high operation latency mainly caused by the iterative decoding process, the interleaver and deinterleaver stage and the estimation process of the information bits. In this work, we show the techniques used to implement modern low-latency turbo decoders suitable for 3G and 4G standards.

Turbo coding, Sliding Windows, Parallelism

Resumen

La turbo codificación es una técnica de codificación de canal que aumenta la capacidad de los sistemas de comunicaciones, especialmente los inalámbricos y móviles. Debido a su alta capacidad de corrección, esta técnica se utiliza en los estándares modernos de comunicaciones inalámbricas como el 3GPP y el LTE/LTE-Advanced. Una de las características de estos sistemas es el incremento en la capacidad de procesamiento de datos, donde se especifican tasas de transmisión de hasta 1 Gbps. Sin embargo, el proceso de turbo codificación presenta inherentemente un rendimiento limitado debido al proceso de turbo decodificación en la etapa de recepción. El turbo decodificador presenta una alta latencia de operación debido al proceso iterativo de decodificación, las etapas de entrelazado y deentrelazado y al proceso de estimación de los bits. En este trabajo se presentan las técnicas que se utilizan para realizar la implementación de turbo decodificadores modernos de baja latencia que permitan su operación en los actuales estándares de comunicación (3G y 4G).

Turbo codificación, Ventanas deslizantes, Paralelismo

Citation: YLLESCAS-CALDERON, Lennin Conrado, PARRA-MICHEL, Ramón and GONZALEZ-PEREZ, Luis Fernando. Implementation Techniques for low latency turbo decoders. Journal of Scientific and Technical Applications. 2019, 5-15: 29-41.

* Correspondence to Author (email: lyllescas@gdl.cinvestav.mx)

† Researcher contributing first author.

Introduction

Wireless communications have had a great development in recent years, due to the growing demand for better quality services and affordable costs. One of the areas that have allowed this development is turbo coding. Turbo coding is one of the most advanced techniques for the correction of errors that exist, since its performance is close to the theoretical limits of channel capacity, which is why they have been incorporated into current mobile communication systems such as Wimax, WCDMA, 3GPP-LTE. However, LTE/LTE-Advanced are radio technologies that take consumers to a new level of mobile broadband access [1]. 3GPP-LTE specifies maximum speeds of up to 326 Mbps on the downlink and 86.4 Mbps on the uplink. On the other hand, for LTE-Advanced, the maximum speeds are 1 Gbps on the downlink and up to 500 Mbps on the uplink. However, the turbo decoding stage has a high processing latency due mainly to its iterative nature, the pseudo random nature of interleaver and deinterleaver of data and the recursive processes in the estimation of the information bits. For this reason, it is important to make adjustments in the turbo decoding process that allow increasing the speed of the decoding process. There are different proposals that can be used to reduce the decoding latency of turbo decoders and achieve the required performance. The approaches range from solutions at high and medium level, to optimizations at the level of arithmetic operations. In this paper we explore the various techniques for reducing latency of turbo decoding. From this, an architecture is proposed for a turbo LTE decoder. The LTE standard requires high transmission rates by which different techniques for reducing latency and raising the transfer rate are incorporated. The incorporation of different techniques increases hardware resources (HW), which is why adjustments are made to make resource consumption more efficient. This document is organized as follows. In section 2, the theory of turbo codes is presented. In section 3, the techniques for reducing latency of the turbo decoding process are presented. In section 4, the architecture of the low-latency decoder turbo is presented, where different techniques proposed in section 3 are incorporated. In section 5 the implementation results are presented and finally, in section 6 the conclusions are presented.

1. Fundamentals of Turbo coding

Since their proposal in 1993 [3], turbo codes have received great attention for their high correction capacity and low decoding complexity. Turbo coding has two parts: the encoder used in the transmitter and the decoder used in the receiver. The turbo encoder consists of two component encoders connected in parallel and separated by an interleaver. The turbo decoder consists of two decoders which decode each of the sequences encoded by the component encoders respectively. The decoders are linked by interleavers and deinterleavers.

1.1. Turbo LTE encoder

The structure of the turbo encoder, specified in the LTE standard [1], is shown in Figure 1. Each component encoder is a Recursive Systematic Convolutional (RSC) of 8 states. The convolutional encoder is a state machine that can be represented by a trellis diagram like the one shown in Figure 2.

Figure 1 Structure of the turbo LTE encoder

The dashed lines represent the state transitions caused by a bit of value information 1. The solid lines represent state transitions when the information bit is 0. Each line, known as a branch, is linked to the input and output of the convolutional encoder (binary label associated with each branch of the trellis). When a bit of information enters the encoder, a two-bit code word is generated at the output that depends on its current state. The coding rate is 1/3, since 3 bits are transmitted for each bit of information.

Figure 2 Trellis diagram of the LTE turbo encoder

The interleaver represented as Π , receives a sequence of data $C = (c_0, c_1, \dots, c_{K-1})$, and exchanges its positions to obtain the sequence $C' = (c'_0, c'_1, \dots, c'_{K-1})$, where the relationship between input and output is given as $c'_x = c_{\pi(x)}$, for $x = 0, 1, \dots, K - 1$, where K represents the size of the data block or frame. In the LTE standard the relationship between input and output is specified by the expression $c'_x = \pi(x) = (f_1x + f_2x^2) \bmod K$, where K can take 188 different values in a range of 40 to 6144, and $x = 0, \dots, K - 1$. The values f_1 and f_2 are defined by the LTE standard [1]. The interleaver can be seen as a block of memory where the data is stored sequentially and extracted pseudo randomly according to $\pi(x)$. The deinterleaver restores the sequence of data to its original order.

1.2. Turbo LTE decoder

The turbo decoder is based on an iterative algorithm [2] and its structure is shown in Figure 3, where the inputs of the decoders are soft inputs and soft outputs (SISO) [3].

Figure 3 Structure of the turbo decoder

The SISO decoders (Dec. SISO) receive the information coming from the channel $(L_C y_{ks}, L_C y_{k1}, L_C y_{k2})$ corresponding to the systematic and parity bits, as well as the a priori information $L_{1,2}(u_k)$, where L_C is the channel reliability value. At the output, an estimate of the transmitted bits Λ_k is obtained through the use of the maximum a posteriori probability algorithm (MAP) in the logarithm domain, called the Log-MAP algorithm [4], using the following equation

$$\Lambda_k = \max_{u_k=1}^* \left(A_{k-1}^{s'} + G_k^{(s',s)}(u_k) + B_k^s \right) - \max_{u_k=0}^* \left(A_{k-1}^{s'} + G_k^{(s',s)}(u_k) + B_k^s \right), \tag{1}$$

Whichh is based on three probabilities related to the coded states S_k^s , being $k \in \{1, \dots, K\}$ and s and $s' \in \{1, \dots, 8\}$. The first is known as branch metric $G_k^{(s',s)}(u_k)$, and is the probability that the transition between the previous states $S_{k-1}^{s'}$ and the state S_k^s , given that the bit that makes the transition (information bit) is $u_k = 1$ or $u_k = 0$. $A_{k-1}^{s'}$ and B_k^s are known as state metrics. $A_{k-1}^{s'}$ represents the probability that the trellis is in state s' , in time $k-1$, since the sequence of the channel has been received from the beginning of the frame to the $k-1$ element of the plot. B_k^s represents the probability that the trellis is in state s , at time k , since the sequence of the channel from element k to the final element of the frame has been received. The calculation of the metrics is made from the equations

$$G_k^{(s',s)}(u_k) = \frac{1}{2} (u_k \cdot L(u_k) + L_C \cdot \sum_{l=1}^n y_{kl} \cdot x_{kl}) \tag{2}$$

Where n is the number of channel outputs that enter the SISO decoder and x_{ki} ($i = 1, 2$) are the output bits (transmitted) that are associated with the branch tag of Figure 2.

$$A_k^s = \max^* \left(A_{k-1}^{s'} + G_k^{(s',s)}(u_k) \right) \tag{3}$$

$$B_{k-1}^{s'} = \max^* \left(B_k^s + G_k^{(s',s)}(u_k) \right) \tag{4}$$

Where the values A_k^s and B_k^s are obtained from recursive processes one forward (Fw) and one backward (Bw) respectively.

For this, the data block is partitioned into segments of size L and each segment is individually decoded as shown in Figure 6. It can be seen that a backward recursion is performed in the second window (dotted lines), which is known as learning stage or convergence. This process allows obtaining the acceptable B metrics that initialize the process backwards in the first window (continuous line).

Figure 6 Window technique

The process of decoding in a window consists of five periods of time. The calculation of the branch metric (G) is performed in the time period t_0 , the state metrics (A and B) are calculated in the time periods t_1 , t_2 and t_3 , and the extrinsic information L_e is obtained in the period t_4 . It can be observed that to obtain L_e it is necessary to store 4 values of G , 8 values of A and 8 values of B , to have them in the calculation of the extrinsic information fourth period, thus the memory requirements are $20L$.

2.2. Parallelism

Parallelism has been introduced at different levels of the turbo decoder such as at the frame level, at the iteration level, at the intra-frame level and at the symbol level [7 and 8]. The frame and iteration level parallelism consists in replicating the turbo decoder to process data blocks in parallel and process iterations in the form of a pipeline, respectively. These types of parallelism consume a lot of resources since the turbo decoder is fully replicated. Therefore, parallelism is commonly exploited at the intra-frame level and at the symbol level.

2.2.1 Parallelism at symbol level

This parallelism is the lowest level, and involves all operations in the decoding of each symbol received in the Log-MAP.

Two approaches can be distinguished: parallelism $\max^*$ and parallelism of state metrics.

2.2.1.1. Parallelism at $\max^*$ level

In equations (3 and 4) it is observed that 8 $\max^*$ operations are used for each symbol for forward recursion and 8 for backward recursion. Thus, by replicating all the operators, it is possible to reduce the processing time by a factor of 8 for each of the recursions. In the case of equation (5) for each decoded symbol, 14 $\max^*$ operations are used, reducing the time τ_Λ by a factor of 14.

2.2.1.2. Parallelism at state level metrics

The parallelism of state metrics consists in parallel executing the calculations of the branch metrics, the state metrics and the calculation of the extrinsic information which allows reducing the processing time of the decoder SISO by a factor of 4.

2.2.1. Parallel timing schemes at symbol level

In the state of the art there are different proposals. The parallelism at the symbol level is the most exploited. Figure 7 shows the Forward-Backward scheme where the A and G values (dashed lines) are first calculated and then the B and L_e values (solid lines), obtaining a processing time in the 2K SISO decoder.

Figure 7 Forward-Backward scheme

The memory requirements are $12K$. Figure 8 shows the butterfly scheme, where the two recursions begin simultaneously. In a first stage (continuous lines) they are executed in parallel A , B and G and subsequently they are executed in parallel A , B and L_e , thus reducing the processing time of the SISO decoder to K .

Figure 8 Butterfly scheme

The memory requirements are similar to the Forward-Backward scheme. In figure 9, the sliding window scheme [9,10] is presented, where the data block is partitioned into windows of size L .

Figure 9 Sliding window scheduling

The process starts with the calculation of G and A . Subsequently, the calculation of B and L_e starts in the fourth window, however it is necessary to obtain the initialization values of B , which are obtained in the previous window that start in the third window (B with dashed lines). Thus, the processing time of the SISO decoder for this scheme is $K + L$, however, the memory requirement is $3L$.

2.2.2. Intra-frame parallelism

The intra-frame parallelism consists in replicating the SISO decoder where each one executes the Log-MAP algorithm. Two proposals are distinguished at this level: shuffle parallelism and subframe parallelism.

2.2.2.1. Parallelism at the subframe level

Parallelism at the subframe level is initially considered in [11] and subsequently retaken by [12].

It consists of dividing into p segments the block of data coming from the channel and each segment of data is processed by a SISO decoder. Figure 10 shows the architecture of a turbo decoder with p parallel windows of $L=K/P$ trellis stages or code words to be decoded. When processing p sub-blocks in parallel the decoding time is reduced by a factor of p . However, resources multiply by a factor of p .

Figure 10 Parallelism at the subframe level

2.2.2.1. Shuffle parallelism

The shuffle parallelism consists in executing in parallel the two half iterations that integrate an iteration [8]. Figure 11 shows the architecture of the shuffle parallelism. For this, the SISO decoder is replicated twice and one operates with the part corresponding to the sequential data and the other with the interleaver data.

Figure 11 Shuffle parallelism

2.3. Interleaver conflict management

The interleaver of the data is carried out by means of a storage block and an addressing stage where a process of reading and writing is carried out per unit of time. However, to introduce a parallelism at the subframe level, the interleaver must attend to the SISO decoders so it must have p inputs and outputs.

Therefore, the memory must be partitioned into p sub-blocks that allow the extraction and introduction of values independently. However, this partitioning coupled with the pseudo-random permutation process can cause conflicts in memory access, as is the case with the 3GPP and 3GPP2 standards. That is, when two or more SISO decoders request writing or extracting data to the same memory subblock at the same time, in other words, multiple values must be extracted from (entered into) the same memory in an instant. An example is shown in Figure 12 where SISO 0 and 3 want to access MEM 1 simultaneously. To solve this, three alternatives are managed: conflict resolution at runtime, solution at design time and conflict prevention.

2.3.1. Runtime Conflict Resolution

It consists in resolving conflicts at the time of the interleaver operation. The objective is to hide the collisions by means of a buffer system to the entries of the memories to store the data that cannot be written or read in a given moment of time [12]. In [13] a ring-shaped network of p bottleneck interrupter cells, placed between the SISO decoders and the interleaver memories, where each cell is connected to the previous and the next cell to form the ring structure is introduced.

Figure 12 Conflicts in memory access in the interleaver

When an SISO decoder generates a L_e , an identification of the destination sub-block is attached so that in the interleaver it is transferred through the ring to the cell that is connected to the destination sub-block, where it is stored in the memory sub-block or in case of collision in an internal buffer of the cell.

On the other hand, lateral buffers are introduced to manage the data circulating simultaneously in the ring.

In [14] an approach with Network on a Chip (NoC) is presented. In this architecture, control flows are introduced into the network to reduce the size of buffers and as a consequence the area. This type of approach allows independence of the permutation scheme architecture, which in turn allows supporting different standards, however, multi-standard schemes implicitly have a greater requirement of hardware resources in addition to greater latency.

2.3.2. Conflict resolution at design time

This solution lies in making an association between the inputs and outputs without collision problems. In [15] it is proposed to introduce three levels of permutation where two permutation networks are used that separate the memories of the SISO decoders. These networks perform a spatial permutation while entering the data into the memories, a temporary permutation is performed. To obtain the appropriate values of the control of the three stages, it is necessary to preprocess the permutation patterns, which requires additional memories to store these values.

2.3.3. Conflict prevention

Prevention is based on defining a permutation scheme that allows the introduction and extraction of data without collisions. There are different proposals that allow mapping without collisions.

In [16] a divisible interleaver method where they introduce the restriction of parallelism to the S-Random interleaver is presented. In the S-Random interleavers the inputs, within a distance S , must be mapped to the outputs with a separation greater than S . The restriction of parallelism establishes that the data in the same index i within each sub-block are mapped to the same j index in each sub-block. In [17] the use of reduced interleavers is presented where they introduce the parallelism and S-Random restrictions. Likewise, the construction of an interleaver of large length and obtaining shorter lengths is proposed, eliminating the last positions.

In [18] a scheme based on almost regular permutations (ARP) is introduced, which is based on circular permutations where the data is stored in a linear memory from 0 to K-1 and is extracted by circular displacements introducing thus the disorganization of the data. In the LTE standard [1] an interleaver based on quadratic permutations polynomials (QPP) on integer rings [19] is introduced. The QPP interleavers are designed to be collision free which allows parallelism. An interleaver is collision free if and only if it satisfies

$$\left\lfloor \frac{f(x+n_1L)}{L} \right\rfloor \neq \left\lfloor \frac{f(x+n_2L)}{L} \right\rfloor, \tag{9}$$

Where $0 \leq x < L$, $0 \leq n_1, n_2 < L$ and $n_1 \neq n_2$ [NBCB08]. The terms on both sides of the inequality represent the addresses of memory in the interleaver that are accessed simultaneously by the n_1 -th and n_2 -th SISO decoders.

2.4. Trellis Compression

This technique seeks to simultaneously process several contiguous stages of the trellis or code words. In a trellis where $m=1$ number of bits are processed in each state, there are 2^m output transitions. This process is known as radix- 2^m , so for $m=1$ you have a radix-2. It can be calculated simultaneously $m = 1,2,3,4$ stages of the trellis or by transformations that result in structures called radix-4 [20,21], radix-8 [22] and radix-16. According to the radix- 2^m it is possible to reduce the processing time of the SISO by a factor of m .

2.5. Pipeline

In this technique, records are introduced in the architecture to cover mainly two objectives: the first is to reduce the critical paths and the second is to reuse the different units for the calculation of the metrics that make up the SISO decoder [23]. For the introduction of the pipeline stages, the positions defined by the dotted lines in the Log-MAP decoder of Figure 5 are considered.

2.6. Retemporization

It consists of relocating the records in a digital circuit to improve their performance, area or power characteristics, without affecting their behavior at the outputs [23].

An example is the re-timing of the ACSO operator, in which the register is transferred to the multiplexer output to reduce the critical path, so it is necessary to register the output of the LUT to maintain the behavior of the outputs. Figure 13 shows the architecture of this structure called compensation, sum, comparison and selection (OACS).

Figure 13 Structure of the OACS

2.7. Restructuring

It consists of rethinking the structure of the interleaver through the analysis of the Log-MAP equations. In [24,25], a simplified structure of the turbo decoder has been proposed where the interleaver associated with the systematic symbols entering the SISO 2 decoder is eliminated. Figure 14 shows the structure of the simplified decoder turbo.

Figure 14 Turbo decoder modified

The performance in the correction capacity is not affected by changes in the structure of the turbo decoder.

3. Application

A turbo decoder for a high speed system must have low latency.

In this sense, the modified decoder turbo has a shorter processing time since the interlacing process at the entrance of the SISO 2 decoder is suppressed. On the other hand, the parallelism at a low level and at a subframe level, according to what was stated above, reduces the processing time of the SISO decoder and must therefore be considered to reach high speeds.

The introduction of parallelism at the subframe level requires the interleaver to respond to the multiple requests of the SISO decoders, which may introduce conflicts in memory access. However, the LTE interleaver is collision free and allows parallelism as long as equation (9) is satisfied. The values that were selected and that comply with this restriction are $p = 8$, for K values from 40 to 512; $p = 16$, for K values from 528 to 1024; $p = 32$, for values of K from 1056 to 2016; and $p = 64$, for K values from 2048 to 6144.

The butterfly scheme is the one with the shortest processing time, which is why it is presented as the first option to integrate it into the architecture. However, according to equation (8) the throughput depends on the clock frequency, so it is necessary to reduce the critical paths in order to raise the throughput. In this sense, the pipeline is mandatory to raise the operating frequency. Under these considerations, we have $\tau_{SISO} = L + \tau_{propagation}$, where the propagation time is the time it takes to travel the pipeline stages. In the case of memory resources, for each SISO decoder $20L$ memory locations are required that lead to a total of $20Lp$.

The introduction of parallelism increases resources in HW, so it is considered the sliding window technique that allows to reduce memory consumption. As stated earlier, sliding windows increase the processing time due to the learning period of backward recursion. However, the initialization values can be obtained from the recursions of the previous iterations, thus eliminating the period corresponding to the learning stage. Thus, the $\tau_{SISO} = L + \tau_{propagation}$ and the memory requirements are $20wp$, w being the size of the sliding windows. Now, if the highest degree of parallelism is $p=64$, and if we propose a $w=64$, we see that the maximum memory consumption, for the maximum size of K , is 81920 memory locations.

On the other hand, if no sliding windows are introduced, the memory consumption rises to 122880 memory locations. Thus, with sliding windows, memory consumption is reduced by approximately 33%.

The scheduling of parallel windows with sliding windows and initialization of values from the previous iteration is shown in Figure 15.

Figure 15 Scheduling of parallel windows and sliding windows. The tables are the initialization metrics A and B from the previous iteration (blue and red respectively). Circles are the initialization metrics for the next iteration

It can be seen that each parallel window is partitioned into L/w windows of size w . In the first half of each sliding window, of size w , only the calculation of the metrics A, B and G is performed and stored in memory. During the second half of the sliding windows, the calculation of the metrics continues, but the values obtained are no longer stored, instead they are used, together with the values that were stored in the first half window, to obtain the extrinsic information L_e .

To reduce the latency of turbo decoding, the interlacing process is carried out simultaneously with the SISO decoding process. In this way the latency of turbo decoding is $2I(L + \tau_{propagation}) + \tau_I$. Now, by entering the data to the turbo decoder in parallel, τ_I can be reduced to L . In the end an architecture with a throughput that is expressed as follows is obtained

$$th = \frac{f_{max} \cdot K}{2I(L + \tau_{propagation}) + L} \quad (8)$$

3.4. Architecture

Figure 16 shows the architecture of the turbo decoder, which is composed of $p=64$ SISO Decoders and supports the four degrees of parallelism discussed. When considering sliding windows of size $w=64$, values of $L < w$ can be obtained, which occurs when there are frames smaller than 4160, so the sliding window process is only carried out in frames larger than 4160. In the turbo architecture SISO decoders have six inputs and four outputs. Two data flows, one forward and one backward, come from the symbol memories. From the Π/Π^{-1} unit there are two data buses corresponding to the two flows, forward and backward of the a priori information. Finally, from the adjacent SISO decoders come the metrics to initialize the forward and backward recurrences B. As outputs, the SISO units deliver the extrinsic information to the Π/Π^{-1} unit, which are generated in both directions forward and backward. Finally, the initialization metrics for forward and backward recursions are sent to adjacent SISO decoders.

Figure 16 Architecture of the turbo decoder LTE

The Π/Π^{-1} unit has up to 128 inputs and outputs to receive the extrinsic information corresponding to the forward and backward data flows of the 64 SISO decoders and delivers two forward and backward flows of the information a priori during the second half of the iteration; in the end, it delivers the decoded sequence.

4. Results

The proposed architecture was implemented in a FPGA Altera Stratix IV (EP4SE820F43C3). For this, 5 bits are considered for channel outputs, 10 bits for internal operations and 8 bits for extrinsic information values. The proposed architecture uses combinational 231,835 ALUT (adaptive search tables), 25,661 ALUTs of memory, 165,776 records of dedicated logic and 254,960 bits of block memory and a maximum frequency of 127 MHz is reached. Table 1 shows the reached throughput with this frequency according to equation (8).

I	th (Mbps)	I	th (Mbps)
2	1733.9	6	586.6
3	1164.6	7	503.4
4	876.7	8	440.8
5	702.9	9	392.1

Table 1 Throughput ($K = 6144$, $P = 64$)

The throughput is calculated for a 6144 frame which is decoded with the 64 SISO decoders, where the turbo decoder has a $\tau_{propagation}$ of 14 clock cycles. The $\tau_I = L$, as mentioned in the previous section, however, for the implementation it was considered to perform in parallel the reception of the channel outputs and the operation of the SISO decoders, reducing the $\tau_I = 10$. It can be seen that all the values of I cover the LTE standard and with 3 iterations the speed set for the LTE-Advanced standard is reached.

Figure 17 Performance curves of the LTE decoder turbo for $I = 5$

Figure 17 shows the bit error rate (BER) with respect to the signal to noise rate (SNR) with different sizes of K and p , at five iterations. For this, a binary phase shift keying (BPSK) with additive white Gaussian noise (AWGN) was used. The performance curves show an average degradation of 0.1 dB of the implementation in HW (Fix) compared to the floating point model (Flo).

5. Conclusions

In this paper the main problems of the turbo decoding algorithm for applications with high transmission rates were analyzed. Likewise, the different techniques used for latency reduction were presented. By integrating different techniques, the speeds required for the LTE/LTE-Advanced standards are achieved. On the other hand, incorporating the sliding window technique reduces memory consumption by 33%. In the end, the performance curves showed a loss of about 0.1 dB with respect to the floating point model.

6. Acknowledgments

This work was carried out with the support of ANR-CONACYT TOLTECA project No. 273562.

References

1. 3GPP. (September 2008). Evolved universal terrestrial radio access (E-UTRA); Multiplexing and channel coding.
2. C. Berrou, A. G. (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. Proceedings of ICC '93 - IEEE International Conference on Communications (págs. 1064-1070). Geneva, Switzerland: IEEE.
3. Sklar, B. (2001). Digital Communications. Fundamental and Applications. Los Angeles: Prentice Hall.
4. Lajos Hanzo, T. H. (2002). Turbo Coding, Turbo Equalisation and Space-Time Coding: For Transmission over Fading Channels. Wiley.
5. Robertson, P., Villebrun, E., & Hoeher, P. (1995). A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain. Proceedings IEEE International Conference on Communications ICC '95. Seattle, USA: IEEE.
6. F. Raouafi, A. D. (1999). Saving memory in turbo-decoders using the max-log-MAP algorithm. IEE Colloquium on Turbo Codes in Digital Broadcasting - Could It Double Capacity. London, UK: IET.
7. G. Prescher, T. G. (2005). A parametrizable low-power high-throughput turbo-decoder. IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Philadelphia, PA, USA: IEEE.
8. Camille Leroux, C. J. (2015). Parallel Architectures for Turbo Product Codes Decoding. Springer.
9. E. Boutillon, W. G. (2003). VLSI architectures for the MAP algorithm. IEEE Transactions on Communications, 175-185.
10. Mohammad M. Mansour, N. R. (2003). VLSI architectures for SISO-APP decoders. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 627-650.
11. H. Dawid, G. G. (1993). Map channel decoding: Algorithm and VLSI architecture. Proceedings of IEEE Workshop on VLSI Signal Processing. Veldhoven, Netherlands: IEEE.
12. Yoon, S., & Bar-Ness, Y. (2002). A parallel map algorithm for low latency turbo decoding. IEEE Communications Letters.
13. M.J. Thul, N. W. (2002). Enabling high-speed turbo-decoding through concurrent interleaving. 2002 IEEE International Symposium on Circuits and Systems. . Phoenix-Scottsdale, AZ, USA: IEEE.
14. C. Neeb, M. T. (2005). Network-on-chip centric approach to interleaving in high throughput channel decoders. 2005 IEEE International Symposium on Circuits and Systems (págs. 1766-1769). Kobe, Japan: IEEE.
15. G. Prescher, T. G. (2005). A parametrizable low-power high-throughput turbo-decoder. IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Philadelphia, PA, USA: IEEE.

16. Jaeyoung Kwak, S. M.-S. (2003). Implementation of a parallel turbo decoder with dividable interleaver. Proceedings of the 2003 International Symposium on Circuits and Systems. Bangkok, Thailand: IEEE.
17. L. Dinoi, S. B. (2005). Variable-size interleaver design for parallel turbo decoder architectures. IEEE Transactions on Communications . IEEE.
18. Claude Berrou, Y. S. (2004). Designing good permutations for turbo codes: towards a single model. IEEE International Conference Communitions. Paris, France: IEEE.
19. Takeshita, O. Y. (2006). On maximum contention-free interleavers and permutation polynomials over integer rings. IEEE Trans. Inf. Theory, 1249-1253.
20. Yang Sun, J. R. (2011). Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder. Integration the VLSI Journal, 305-315.
21. Trio Adiono, M. (2012). Radix-4 Max-log-MAP parallel turbo decoder architecture with a new cache memory data flow for LTE. International Symposium on Intelligent Signal Processing and Communications Systems (págs. 792-797). Taipei, Taiwan: IEEE.
22. Faxun Jin, J. T. (2008). A Radix-8 Log-MAP Recursion VLSI architecture. 11th IEEE International Conference on Communication Technology. Hangzhou, China: IEEE.
23. S. M. Karim, I. C. (2011). Design of efficient high throughput pipelined parallel turbo decoder using QPP interleaver. International Conference on Multimedia, Signal Processing and Communication Technologies (págs. 248-251). Aligarh, India: IEEE.
24. K.K. Loo, T. A. (2003). High performance parallelised 3GPP turbo decoder. 2003 5th European Personal Mobile Communications Conference. Glasgow, UK: IET.
25. Lennin C. Yllescas-Calderon, A. J.-O.-M.-P. (2011). Design and implementation of a simplified turbo decoder for 3gpp2. . 2011 International Conference on Reconfigurable Computing and FPGAs. Cancun Mexico: IEEE.
- An i, P. H. (2016). 1.5 Gbit/s FPGA Implementation of a Fully-Parallel Turbo Decoder Designed for Mission-Critical Machine-Type Communication Applications. IEEE Access, 5452-5473.
- Ashfaq Ahmed, M. A. (2011). A high throughput turbo decoder VLSI architecture for 3GPP LTE standard. IEEE 14th International Multitopic Conference (pags. 340-346). Karachi, Pakistan: 2011.
- Barco, M. I., & Gabriel N. Maggio, D. A. (2009). FPGA implementation of high-speed parallel maximum a posteriori (MAP) decoders. Argentine School of Micro-Nanoelectronics, Technology and Applications. San Carlos de Bariloche, Argentina: IEEE.
- Cheng-Chi Wong, H.-C. C. (2011). High-Efficiency Processing Schedule for Parallel Turbo Decoders Using QPP Interleaver. IEEE Transactions on Circuits and Systems I: Regular Papers (Volume: 58, 1412-1420.
- Christoph Studer, C. B. (2011). Design and implementation of a parallel turbo-decoder ASIC for 3GPP-LTE. IEEE Journal of Solid-State Circuits, 8-17.
- Dhiraj Reddy Nallapa Yoge, N. C. (2012). GPU Implementation of a Programmable Turbo Decoder for Software Defined Radio Applications. 25th International Conference on VLSI Design (pags. 149-153). Hyderabad, India: IEEE.
- Di Wu, R. A. (2009). Implementation of A High-Speed Parallel Turbo Decoder for 3GPP LTE Terminals. The 8th international Conference on ASIC (IEEE ASICON 2009) (págs. 481-484). 2017: Guiyang China.
- Guohui Wang, H. S. (2014). Parallel Interleaver Design for a High throughput HSPA+/LTE Multi-Standard Turbo Decoder. IEEE Transactions on Circuits and Systems I: Regular Papers.
- Injae Yoo, B. K.-C. (2014). Tail-overlapped SISO decoding for high throughput LTE-Advanced turbo decoders. IEEE Transactions on Circuits and Systems I: Regular Papers, 2711-2720.

Michael Wu, G. W. (2013). HSPA/LTE-A turbo decoder on GPU and multicore CPU. Asilomar Conference on Signals, Systems and Computers. Pacific Grove, CA: IEEE.

O. Muller, A. B. (2006). Exploring parallel processing levels for convolutional turbo decoding. 2nd International Conference on Information & Communication Technologies (págs. 2353-2358). Damascus, Syria: IEEE.

Rahul Shrestha, R. P. (2014). High-Throughput Turbo Decoder With Parallel Architecture for LTE Wireless Communication Standards. IEEE Transactions on Circuits and Systems I: Regular Papers.

C. Berrou, A. G. (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. Proceedings of ICC '93 - IEEE International Conference on Communications (págs. 1064-1070). Geneva, Switzerland: IEEE.

Wu, M., Sun, Y., & Cavallaro, J. R. (2010). Implementation of a 3GPP LTE turbo decoder accelerator on GPU. IEEE Workshop On Signal Processing Systems (págs. 192-197). San Francisco, CA: IEEE.

Xian, L., Brejza, M., Maunder, R. G., Al-Hashimi, B. M. & Hanzo, L. (2019). Arbitrarily Parallel Turbo Decoding for Ultra-Reliable Low Latency Communication in 3GPP LTE. IEEE Journal on Selected Areas in Communications.