J Sign Process Syst (2011) 65:185–197
DOI 10.1007/s11265-011-0604-z

Architecture and Finite Precision Optimization for Layered LDPC Decoders

Cédric Marchand · Laura Conde-Canencia · Emmanuel Boutillon

Received: 23 January 2011 / Revised: 14 June 2011 / Accepted: 14 June 2011 / Published online: 28 July 2011

Abstract Layered decoding is known to provide efficient and high-throughput implementation of LDPC decoders. However, two main issues affect performance and area of practical implementations: quantization and memory. Quantization can strongly degrade performance and memory area can constitute up to 70% of the total area of the decoder implementation. This is the case of the DVB-S2,-T2 and -C2 decoders when considering long frames. This paper is then dedicated to the optimization of these decoders. We first focus on the reduction of the number of quantization bits and propose solutions based on the efficient saturation of the channel values, the extrinsic messages and the a posteriori probabilities (APP). We reduce from 6 to 5 the number of quantization bits for the channel and the extrinsic messages and from 8 to 6 the APPs, without introducing any performance loss. We then consider the optimization of the size of the extrinsic memory considering a multiple code rates decoder. The paper finally presents an optimized fixed-point architecture of a DVB-S2 layered decoder and its implementation on an FPGA device.

Keywords Low-density parity-check (LDPC) code · Layered decoding · VLSI implementation · DVB-S2

C. Marchand (✉) · L. Conde-Canencia · E. Boutillon
Lab-STICC, CNRS UMR 3192,
Université de Bretagne Sud - UEB,
BP 92116 - 56 321 Lorient Cedex, France
e-mail: cedric.marchand@univ-ubs.fr

L. Conde-Canencia
e-mail: laura.conde-canencia@univ-ubs.fr

E. Boutillon
e-mail: emmanuel.boutillon@univ-ubs.fr

1 Introduction

Low-Density Parity-Check (LDPC) codes were initially proposed by Gallager in the early 60's [1], but they were not used for three decades. This was mainly because the technology was not mature enough for practical implementation. LDPC codes were rediscovered by MacKay [2] in 1995 and are now included in many standards. The existing standards can be categorized into two type of standards: the standards using short frames (648, 1296 and 1944 bits for Wi-Fi) and the standards using long frames (16200 and 64800 bits for DVB-S2). The use of long frames makes it possible to get closer to the Shannon limit, but leads to delays that are not suitable for internet protocols or mobile phone communications. On the other hand, long frames are suitable for streaming or Digital Video Broadcasting (DVB). The 2nd Generation Satellite Digital Video Broadcast (DVB-S2) standard was ratified in 2005, the 2nd Generation Terrestrial DVB (DVB-T2) standard was adopted in 2009 and the 2nd Generation Cable DVB (DVB-C2) was adopted during 2010. These three DVB standards include a common Forward Error Correction (FEC) block. The FEC is composed of an LDPC inner code and BCH outer code. The FEC supports eleven code rates for the DVB-S2 standard frames and six code rates for the DVB-T2 standard frames. In the following, DVB-X2 stands for DVB-S2, -T2, -C2. The LDPC codes defined by the DVB-X2 standards are structured codes or architecture-aware codes (AA-LDPC) [3] which can be efficiently implemented using the layered decoder architecture. The layered decoder benefits from three architecture improvements: parallelism of structured codes, turbo message passing, and Soft-Output (SO) based Node Processor (NP) [4–6].

Even if the state-of-the-art decoder architecture converges to the layered decoder solution, the search of an efficient trade-off between area, cost, low consumption, high throughput and high performance still make the implementation of the LDPC decoder a challenge. Furthermore, the designer has to deal with many possible choices of algorithm, parallelism, quantization parameters, code rates and frame lengths. In this article, we study the optimization of the layered decoders. We consider the DVB-S2 standard for comparison with previous designs in literature. However, our work can also be applied to the Wi-Fi and WiMAX LDPC standards, or more generally, to any layered LDPC decoder.

The well-known Min-Sum algorithm [7] and its variants significantly reduce the memory requirements thanks to the compression of the extrinsic messages. For this reason, we consider the Min-Sum algorithm for our study. We also consider the Sum-Product algorithm for comparison in terms of memory area.

The natural way to reduce the memory needs is to use a minimum number of bits to represent the data in the circuit. This can be done first by an appropriate scaling of the input symbol, second by an appropriate saturation of internal data. One should note that using a minimum number of bits to represent data also leads to a reduction in the complexity of the interconnection routing scheme, the area of the processing units and their associated critical path. In the state-of-the-art, saturation of the SO and the extrinsic messages is rarely explicitly explained. In this article, we will discuss efficient saturation of the channel values, the extrinsic messages and the SO values. We also present some ideas related to the efficient use of saturation which leads to significant memory savings. Finally, we introduce a methodology to optimize the implementation of the extrinsic memory under the constraints of 11 code rates and a single port RAM.

The paper is organized as follows: Section 2 presents the layered decoder and the Min-Sum sub-optimal algorithm. In Section 3, we explain the saturating process. Section 4 deals with the optimization of the size of the extrinsic memory. Finally, simulation and synthesis results are provided in Section 5.

2 LDPC Decoder and Layered LDPC Decoder

In this Section we first overview the principles of LDPC decoding and then focus on the layered decoding approach and architecture.

2.1 LDPC Code Decoding Principles

An LDPC code is defined by its parity-check matrix $\mathbf{H}$ of M rows by N columns. Each column in $\mathbf{H}$ is associated with one bit of the codeword, and each row corresponds to a parity check equation. A nonzero element in a row means that the corresponding bit contributes to the parity check equation. The set of valid code words $x \in C$ satisfy the equation:

$$x \cdot \mathbf{H}^t = 0, \forall x \in C \tag{1}$$

An LDPC decoder can be described by a Tanner graph [8] which is a graphical representation of the associations between code bits and parity-check equations. Code bits are represented by Variable Nodes (VN) and parity-check equations are represented by Check Nodes (CN), with edges connecting them accordingly to the parity check matrix. A message going from CN c to VN v is called $M_{c\to v}$, and a message going from VN v to CN c is called $M_{v\to c}$.

The decoding iterative process known as the Belief Propagation (BP) algorithm, was first introduced by Gallager [1] and rediscovered by MacKay [9]. This algorithm propagates probability messages to the nodes through the edges. These messages provide soft information about the state (0 or 1) of a VN. The Log-BP algorithm considers the probability messages in the log domain and are called Log Likelihood Ratios (LLR). The LLRs are defined as:

$$LLR_v = \log\left(\frac{P(v=0)}{P(v=1)}\right) \tag{2}$$

where $P(v = x)$ is the probability that bit v eguals x.

The order in which the nodes are updated is called the scheduling. The flooding schedule, first proposed by Gallager [1] consists in four steps as follows:

2.1.1 Initialization

Set all the $M_{v\to c}$ to the channel LLR values, i.e: $\forall(c, v)$ such that $\mathbf{H}(\mathbf{c}, \mathbf{v}) = 1$, $M_{v\to c} = LLR_v^{in}$.

2.1.2 CNs Update

The update of a node means that a node reads the incoming messages and then updates the outgoing messages. For all the CNs, update of the messages by applying the Bayes law in the logarithmic domain. For

implementation convenience, the sign and the absolute value of the $M_{c\to v}$ messages are updated separately:

$$sign\left(M_{c\to v}^{\text{new}}\right) = \prod_{v'\in v_c/v} sign\left(M_{v'\to c}\right) \tag{3}$$

$$|M_{c\to v}^{\text{new}}| = f\left(\sum_{v'\in v_c/v} f\left(|M_{v'\to c}|\right)\right) \tag{4}$$

where v_c is the set of all the VNs connected to CN c and v_c/v is v_c without v. The function $f(x)$ is expressed by the equation:

$$f(x) = -\ln\tanh\left(\frac{x}{2}\right) = \ln\frac{\exp x+1}{\exp x-1} \tag{5}$$

2.1.3 VNs Update

All the VNs are updated. The VN update is performed in two steps. First, equations the Soft Output (SO) value is computed as in Eq. 6 where the LLR_v^{in} value is the initial soft input from the channel. Then, the new $M_{v\to c}$ messages are computed as in Eq. 7.

$$SO_v = LLR_v^{in} + \sum_{c\in c_v} M_{c\to v} \tag{6}$$

$$M_{v\to c} = SO_v - M_{c\to v} \tag{7}$$

2.1.4 Hard Decision and Stopping Criteria

If a codeword is found or if the maximum number of iteration is reached, then the decoding process stops, else the iterative process continues in step two. At the end of the iterative process, a hard decision is made on the VN to output the codeword. The word estimation $\hat{y}$ is given by the hard decision of the SO: $\hat{y}_v = sign(SO_v)$, $v \in [0, ..., N]$ where $sign(x) = 0$ if $x > 0$ and $sign(x) = 1$ if $x < 0$. If $\hat{y}\cdot\mathbf{H}^t = 0$ then a codeword has been found.

From an implementation point of view, the flooding scheduling LDPC decoder is not optimized. Major improvements have been proposed in the literature, leading to the horizontal layered decoder which is currently the most efficient LDPC decoder architecture and can be applied to the DVB-X2 matrices.

2.2 Horizontal Layered Decoder

The layered decoder constitutes an optimized solution for LDPC decoding in terms of convergence speed, parallelism, latency, memory requirements and complexity. These are based on the following features:

2.2.1 The Turbo Message Passing Schedule [3, 10]

The CNs are processed one by one. The VNs connected to an updated CN are immediately updated with newly generated $M_{c\to v}$ messages. The next CNs will thus benefit from newly updated VNs which improves the convergence speed. Simulations show that for the DVB-S2 decoder with Eb/No leading to Quasi-Error Free (QEF) decoding, the number of decoding iterations is reduced by two.

2.2.2 Structured Matrices

The Group Horizontal Shuffle or Layered Horizontal Shuffle [5] follows the same scheduling principle as the turbo message passing, but instead of processing the CNs one by one, they are processed by groups. This is possible when the LDPC matrix is composed of shifted Identity Matrices (IM) of size P. Then a group of P CNs can be processed in parallel. The IEEE WiMAX standard [11] uses this matrix structure leading to efficient decoding architectures as the one presented in [4]. Figure 1 shows the structure of the rate-2/3 short-frame DVB-S2 LDPC parity check matrix. This structured matrix is composed of shifted IMs of size $P = 360$, allowing for parallel processing of up to 360 CNs.

2.2.3 SO Centric Decoder

Hereafter we present how the SO-centric NP architecture is deduced. The update of the VNs connected to a given CN is done serially in three steps. First, the message $M_{v\to c}$ is calculated using Eq. 8:

$$M_{v\to c} = SO_v - M_{c\to v}^{\text{old}} \tag{8}$$

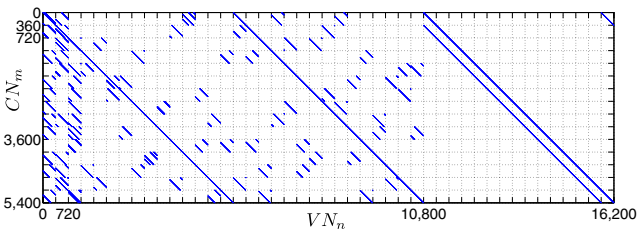

Figure 1 Block-structured rate-2/3 DVB-S2 matrix (N = 16200).

The second step is the serial $M_{c\to v}$ update as described in Eqs. 3 and 4. Finally, the third step is the calculation of the SO_{new} value:

$$SO_v^{\text{new}} = M_{v\to c} + M_{c\to v}^{\text{new}} \tag{9}$$

Because the $M_{v\to c}$ is calculated from the SO_v and $M_{c\to v}^{\text{old}}$ values, the SO-centric decoder does not need to save the $M_{v\to c}^{\text{old}}$ values, leading to memory saving.

2.3 Architecture Overview

From Eqs. 8 and 9, the NP architecture in Fig. 2 can be derived. The left adder of the architecture performs Eq. 8 and the right adder performs Eq. 9. The central part is in charge of the serial $M_{c\to v}$ update.

Several CNs may be grouped together to form a layer, whenever the column weights in the layer does not exceed one. The structured matrices made of shifted IM of size P allow us to compute layers made of P CNs. The layered decoder architecture is mainly based on P NPs that first read serially the Groups of P VNs (VNGs) linked to one Group of CN (CNG). Then after a given number of cycles ϵ, i.e. the NP latency, the P NPs write back the result to the VNGs in the same order.

Pipelining allows a more efficient use of the NP and an increase of the throughput [12–16]. The pipelining consists in reading the v_{c_i} of one sub-iteration while writing on $v_{c_{i-1}}$ the result of the previous sub-iteration. This means that as soon as the reading of one sub-iteration is finished, a new one is started. The corresponding maximum throughput is given by:

$$D = \frac{64800.F_{Clk}}{d_c.\frac{M}{P}.N_{it} + d_c + \epsilon + init} \quad bit.s^{-1} \tag{10}$$

where N_{it} is the number of iterations to decode a codeword, M is the number of CN, P is the number of NPs working in parallel, d_c is the average number of VNs linked to a CN, *init* is the number of cycle required to init the decoder, F_{clk} is the clock frequency and K is the number of information bits in a codeword.

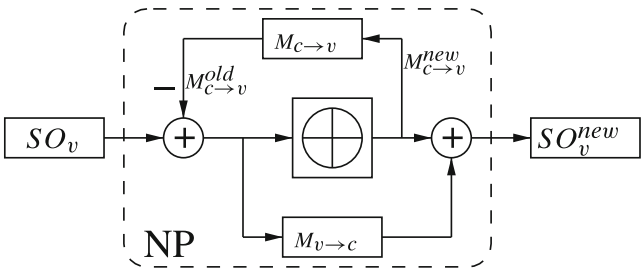

Figure 2 Check Node centric Node Processor.

The central part of Fig. 2 manages the serial $M_{c\to v}$ update described in Eq. 4. The $f(x)$ function shown in this equation is difficult to implement. This function can be implemented using look up tables or linear piecewise approximation [17], but can also be implemented more efficiently by using a sub-optimal algorithm as described in the following subsection.

2.4 The Normalized Min-Sum Algorithm and Other Related Algorithms

The most used sub-optimal algorithms are improved versions of the well-known Min-Sum algorithm [7] such as: the Normalized Min-Sum algorithm (NMS), the Offset Min-Sum (OMS) algorithm, the A-min* algorithm [18], the λ-min algorithm [19] and related [20]. The advantages of these algorithms are the simplified computation of Eq. 4 and the compression of the $M_{c\to v}$ messages. Although all these algorithms present different performances, the memory space they require to store the $M_{c\to v}$ messages is identical (except if $\lambda > 2$ for the λ-min algorithm). Hence, without loss of generality, for the rest of the paper, we will consider the NMS algorithm. With this algorithm, Eq. 4 becomes:

$$|M_{c\to v}^{\text{new}}| = \alpha \min_{v'\in v_c/v} |M_{v'\to c}| \tag{11}$$

where α is the normalization factor, $0 < \alpha \leq 1$.

The CN generates two different values: *Min* and *Submin*. The *Min* value is the normalized minimum of all the incoming $M_{v\to c}$ values and the *Submin* is the second normalized minimum. Let $Index_{\min}$ be the index of the minimum. For each $|M_{c\to v}^{\text{new}}|$ value, if the index of $M_{c\to v}^{\text{new}}$ is $Index_{\min}$ then $|M_{c\to v}^{\text{new}}| = submin$, else $|M_{c\to v}^{\text{new}}| = Min$. The $M_{c\to v}$ from one CN can be compressed into four elements, i.e. *Min*, *Submin*, $Index_{\min}$ and $sign(M_{c\to v}^{\text{new}})$. For matrices with a check node degree greater than four, this compression leads to significant memory savings.

3 Saturation

An SO value is the sum of the channel LLR with all the incoming extrinsic messages. Considering the case of the LDPC codes from the DVB-S2 standard, the maximum variable node degree (d_v) is 13. Even if the channel LLR and the $M_{c\to v}$ are quantized on 6 bits, the SO values must be quantized on 10 bits to prevent overflows. However, to avoid prohibitive word sizes, efficient saturation of the values can significantly reduce the size of the data.

3.1 Channel LLR Saturation

For floating point simulation, it is known that the decoders using the Normalized Min-Sum algorithm are not sensitive to scaling in the LLR^{in} values. During the initializing process, the equation $LLR^{in} = 2y/\sigma^2$ can be simplified to $LLR^{in} = y$, saving the need to compute the variance. The received y value is quantized in a way to have integer values at the input of the decoder. We assume here that the quantized value of y denoted by LLR_q is represented on n_{LLR} bits and that the quantification function is defined as:

$$LLR(y)_q = \left\lfloor sat(y, R) \times \frac{2^{n_{\text{LLR}}-1} - 1}{R} + 0.5 \right\rfloor, \tag{12}$$

where $sat(a, b) = a$ if a belongs to $[-b, b]$ and $sat(a, b) = sign(a) \times b$ otherwise. The R value is the interval range of quantization (y is quantized between $[-R, R]$) and also represents the saturation threshold value. Considering the BPSK modulation, we saturate y at $R = 1 + \beta$ where β is a positive factor defined later.

Figure 3 shows the Probability Density Function (PDF) of a quantized BPSK modulation (-1 and $+1$) that is perturbed by an Additive White Gaussian Noise (AWGN) of variance $\sigma = 0.866$ (corresponding to Eb/No = 2 dB). The channel is quantized on 5 bits and the saturation threshold is $R = 1 + \beta = 2.47$. The distribution filled in black (respectively unfilled) shows the received PDF distribution for the transmission of $+1$ (respectively -1). The quantized distribution varies from $LLR_q^{\min} = -(2^4 - 1)$ to $LLR_q^{\max} = 2^4 - 1$. The problematic is to find the saturation threshold providing the best performance for a given number of bits of quantization. If the saturation threshold is low, then the information given by the tail of the Gaussian curve is saturated. If the threshold is high, then the quantization step increases. The precision is then reduced and the Root Mean Square (RMS) value of the quantization error increases (proportionally with the quantization step).

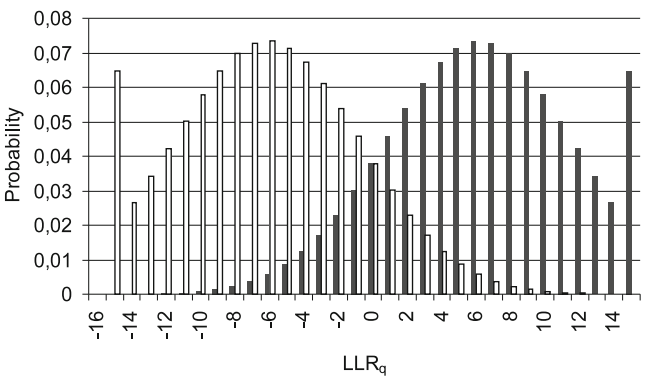

Figure 3 Resulting distribution of a quantized BPSK modulation.

The saturation limit $1 + \beta$ can be calculated so that the proportion of saturated values is equal to the average proportion of the other values. The average proportion of a given value is $1/(2^{n_{\text{LLR}}} - 1)$(probability of a value in a uniform distribution). On the other side of the equality, the Cumulate Distributive Function (CDF) of a -1 offset distribution applied to the negative saturation limit will give the proportion of saturated values for a -1 offset signal. The equality can be written as:

$$\frac{1}{2}\left[1 + erf\left(\frac{-\beta}{\sqrt{2}\sigma}\right)\right] = \frac{1}{2^{n_{\text{LLR}}} - 1} \tag{13}$$

From Eq. 13, β can be deduced:

$$\beta = \sigma \times \sqrt{2}\left(erf^{-1}\left(\frac{2^{n_{\text{LLR}}} - 1}{2^{n_{\text{LLR}}} + 1}\right)\right) \tag{14}$$

Thus the β value is a function of n_{LLR} and proportional to σ. By applying Eqs. 14 and 12, the optimum saturation threshold and the scaling factor can be computed. The problem of this solution is that an auto-scaling quantization of y is needed that requires a channel estimation of σ.

In fact, to prevent the σ computation, an optimal scaling factor is calculated for a given SNR. If the performance requirement is reached for a given SNR, then for a higher SNR, the quantization would be sub-optimal. However, even with sub-optimal quantization, with a higher SNR, the performance continues to improve. For each code rate, a constant saturation value $R = 1 + \beta$ can be pre-computed, thus saving the need to estimate the σ of the noise.

3.1.1 Effects of the Channel LLR Saturation on BER Performance

Figure 4 shows the simulation results for a normalized Min-Sum fixed-point layered decoder, with a maximum of 30 iterations, long frame, and code rate 2/3 in AWGN channel. The normalization factor is $\alpha = 0.75$. We consider the following notation: a 3-8-4 configuration refers to channel LLRs quantized on 3 bits, an SO value word size of 8 bits and a $M_{c\to v}$ word size of 4 bits. In Fig. 4 "Std limit" corresponds to the standard limit, which is set at 1 dB from the Shannon limit. The quantization values of the $M_{c\to v}$ messages and SO values are not optimized and are chosen to be large enough so that they

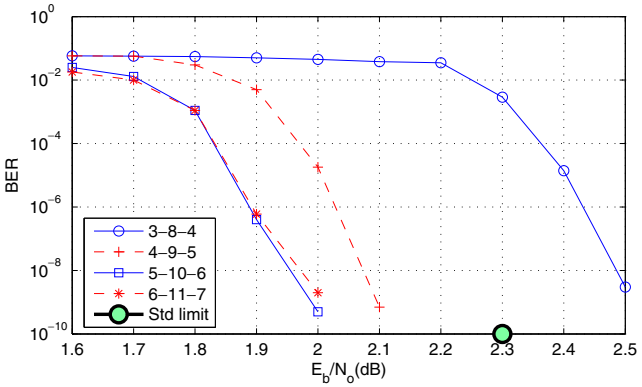

Figure 4 Effect of the channel LLR quantization on the BER performance. Simulation of a rate-2/3 long frame over the AWGN channel.

do not affect the results of the channel quantization. Fig. 4 shows that a quantization on 4 or 5 bits of the LLR^{in} is enough to fulfill the standard requirements.

3.2 SO Saturation

Once the LLR^{in} are quantized, they are stored in an SO memory that evolves with the iterative process. This SO memory needs to be saturated to limit its size.

3.2.1 The Problem of SO Saturation

Let us first consider the saturation case where $SO_{\max} < SO_v^{\text{new}}$ during the SO update (Eq. 9). The saturation process will bound SO_v^{new} to the $SO_{\max}$ value. This will introduce an error ϵ_v in the SO_v^{new} value ($\epsilon = SO_v^{\text{new}} - SO_{\max}$). During the next iteration, the new $M'_{v\to c}$ value will be $M'_{v\to c} = SO_v - M_{c\to v} = M_{v\to c} - \epsilon_v$.

Let us also consider the worst case: during an iteration, SO_v is saturated at $+SO_{\max}$, each CN confirms a positive $M_{c\to v}$ value, and $d_v = 13$ (i.e. SO_v is saturated 13 times). At the beginning of the next iteration, $SO_v = SO_{\max}$. From Eqs. 8 and 9, we can deduce that $SO_{\text{new}} = SO_{\text{old}} + \Delta M_{c\to v}$ where $\Delta M_{c\to v} = M_{c\to v}^{\text{new}} - M_{c\to v}^{\text{old}}$. If $\Delta M_{c\to v} < 0$, the SO value decreases. The SO value can even decrease 13 times and change its sign. To summarize, with the SO saturation, the SO value cannot increase, but it can decrease. The saturation introduces a non-linearity that can produce pseudo-codewords and an error floor. In the next section we propose a solution to overcome this problem.

3.2.2 A Solution for SO Saturation

The solution that we propose was first introduced in [21] and relies partially on the A Priory Probability (APP) based decoding algorithm [7]. The APP-variable decoding algorithm simplifies Eq. 8 to:

$$M_{v\to c} = SO_v \tag{15}$$

which greatly reduces the architecture complexity but introduces significant performance loss. The idea is to use Eq. 15 only when there is saturation. This leads to the APP-SO saturation algorithm, which is described as follows:

Algorithm 1 APP-SO saturation algorithm

if $SO_v = SO_{max}$ **then**
  $M_{v\to c} = SO_v$
else
  $M_{v\to c} = SO_v - M_{c\to v}$
end if

3.2.3 Effects of the SO Saturation on the BER Performance

Figure 5 shows the simulation results using the same conditions as in Fig. 4. An asterisk symbol in the legend means that the APP-SO algorithm is used. The results show that with the APP-SO algorithm it is possible to reduce the number of SO quantization bits from 7 to 6 without performance loss (compare curves 5-7-6 and 5-6-5*). However, without considering the APP-SO algorithm this one-bit reduction would lead to dramatic performance loss (curve 5-6-5).

3.3 Saturation of the Extrinsic Messages

Figure 6 shows the SO based node processor. The newly updated extrinsic $M_{c\to v}^{\text{new}}$ is used to compute SO_v^{new} from

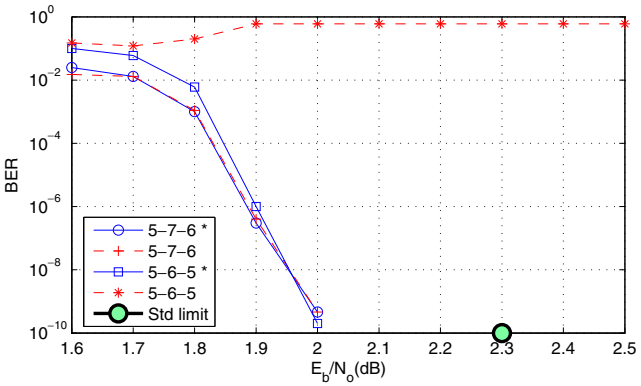

Figure 5 Effect of the channel LLR quantization on the BER performance. Simulation of a rate-2/3 long frame over the AWGN channel.

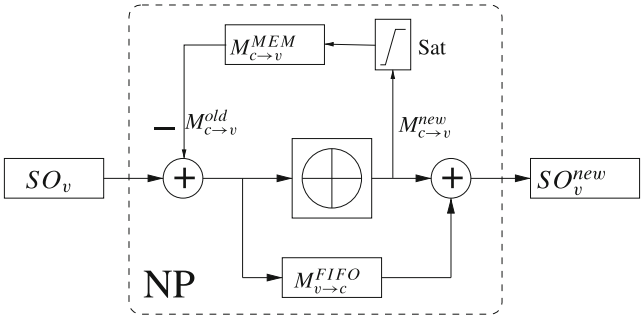

Figure 6 NP with saturation of the extrinsic messages.

Eq. 9. $M_{c\to v}^{\text{new}}$ is also stored in the extrinsic memory for the calculation of $M_{v\to c}$ (Eq. 8) at the next iteration. Any saturation on the value $M_{c\to v}^{\text{new}}$ responsible for the SO update would not produce area savings and would degrade performance. This is the reason why we do not saturate this value. On the other hand, saturation of the $M_{c\to v}^{\text{new}}$ messages that are stored in a memory would lead to significant area savings. Furthermore, the saturation of the $M_{c\to v}^{\text{new}}$ stored in the extrinsic memory is much less critical because it will be used only once during an iteration to compute an $M_{v\to c}$. Furthermore, this computed $M_{v\to c}$ will affect SO^{new} only if it is equal to the minimum value or second minimum value of the incoming $M_{v'\to c}$ of check node c (see Eq. 11).

3.3.1 Effects of the Extrinsic Message Saturation on the BER Performance

Figure 7 shows the simulation results using the same conditions as in Fig. 4. Using the same number of bits for the LLR and the extrinsic messages quantization, i.e., $n_{\text{LLR}} = n_{ext} = 5$, results in performance results

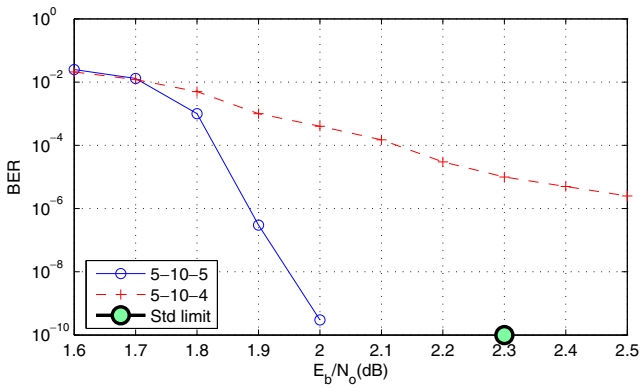

Figure 7 Effect of the extrinsic message saturation on the BER performance. Simulation of a rate-2/3 long frame over the AWGN channel.

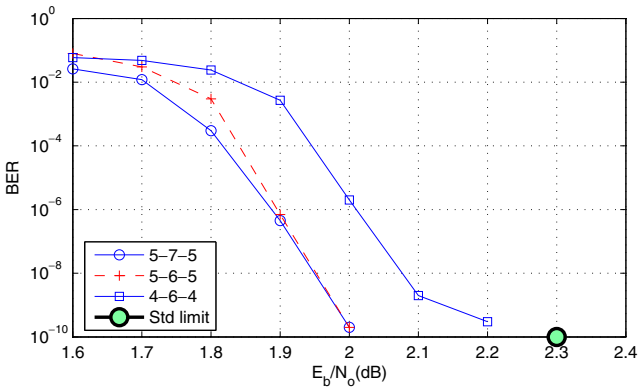

Figure 8 Effect of the SO and the extrinsic message saturation on the BER performance. Simulation of a rate-2/3 long frame over the AWGN channel.

within the standard requirement. However, considering $n_{ext} < n_{\text{LLR}} = 5$, leads to dramatic performance loss.

3.4 Combining the SO and the Extrinsic Saturation Processes

Figure 8 shows to simulation results when combining the APP-SO saturation algorithm with the extrinsic saturation. These results show that the quantization and saturation solution we propose makes it possible to reduce the 6-8-6 configuration to a 4-6-4 configuration that respects the standard requirements. Considering a 5-6-5 configuration leads to performance gains of about 0.1 or 0.2 dB, compared to the 4-6-4 configuration.

3.5 Conclusion on the Saturation Optimization

The analysis of the saturation process shows that a better trade-off between word size and performance can be obtained with an efficient saturation of the LLR^{in}, the SO and the extrinsic values. Simulations show the robustness of our saturation solutions allowing for the use of fewer bits than the usual 6-8-6 configuration. To be specific, the 5-6-5 configuration leads to quantization bit saving while providing the same performance as any other known implementation.

4 Optimizing the Size of the Extrinsic Memory

The decoder requirements in terms of extrinsic memory size strongly depend on the coding rate. In this section we focus on the design of an optimal decoder implementation that supports all the DVB-S2 standard code rates.

Table 1 Memory size of extrinsic messages as a function of the DVB-S2 code rates when implementing the Min-Sum algorithm.

Rate	M	W_{Sign}	W_{Index}	$W_{M_{c\to v}}$	Memory
1/4	48,600	4	2	14	680,400
1/3	43,200	5	3	16	691,200
2/5	38,880	6	3	17	660,960
1/2	32,400	7	3	18	583,200
3/5	25,920	11	4	23	596,160
2/3	21,600	10	4	22	475,200
3/4	16,200	14	4	26	421,200
4/5	12,960	18	5	31	401,760
5/6	10,800	22	5	35	378,000
8/9	7,200	27	5	40	288,000
9/10	6,480	30	5	43	278,640

4.1 Memory Size

The memory requirements of each CN is determined by the $M^{\text{old}}_{c\to v}$ messages needed for the CN computation. In the case of the normalized Min-Sum algorithm [7], the $M^{\text{old}}_{c\to v}$ values are compressed with *Min*, *Submin*, $index_{\min}$ and $Sign(M_{c\to v})$. In terms of memory, one address must be allocated for every CN, which means that the RAM address range (R_{RAM}) is given by the number of CNs (M). The RAM word size (W_{RAM}) is given by the size of the compressed $M^{\text{old}}_{c\to v}$ values. If we denote by W_h the word size of h, then $W_{Mc\to v} = W_{|\text{Min}|} + W_{|\text{Submin}|} + W_{\text{Index}} + W_{\text{Sign}}$. Table 1 presents the required memory capacity ($M \times W_{M_{c\to v}}$) for each rate. To calculate $W_{M_{c\to v}}$, we give $W_{|\text{Min}|}$ and $W_{|\text{Submin}|}$ a constant value of 4. To handle the eleven code rates of the standard, a simple implementation would define R_{RAM} with the maximum M value, and W_{RAM} with the maximum $W_{Mc\to v}$ in Table 1. The total memory capacity would give: $48600 \times 43 = 2089800$ bits. For rate 1/4, 67% of word bits are wasted but addresses are fully used. On the other hand, for rate 9/10, word bits are fully used but 86% of the addresses are wasted. Theoretically, a memory size of 691200 bits would be enough to cover all the rates. A solution needs to be found for a better utilization of the memory.

4.2 Optimization Principle

The idea is to add flexibility to both the address range and the word size. For this, we benefit from the fact that the RAM that stores the compressed $M^{\text{old}}_{c\to v}$ value is needed only once per layer. The delay to compute the next layer is of d_c cycles, so we can use up to d_c cycles to fetch the data in the memory. A word can be split into two if we take two cycles to fetch the data, and split in three if we take three cycles. If we consider a single port RAM to implement the memory, up to $\lfloor d_c/2 \rfloor$ cycles can be used to read data, and $\lfloor d_c/2 \rfloor$ cycles to write new data.

Let us consider the example of a memory bank of size $48600(R_{\text{RAM}}) \times 23(W_{\text{RAM}})$. In a first configuration, where one cycle is used, we have a memory size of 48600×23. This first configuration fits the rates 1/4, 1/3, 2/5, 1/2, 3/5, and 2/3. In a second configuration, where two cycles are used and two words of size 23 are fetched at consecutive addresses, we have the equivalent of a memory of size 24300×46 which fits the rates 3/4, 4/5, 5/6, 8/9 and 9/10. The total memory size for the two-cycle option is equal to $48600 \times 23 = 111780$ bits. This constitutes a memory saving of 47% compared to the straightforward implementation.

4.3 Results

The previously described process can be used for different word sizes. Table 2 gives an example with $W_{\text{RAM}} = 9$. For each rate, the number of cycles is given by:

$$n_{\text{cycles}} = \lceil W_{Mc\to v} / W_{\text{RAM}} \rceil$$

The RAM range for a code rate R_{RAM} is deduced from $R_{\text{RAM}} = n_{\text{cycles}} \times M$. The global RAM range ($R^{\text{global}}_{\text{RAM}}$) is given by the maximum R_{RAM} in Table 2 and the total memory capacity is $R^{\text{global}}_{\text{RAM}} \times W_{\text{RAM}} = 97200 \times 9 = 874800$ bits.

Figure 9 shows the total memory capacity as a function of the word length W_{RAM}. There are local minima for word sizes 1, 9, 14, 18 and 21 bits. As the number of clock cycle to fetch $M^{\text{old}}_{c\to v}$ is bounded by $\lfloor d_c/2 \rfloor$, the possible solutions are limited to W_{RAM} values greater than 7. A word size of 9 bits gives the best memory optimization of 874800 bits. This is only 26% more than the theoretical minimum memory size.

Table 2 Memory capacity of the extrinsic message with $W_{\text{RAM}} = 9$.

Rate	M	$W_{Mc\to v}$	n_{cycles}	R_{RAM}
1/4	48,600	14	2	97,200
1/3	43,200	16	2	86,400
2/5	38,880	17	2	77,760
1/2	32,400	18	2	64,800
3/5	25,920	23	3	77,760
2/3	21,600	22	3	64,800
3/4	16,200	26	3	48,600
4/5	12,960	31	4	51,840
5/6	10,800	35	4	43,220
8/9	7,200	40	5	36,000
9/10	6,480	43	5	32,400

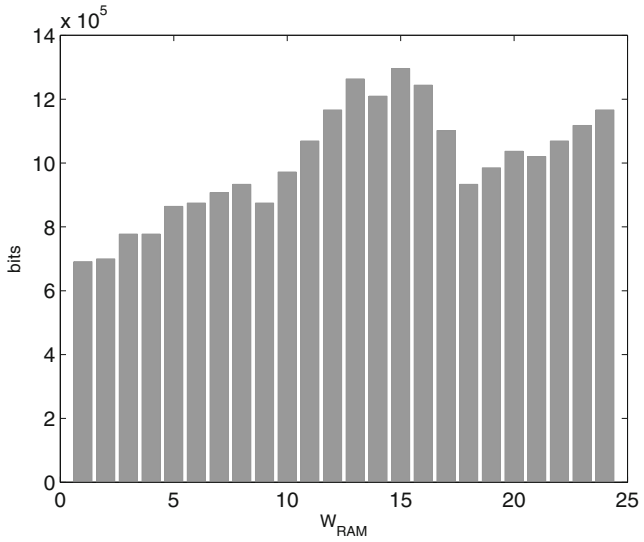

Figure 9 Memory capacity as a function of W_{RAM}.

4.4 Case of the Sum-Product Algorithm

When using a Sum-Product algorithm [22–24] instead of a Min-Sum algorithm, the CN update Eq. 4 is computed for each $M_{c\to v}$ value and then each $M_{c\to v}$ value is stored. The process described in Section 4.2 can be used, but a simpler and more efficient implementation is also possible. We consider the architecture overview of Fig. 2 with a dual port RAM that stores the $M_{c\to v}$ values. At every cycle, a $M_{c\to v}^{\mathrm{new}}$ value is stored and a $M_{c\to v}^{\mathrm{old}}$ value is read from the dual port RAM simultaneously. The address range of this memory is then determined by the number of $M_{c\to v}$ values. With the constraint of 11 code rates, 5 bits quantization and dual port RAMs, the memory requirement is determined by the code rate with the maximum number of $M_{c\to v}$ values. The code rate 5/6 requires storing 237600 values of 5 bits, which results in 1.2 Mbits. Although this size is 37% higher than the solution that we proposed for the Min-Sum algorithm, the additional cost can be worth it, due to the performance increase especially at low code rates. The implementation of a FIFO memory with single port RAM for allowing simultaneous read and write operations is presented in [25]. This solution requires one memory bank for even addresses and another memory bank for odd addresses. This design leads to area savings when compared to the dual port RAM solution.

4.5 Conclusion on the Extrinsic Memory Optimization

The $M_{c\to v}$ memory optimization becomes an issue, especially when the LDPC decoder supports multiple code rates with single port RAMs. A careful implementation of the Min-Sum algorithm results in only 26% more than the theoretical minimum. A straightforward implementation would require 200% more memory over the theoretical minimum. The Sum-Product algorithm leads to a 37% over cost compared to the Min-Sum algorithm, but presents better performance, especially for low code rates. The implementation of the Sum-Product algorithm can then be a reasonable solution.

5 Finite Precision Architecture of the Layered Decoder

Figure 10 presents the finite precision architecture of the NP (Fig. 2). Word size, type (signed or absolute) and direction are detailed for each signal connection. The architecture implements the normalized Min-Sum algorithm.

In the CN core, the $M_{v\to c}$ values arrive serially in a two's complement representation from the adder modified to implement Algorithm 1 (see Section 3.2.2). The values are transformed to sign and magnitude representation, so that the sign and magnitude of the messages can be computed separately. The serial sorting of the incoming magnitude values is implemented, in order to give *Min*, *SubMin* and *Index* values until all the incoming messages are read. In the serial $M_{c\to v}$ block, the previously sorted *Min*, *SubMin* and *Index*

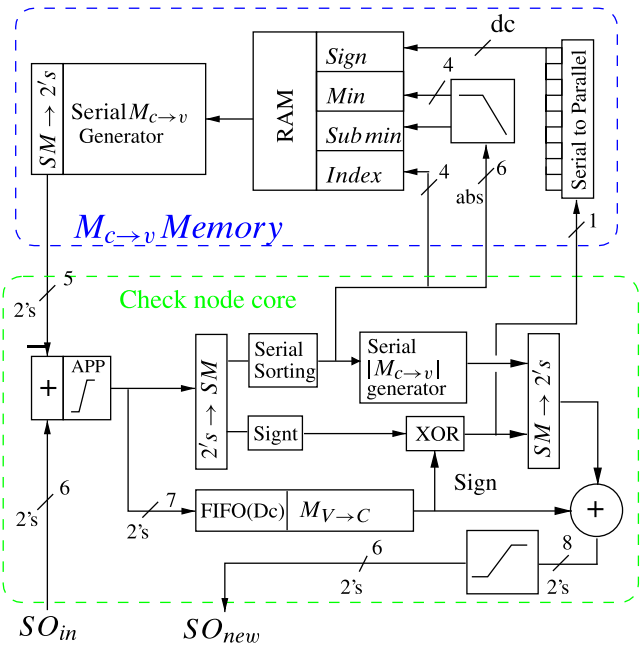

Figure 10 Finite precision architecture of the Node Processor.

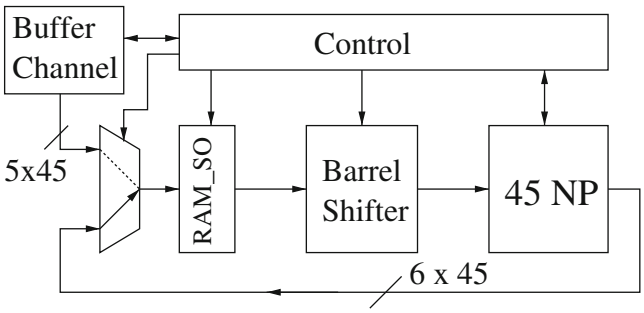

Figure 11 Layered decoder architecture.

values are used to serially compute the new outgoing messages. An XOR function is applied recursively on the incoming sign bits in the *Signt*, until all the incoming messages are read. Next, a XOR block computes an XOR function between the output of the *Signt* block and the sign of the outgoing message $M_{v\to c}$ from the FIFO memory of size d_c. The sign and magnitude, from the XOR block and the serial $M_{c\to v}$ block, respectively, are transformed into a two's complement representation ready for subsequent computations.

In the $M_{c\to v}$ memory block, the *Min*, *SubMin*, *index* and *sign* of each $M_{c\to v}$ linked to one check node are stored in a RAM. From the *Min*, *Submin*, *Index* and *Sign* values, the serial $M_{c\to v}$ generator computes the two's complement representation of the $M_{c\to v}$ value. *Min* and *SubMin* are quantized on 4 bits (absolute values) which gives $M_{c\to v}$ values quantized on 5 bits (with the sign bit). Note that *Signt* represents the result of the parity check equation of one CN. The syndrome can easily be computed by using the result of *Signt* for each CN. The computation of the syndrome allows for the decision of an early termination of the decoding process that can significantly reduce the number of iterations.

Figure 11 is an overview of the proposed layered decoder architecture (see [26] and [27] for a more detailed description). In this figure, the NP block is made of 45 NPs (Fig. 10) working in parallel. The Barrel Shifter shifts six 45-bit words. The RAM_SO block stores the SO values. Due to the systematic syndrome calculation, it is possible to use a built-in stopping rule for a variable-iteration decoder. The addition of a buffer on the decoder input allows for the exploitation of the decoding time variations of the different frames. A preemptive buffer control, described in [28], is used to reduce the buffer size. Note that the saturation and quantization solutions described in Section 3 for memory area saving also leads to area reductions of the NP and the barrel shifter. The latency in the CN core is also reduced due to the complexity reduction of the addition and comparison hardware.

The conflicts due to pipelining [26] and the conflicts due to the DVB-X2 matrices structure [27] are not in the scope of the article, however the descripted architecture can run conflict-free without modification. The conflict due to pipelining are solved by layer scheduling [26] and the conflict due to the matrices structure is solved by an improved method of [27] which involve only the control. The methode is based on layer duplication and on time write disable of the memory and is actually under a patent process.

6 Results

6.1 Comparison of Error Performances

Figure 12 shows simulation results for code rates 1/4, 1/2, 2/3, 3/4, 4/5 and 5/6 with the 5-6-5 configuration. The code rates 2/3, 3/4, 4/5 and 5/6 fulfill the standard requirements. The code rate 1/4 shows poor performance. In fact, the probability that the normalisation factor α gets close to the optimal value decreases with the CN degree. The code rates 1/4, 1/3 and 2/5 have a CN degree smaller than 7 (4, 5 and 6 respectively), which leads to an error floor when using the Normalized Min-Sum algorithm. Note that a code rate of 1/2, with a CN degree of 7, produces an error floor that can be corrected with the BCH outer decoder. The error floor produced by low code rates can be solved by implementing an A-min* or λ-min algorithm instead of the normalized Min-Sum in the NP, with no change in the rest of the architecture. It is also possible to implement a Sum-Product algorithm, described in [17], combined with the proposed saturation processes. The rates 2/3, 3/4, 4/5 and 5/6 show performance at less than 0.1dB from the ideal error performance requirement defined in [29]. Other recent implementations [20, 30–32]

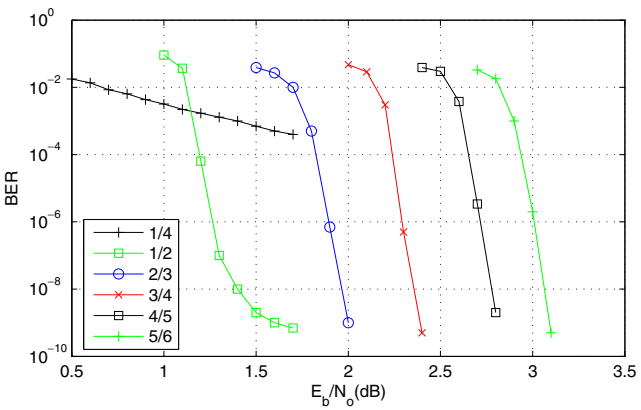

Figure 12 BER for long frames.

Table 3 Synthesis results for the DVB-S2 LDPC decoder.

XQ5VLX85	LUT	LUT RAM	BRAM
Node processor	143	2	0
Sorting	37	0	0
Gen $m_{c\to v}$	34	0	0
Fifo $m_{v\to c}$	12	2	0
$m_{c\to v}$ memory	56	0	2
Total 1 node	199	2	2
Total 45 nodes	8,955	90	90
Control	787	3	0
Block SO RAM	360	0	22
Channel RAM	48	0	25
Barrel shifter	945	0	0
Total	11,305	93	137
Percentage (%)	6	1	38

also fulfill the standard requirement which allow a fair comparison of other parameters.

6.2 Implementation Results

The architecture presented in Fig. 11 was synthesized on a Virtex-V Pro FPGA (XQ5VLX110) from Xilinx for validation purposes. The system decodes all code rates with a parallelism of 45. The parallelism of 45 is obtained by a reordering of the matrix [26, 27]. Table 3 gives the hardware resources required. The average number of iterations is 15 and the clock frequency is reduced from 300 to 200 Mhz to obtain an average air throughput of 135 Mbps as in [20, 30, 32]. For the $M_{c\to v}$ memory, to balance complexity of the control and the memory saving, we considered the option with its two configurations as described in Section 4.2 and also implemented in [31].

The state-of-the-art provides results for different technologies, architectures and sub-optimal algorithm implementations which make comparison difficult. However, the number of NPs and the required memory capacity are fair points of comparison to highlight our contributions.

The implementation of a layered decoder for the DVB-S2 standard was considered in [20, 30, 31] and [32]. Table 4 considers these implementations to compare the parallelism, the air throughput and signals width. The parallelism provides information on the number of implemented NPs and an indication on the shuffling network complexity. Compared with other architectures, the parallelism reduction solution also employed in [13, 33] allows significant reduction of the barrel shifter area and the number of NPs. The air throughput is in all cases higher or equal to the standard requirement defined at 135 Mbps. The extrinsic and SO_{ram} word width directly impacts on the memory size and the NP complexity. Note that no information is provided on the ROM memories that store the matrices for every rate. The main contribution of our architecture resides in the reduction of the signals width. Channel and extrinsic quantization are reduced from 6 to 5 bits and SO width from 8 to 6 bits. In our architecture, a buffer of size two is added to store the channel LLR values to halve the average number of iterations as described in [28]. The memory capacity is decomposed as follow: 64800×6 bits for the SO memories, 48600×23 bits for the $M_{c\to v}$ memories, $64800 \times 5 \times 2$ bits for the buffer and $30 \times 7 \times 45$ bits for the $M_{v\to c}$ memories. Because of the buffer we require more capacity than [31] but still less than other implementations.

7 Conclusion

In this paper, we have presented optimization solutions for a layered LDPC decoder. Our first approach was to analyze the saturation problem in the layered decoder. An efficient saturation leads mainly to a reduction of memory area and also a reduction of latency in the computing elements. We developed a finite precision layered decoder architecture that implements the proposed saturation solution. This architecture

Table 4 Layered decoder implementation comparison.

Paper	Yan et al. [30]	Botao et al. [31]	Muller et al. [20]	Urard et al. [32]	This
Parallelism	360	360	180	180	45
Algorithm	NMS	OMS	3-min	–	NMS
Throughput (Mbps)	135	1000	135	135	135
Frequency (MHz)	105	320	105	105	200
Extrinsic (bits)	6	6	6	6	5
SO_{ram} (bits)	8	8	10	8	6
Channel (bits)	6	6	6	6	5
Buffer	Yes	No	–	–	Yes
Capacity (Mbits)	2.3	1.68	2.68	3.18	2.2
BCH	No	No	Yes	Yes	No
Technologie (nm)	130	90	90	65	65

outperforms the state-of-the-art in terms of memory needs while satisfying the standard requirements in terms of performance. In our second approach, we studied the problem of implementing an efficient multi-code-rate decoder. Our solution relied on the word split of the extrinsic memory for the Min-Sum algorithm. This solution allows for the use of single port RAMs and leads to significant memory reduction. Even though we have only considered the DVB-S2 standard in our study, the proposed techniques can be extended to DVB-T2, -C2 and to any layered LDPC decoder. Future work will be dedicated to optimize the hardware implementation (area and frequency) of the proposed decoder architecture and to the evaluation of its performance at low BER.

Acknowledgements The authors thank NXP Semiconductors Caen for the funding of the study, the "Région Bretagne" and the "European Funds for Regional Development"(FEDER) for funding materials used in the study.

References

1. Gallager, R. (1963). Low-density parity-check codes. PhD thesis, Cambridge.
2. MacKay, D. (1999). Good error-correcting codes based on very sparse matrices. *Transactions on information theory IEEE, 45*, 399–431.
3. Mansour, M. M., & Shanbhag, N. R. (2003). High-throughput LDPC decoders. *IEEE transactions on very large scale integration VLSI systems, 11*, 976–996.
4. Brack, T., Alles, M., Kienle, F., & Wehn, N. (2006). A synthesizable IP core for WIMAX 802.16e LDPC code decoding. In *2006 IEEE 17th international symposium on personal, indoor and mobile radio communications* (pp. 1–5). Helsinki, Finland.
5. Rovini, M., Rossi, F., Ciao, P., L'Insalata, N., & Fanucci, L. (2006). Layered decoding of non-layered LDPC codes. In *9th EUROMICRO conference on digital system design: Architectures, methods and tools, 2006. DSD 2006* (pp. 537–544). Dubrovnick, Croatia.
6. Hocevar, D. (2004). A reduced complexity decoder architecture via layered decoding of LDPC codes. In *IEEE workshop on signal processing systems, 2004. SIPS 2004* (pp. 107–112). Austin, USA.
7. Fossorier, M., Mihaljevic, M., & Imai, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications, 47*, 673–680.
8. R. Tanner (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory, 27*, 533–547.
9. MacKay, D., & Neal, R. (1997). Near shannon limit performance of low density parity check codes. *Electronics Letters, 33*, 457–458.
10. Mansour, M., & Shanbhag, N. (2002). Low-power VLSI decoder architectures for LDPC codes. In *Proceedings of the 2002 international symposium on low power electronics and design, 2002. ISLPED '02* (pp. 284–289). Monterey, USA.
11. I. std (2005). Air interface for fixed and mobile broadband wireless access systems. In *P802.16e/D12 Draft* (pp. 100–105). Washington, DC, USA: IEEE.
12. Sun, Y., Karkooti, M., & Cavallaro, J. (2006). High throughput, parallel, scalable LDPC encoder/decoder architecture for OFDM systems. In *2006 IEEE Dallas/CAS workshop on design, applications, integration and software* (pp. 39–42). Richarson, USA.
13. Dielissen, J., Hekstra, A., & Berg, V. (2006). Low cost LDPC decoder for DVB-S2. In *Proceedings of design, automation and test in Europe, 2006. DATE '06* (Vol. 2, pp. 1–6). Munich, Germany.
14. Segard, A., Verdier, F., Declercq, D., & Urard, P. (2006). A DVB-S2 compliant LDPC decoder integrating the horizontal shuffle schedule. In *IEEE international symposium on intelligent signal processing and communication systems (ISPACS 2006)*. Tottori, Japan.
15. Bhatt, T., Sundaramurthy, V., Stolpman, V., & McCain, D. (2006). Pipelined block-serial decoder architecture for structured LDPC codes. In *IEEE international conference on acoustics, speech and signal processing, 2006. ICASSP 2006 proceedings* (Vol. 4, p. IV). Toulouse, France.
16. Rovini, M., Gentile, G., Rossi, F., & Fanucci, L. (2007). A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes. In *IFIP international conference on very large scale integration, 2007. VLSI—SoC 2007* (pp. 236–241). Atlanta, USA.
17. Hu, X.-Y., Eleftheriou, E., Arnold, D.-M., & Dholakia, A. (2001). Efficient implementations of the sum-product algorithm for decoding LDPC codes. In *Global telecommunications conference, 2001. GLOBECOM '01* (Vol. 2, pp. 1036–1036E). IEEE.
18. Jones, C., Valles, E., Smith, M., & Villasenor, J. (2003). Approximate-min* constraint node updating for LDPC code decoding. In *IEEE military communication conference* (pp. 157–162).
19. Guilloud, F., Boutillon, E., & Danger, J.-L. (2003). Lambda-min decoding algorithm of regular and irregular LDPC codes. In *Proceedings of the 3rd international symposium on turbo codes and related topics*.
20. Muller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., & Wehn, N. (2009). A novel LDPC decoder for DVB-S2 IP. In *Design, automation & test in Europe conference & exhibition, 2009. DATE'09*. Nice, France.
21. Doré, J. (2007). *Optimisation conjointe des codes LDPC et de leurs architecture de décodage et mise en oeuvre sur FPGA*. PhD thesis, INSA, Rennes, France.
22. Papaharalabos, S., & Mathiopoulos, P. (2009). Simplified sum-product algorithm for decoding LDPC codes with optimale performance. *Electronics Letters, 45*, 536–539.
23. Gones, M., Falcao, G., Goncalves, J., Silva, V., Falcao, M., & Faia, P. (2006). HDL library of processing units for generic and DVB-S2 LDPC decoding. In *International conference on signal processing and multimedia applications (SIGMAP2006)*. Setubal, Portugal.
24. Eljamaly, O., & Sweeney, P. (2007). Alternative approximation of check node algorithm for DVB-S2 LDPC decoder. In *Second international conference on systems and networks communications (ICSNC 2007)* (pp. 157–162).
25. Andreev, A., Bolotov, A., & Scepanovic, R. (2007). *Fifo memory with single port memory modules for allowing simultaneous read and write operations*. US patent 7181563.
26. Marchand, C., Doré, J.-B., Conde-Canencia, L., & Boutillon, E. (2009). Conflict resolution for pipelined layered LDPC decoders. In: *IEEE workshop on signal processing systems, 2009. SiPS 2009* (pp. 220–225). Tampere, Finlande.

27. Marchand, C., Doré, J.-B., Conde-Canencia, L., & Boutillon, E. (2009). Conflict resolution by matrix reordering for DVB-T2 LDPC decoders. In *Global telecommunications conference, 2009. GLOBECOM 2009* (pp. 1–6). Honolulu, USA: IEEE.
28. Rovini, M., & Martinez, A., (2007). On the addition of an input buffer to an iterative decoder for LDPC codes. In *IEEE 65th vehicular technologie conference, VTC2007* (pp. 1995–1999). Dublin, Ireland.
29. D. V. B. (DVB) (2009). Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2). *European Standard (Telecommunications series) ETSI EN 302 307 V1.2.1 (2009-08)*.
30. Yan, Y., Dan, B., Shuangqu, H. Bo, X., Yun, C., & Xiaoyang, Z. (2009). A cost efficient LDPC decoder for DVB-S2. In *IEEE 8th international conference on ASIC, 2009. ASICON '09* (pp. 1007–1010). Changsa, China.
31. Botao, Z., Hengzu, L., Xucan, C., Dongpei, L., & Xiaofei, Y. (2009). Low complexity DVB-S2 LDPC decoder. In *Vehicular technologie conference (VTC2009)* (pp. 1–5). Setubal, Portugal.
32. Urard, P., Paumier, L., Heinrich, V., Raina, N., & Chawla, N. (2008). A 360mw 105b/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes enabling satellite—transmission portble devices. In *Solid-state circuits conference, 2008. ISSCC 2008. Digest of technical papers* (pp. 310–311). IEEE International: San Francisco, USA.
33. Gomes, M., Falcao, G., Silva, V., Ferreira, V., Sengo, A., & Falcao, M. (2007). Flexible parallel architecture for DVB-S2 LDPC decoders. In *Global telecommunications conference, 2007. GLOBECOM '07* (pp. 3265–3269). Washington, USA: IEEE.

Cédric Marchand was born in 1976, France. He received the B.E. degree in electrical and electronics engineering from the North East Wales Institute, Wrexhan, Wales, 1999, M.Sc. and Ph.D. degrees in electrical engineering from the Université Européenne de Bretagne in 2007 and 2011 respectively. From 2007 to 2011, he has been working with NXP Semiconductor France on the implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards. His current research interests include error correcting code decoder and VLSI design.

Laura Conde-Canencia received her M.Sc. degree from Universidad Politecnica de Madrid, Spain, in 2000, and Ph.D. from Telecom Bretagne, Brest, France, in 2004. Her Ph.D. thesis dealt with spectral-efficiency maximization in high-performance digital communication systems. In 2004, she joined Universidad San Pablo-CEU, in Madrid (Spain), as an Assistant Professor in Telecommunication Engineering. In 2006, she joined the Lab-STICC Laboratory in Université de Bretagne Sud (Brittany, France), where she is currently an Associate Professor. Her research interests include Advanced Channel Coding such as Turbo-Codes, LDPC Codes and Cortex codes, focussing on decoder designs.

Emmanuel Boutillon was born in 1966, France. He received the Engineering Diploma from the Ecole Nationale Supérieure des Telecommunications (ENST), Paris, France in 1990. In 1991, he worked as assistant professor in the Ecole Multinationale Supérieure des Télécommunications in Dakar (Senegal). In 1992, he joined ENST as research engineer where he had conducted research in the field of VLSI for digital communications. While he was working as engineer, he obtained his Ph.D in 1995 from ENST. In 1998, he spent a sabbatical year at the University of Toronto, Ontario, Canada. In 2000, he joined the Lab-STICC laboratory (Université de Bretagne Sud, Lorient, France) as Professor. His current research interests deal with the interactions between algorithm and architecture in the field of wireless communications. In particular, he works on Turbo Codes and LDPC decoders.