A Novel Conflict-Free Memory and Processor Architecture for DVB-T2 LDPC Decoding

Alberto Jiménez-Pacheco
Laboratoire de Communications Mobiles
School of Computer and Communication Sciences
EPFL Lausanne, Switzerland
Email: alberto.jimenez@epfl.ch

Onkar Dabeer
School of Technology & Computer Science
Tata Institute of Fundamental Research
Mumbai, India
Email: onkar@tifr.res.in

***Abstract*—In this paper, we present a flexible architecture for an LDPC decoder that fully exploits the structure of the codes defined in the DVB-T2 standard (Digital Video Broadcasting - Second Generation Terrestrial). We propose a processor and memory architecture which uses the flooding schedule and has no memory access conflicts, which are encountered in serial schedule decoders proposed in the literature. Thus, unlike previous works, we do not require any extra logic or *ad hoc* designs to resolve memory conflicts. Despite the typically slower convergence of flooding schedule compared to serial schedule decoders, our architecture meets the throughput and BER requirements specified in the DVB-T2 standard.**

Our design allows a trade-off between memory size and performance by the selection of the number of bits per message without affecting the general memory arrangement. Besides, our architecture is not algorithm specific: any check-node message processing algorithm can be used (Sum-Product, Min-Sum, etc.) without modifying the basic architecture. Furthermore, by simply adding relevant small ROM tables, we get a decoder that is fully compatible with all three second generation DVB standards (DVB-T2, DVB-S2 and DVB-C2). We present simulation results to demonstrate the viability of our solution both functionally and in terms of the bit-error rate performance. We also discuss the memory requirements and the throughput of the architecture, and present preliminary synthesis results in CMOS 130nm technology.

I. Introduction

Low-Density Parity Check (LDPC) codes were first introduced by Gallager in his Ph.D. thesis [1]. However, they truly captured the attention of the coding community only after the advent of turbo codes [2], and by now they have been extensively studied (see [3] and references therein). In particular, random ensembles of LDPC codes in conjunction with belief propagation (BP) decoding have been demonstrated to operate very close to the Shannon limit in a variety of channels [3]. Consequently, LDPC codes have been included in several recent communication standards: IEEE 802.11n, IEEE 802.16, DVB-T2, DVB-S2, etc.

The own nature of the BP algorithm lends itself to highly parallelizable decoding structures. But while random designs lead to powerful LDPC codes, the hardware implementation of their decoders, especially for long block lengths, still poses a considerable challenge in terms of chip area, throughput and routing complexity. Such implementation considerations force engineers to design codes with additional structure. In this paper, we exploit the structure in the LDPC codes adopted for DVB-T2 to propose and study a semi-parallel processing architecture for the decoder. The key aspects of our design are the absence of memory conflicts and its great flexibility. In the remainder of this section, we describe prior art and our contributions in light of these works.

A. Prior Art

Motivated by hardware constraints in the implementation of decoders, many authors have considered LDPC code designs that are decoder architecture-aware [4], [5]. This design philosophy is also reflected in the LDPC codes adopted for IEEE 802.11n-2009 (WiFi), IEEE 802.16e (WiMAX) and for the family of new digital video broadcasting standards: DVB-T2 (Terrestrial), DVB-S2 (Satellite) and DVB-C2 (Cable).

The architecture-aware codes adopted for DVB-T2 belong to the family of Irregular Repeat-Accumulate (IRA) codes [6], [7], which implies that their encoding can be done efficiently with linear-time complexity. Besides, to simplify decoding, the particular codes selected for DVB-T2 exhibit additional structure: the parity check matrix can be decomposed into blocks that exhibit a block-circulant structure. This structure has been exploited by several authors in different ways: the works [8], [9] propose a parallel architecture for the decoder by splitting the check nodes and variable nodes into groups. But they do not exploit the full parallelism in the DVB-T2 codes and consequently they need extra logic to resolve memory conflicts (which are instances where different processors attempt to access the same memory locations). Other solutions include the block-serial approach in [10] and the layered (Gauss-Seidel) decoding method in [11], [12]. Many of these works provide few details of their memory (RAM) arrangement. To the best of knowledge, they all seem to lead to memory conflicts and use some *ad hoc* techniques such as extra logic, or exploit specifics of the serial schedule and Min-Sum algorithm, to resolve conflicts. All these aspects lead to reduced flexibility for the engineer: the implementor cannot change the choice of the algorithm for improving performance and the implementor cannot easily trade off memory and accuracy.

978-963-8111-77-7

B. Our Contribution

In this paper, we exploit the structure in the DVB-T2 LDPC codes to design a parallel processor and memory architecture for the decoder that is conflict-free. While earlier designs also make use of the structure of the parity-check matrices, they do not identify and exploit it fully, and consequently our design is much cleaner and flexible. The key to our architecture is the identification of special sets of edges called *strands*, which lead to a partitioning of the Tanner graph of the DVB-T2 codes. Our architecture arranges memory in accordance with the strands and the parallel processors execute the message computations using node groups (already identified in earlier works [8]). The special properties of the strands and the node groups ensure that there are no memory conflicts, while still using single port RAM for the storage of the messages. Thus, the greater cost, area and power consumption associated to dual port RAM memories is also avoided.

Since our architecture is clean and conflict-free, it is flexible. First, by choosing the number of bits per message, we can trade off the overall size of the memory with the desired decoding performance. Second, even if changes are made to the draft specification codes (without altering the basic structure), the architecture remains the same except for the entries in an associated small ROM. Third, the engineer can pick his choice of the decoding algorithm: Sum-Product, Min-Sum, or any other variant.

We use the flooding schedule for computation, which in principle requires more iterations for convergence than the serial schedule [13]. But since we exploit the full structure in the code, we meet the throughput requirements in the DVB-T2 standard. Even though we have a higher number of parallel processors (360 versus 180 or 90 in [9], for instance), the overall chip area of our design is comparable with earlier works. This is due to the fact that the major contribution to the synthesized area comes from the RAM memory, while that of the processing units is much smaller. Besides, we do not incur any extra area due to additional logic to resolve memory access conflicts, since our architecture is conflict-free.

The rest of the paper is organized as follows: in Section II we describe in detail the structure and properties of the parity-check matrices that define the LDPC codes adopted in DVB-T2. We exploit this structure and propose our decoder architecture in Section III. In Section IV we discuss the sequence of decoding operations and explain why we do not have memory conflicts. In Section V we present simulated performance results using our decoder and discuss first synthesis results. We summarize our conclusions in Section VI.

II. The Structure of DVB-T2 LDPC Codes

An LDPC code is completely specified by its $(N-K)\times N$ binary sparse parity-check matrix $\boldsymbol{H}$ (incidence matrix of the Tanner graph [3]), where N is the length of the codeword (64800 or 16200 bits in DVB-T2) and K is the number of information bits. The code rate $R := K/N$ in DVB-T2 can range from $1/4$ to $5/6$. The rows of $\boldsymbol{H}$ correspond to check-nodes (CNs) and the columns correspond to the variable nodes (VNs). Each 1 in the parity-check matrix corresponds to an edge connecting a CN and a VN in the Tanner graph.

The LDPC codes in DVB-T2 are systematic, meaning that VNs $\{0, 1, \ldots, K-1\}$ correspond the information bits and are referred to as information nodes (INs); the VNs $\{K, K+1, \ldots, N-1\}$ denote the parity bits and are referred to as parity nodes (PNs).

For IRA codes we can write $\boldsymbol{H} = [\boldsymbol{H}^1 \ \boldsymbol{H}^2]$, where $\boldsymbol{H}^2$ is an $(N-K)\times(N-K)$ lower triangular matrix with 1s in its main and first lower diagonals, all other elements being 0.

The structure of matrix $\boldsymbol{H}^1$ is a bit more complex to specify. It is a sparse pseudo-random matrix designed with some periodicity constraints. Let $M = 360$, we define the code specific parameters $Q := (N-K)/M$ and $\hat{Q} := K/M$. Information nodes can be divided into $\hat{Q}$ groups of M nodes each, group q consisting of the INs $\{M\,q, M\,q+1, \ldots, M\,q+M-1\}$, $q \in \{0, 1, \ldots, \hat{Q}-1\}$. We can thus write

$$\boldsymbol{H}^1 = \left[\boldsymbol{H}^1_0, \ldots, \boldsymbol{H}^1_q, \ldots, \boldsymbol{H}^1_{\hat{Q}-1}\right], \tag{1}$$

where the $(N-K)\times M$ submatrix $\boldsymbol{H}^1_q$ specifies the connections between the CNs and IN group q. The matrix $\boldsymbol{H}^1_q$ has a block circulant structure:

$$\boldsymbol{H}^1_q = \begin{bmatrix} \boldsymbol{x}_{q,0} & \boldsymbol{x}_{q,M-1} & \cdots & \boldsymbol{x}_{q,1} \\ \boldsymbol{x}_{q,1} & \boldsymbol{x}_{q,0} & \cdots & \boldsymbol{x}_{q,2} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{x}_{q,M-1} & \boldsymbol{x}_{q,M-2} & \cdots & \boldsymbol{x}_{q,0} \end{bmatrix}, \tag{2}$$

where each $\boldsymbol{x}_{q,j}$ is a $Q\times 1$ vector. We see that column $j \in \{0, 1, \ldots, M-1\}$ is a circular downward shift of column 0 by jQ positions. Thus INs in group q all have the same degree (number of 1s in the column) and we denote it by d^I_q. We note that by design, for each code defined in DVB-T2, d^I_q

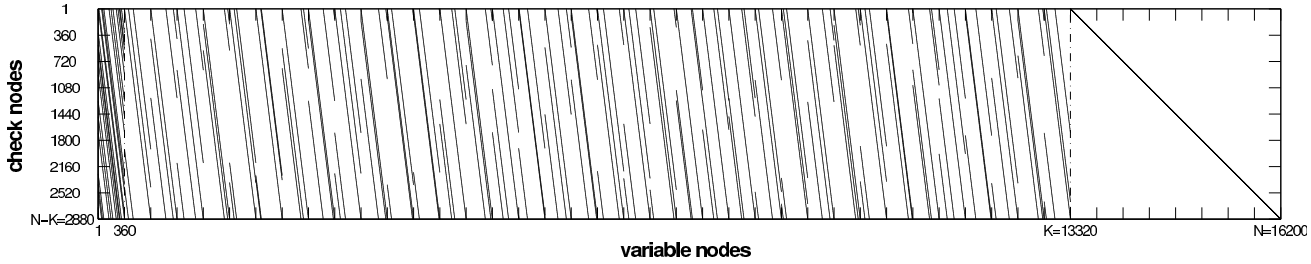

Fig. 1. Image of the parity-check matrix of the $(N = 16200, R = 5/6)$ DVB-T2 code (color black corresponds to 1 elements, white to 0 elements).

TABLE I
PARAMETERS FOR THE LDPC CODES DEFINED IN DVB-T2 ($N = 64800$)

Rate	Q	$\hat{Q}$	w	g_w	$\bar{d}_i$	$\bar{d}_c$	d_c^*	N_s
1/2	90	90	8	36	5	7	5	450
3/5	72	108	12	36	6	11	9	648
2/3	60	120	13	12	4	10	8	480
3/4	45	135	12	15	4	14	12	540
4/5	36	144	11	18	4	18	16	576
5/6	30	150	13	15	4	22	20	600

$\bar{d}_c$ is the average CN degree.
d_c^* is the row weight of submatrix $\boldsymbol{H}^1$.
The rest of the symbols are introduced in Section II.

TABLE II
PARAMETERS FOR THE LDPC CODES DEFINED IN DVB-T2 ($N = 16200$)

Rate	Q	$\hat{Q}$	w	g_w	$\bar{d}_i$	$\bar{d}_c$	d_c^*	N_s
1/4	36	9	12	4	5	7	2†	63
1/2	25	20	8	5	5	4.25	5†	85
3/5	18	27	12	5	6	4.67	7	126
2/3	15	30	13	3	4	4	8	120
3/4	12	33	12	1	4	3.27	11†	108
4/5	10	35	–	0	4	3	11†	105
5/6	8	37	13	1	4	3.27	17†	121

† Except for rates $\frac{3}{5}$ and $\frac{2}{3}$, the row weight of $\boldsymbol{H}^1$ is not constant; the value given here is the maximum row weight of matrix $\boldsymbol{H}^1$.

can only take two different degrees: a rate-dependent value w ($8 \le w \le 13$), or 3; and that these appear in two ordered sets: INs groups $\{0, 1, \ldots, g_w - 1\}$ have degree w, and INs groups $\{g_w, g_w + 1, \ldots, \hat{Q} - 1\}$ have degree 3 (g_w is a rate-dependent value as well, see Tables I and II).

The block circulant structure implies that the 1s in $\boldsymbol{H}_q^1$ appear in positions

$$\big((r_d(q) + j\,Q) \mod (N-K), j\big), \quad 1 \le d \le d_q^I,\ 0 \le j \le M-1, \tag{3}$$

where the values of $r_d(q)$ denote the rows in the first column of $\boldsymbol{H}_q^1$ where there are 1s, and these positions are specified in the tables of [14, Annexes A, B].

As can be seen in the example of Fig. 1, Eq. (3) implies that matrices $\boldsymbol{H}_q^1$ only have 1s along d_q^I diagonals with slope Q, circularly wrapped around. We refer to these diagonals as *strands*, and they are central to arranging the memory to ensure that there are no conflicts. More precisely a strand $S_{q,d}$ is a set of edges defined as follows:

$$S_{q,d} := \Big\{ \big(r_d(q) + j\,Q) \mod (N-K), j\big) : \quad j = 0, 1, \ldots, M-1 \Big\},\ d = 1, \ldots, d_q^I;\ q = 0, 1, \ldots, \hat{Q} - 1. \tag{4}$$

Each strand has M edges, two different strands do not intersect and their union gives the location of all 1s in $\boldsymbol{H}^1$. The total number of strands is $N_s = \hat{Q}\bar{d}_i$, where $\bar{d}_i$ is the average IN degree,

$$\bar{d}_i = \frac{g_w\, w + 3\, g_3}{\hat{Q}}, \tag{5}$$

and $\hat{Q} = g_w + g_3$ is the number of IN groups.

It will be later convenient to list the strands in the following order: $S_{0,0}$, $S_{0,1}$, …, S_{0,d_0^I-1}, $S_{1,0}$, $S_{1,1}$, …, S_{1,d_1^I-1},…, $S_{\hat{Q}-1,0}$, $S_{\hat{Q}-1,1}$, …, $S_{\hat{Q}-1,d_{\hat{Q}-1}^I-1}$. We number the strands from 0 to $N_s - 1$ in this order, and as per this numbering convention, the "CN–IN group q" matrix $\boldsymbol{H}_q^1$ consists of strands $(d_0^I + d_1^I + \ldots + d_{q-1}^I)$,…, $(d_0^I + d_1^I + \ldots + d_{q-1}^I + d_q^I - 1)$. Furthermore, we number the edges within one strand from 0 to $M - 1$ in the following order: for a strand connected to IN group q, 0 corresponds to the element of the strand in the column $M\,q$, 1 corresponds to the element of the strand in column $M\,q + 1$, and so forth.

The circulant structure of $\boldsymbol{H}_q^1$ also leads to a natural grouping of the rows: we define CN group q, $q \in \{1, \ldots, Q\}$, as the set consisting of the M CNs $\{q + j\,Q,\ j = 0, 1, \ldots, M-1\}$, since we see from (2) that the j-th row in the group can be obtained from the first one by a circular shift to the right by j positions. Similar to the "CN–IN group q" matrix, we can partition the edges of the "CN group q – IN" matrix using strands. If strand s belongs to CN group q, then we define the *offset* of this strand to be the index $\in \{0, 1, \ldots, M-1\}$ of the element in strand s that is connected to the first CN of the group (*i.e.*, connected to CN q) [1].

Finally, we define the PN group q as the set of PNs $\{q + j\,Q,\ j = 0, 1, \ldots, M-1\}$, $q \in \{0, \ldots, Q-1\}$. Each PN $m \in \{0, 1, \ldots, N-K-1\}$ is connected to CN m and $m+1$ (except for $m = N-K-1$, which is only connected to CN $N-K-1$); we call these respectively the *forward* and *backward* connections, from the perspective of the CNs.

III. PROPOSED ARCHITECTURE

The core ideas of our architecture are that 1) nodes within one group (as defined in Section II) are processed in parallel, and node groups are dealt with serially; and 2) memory is arranged such that the messages required in a strand form a word in RAM. Consequently, messages are all read/written in parallel from/to the memory without conflicts. A schematic of the semi-parallel architecture we propose is shown in Fig. 2, consisting of the following components:

- A ROM memory that stores the tables with the parameters required for the description of the codes.

[1] We see that by our numbering convention, for INs the strands always have zero offset.

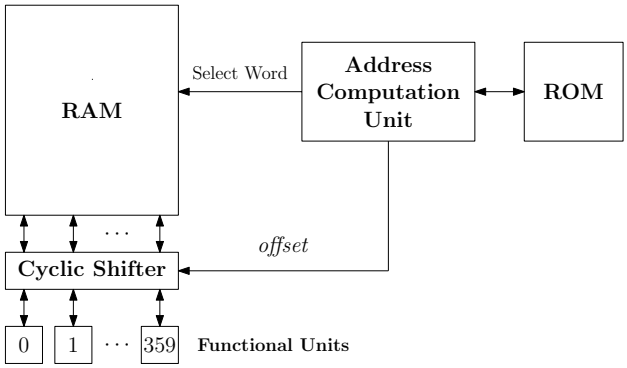

Fig. 2. Proposed decoder architecture.

- A RAM memory that stores the extrinsic messages exchanged between CNs and VNs during belief propagation, as well as the input log-likelihood ratios (LLRs) obtained by the soft de-mapper (the receiver stage immediately preceding the LDPC decoder [14]).
- A cyclic shifter that circularly shifts messages to the left (right) while reading (writing) from (to) the RAM by the specified *offset*, so that the elements of a memory row are aligned with the elements of the node group we are processing. The shifter holds M messages and can shift them in both directions by any offset between 0 and $M-1$ elements. When the offset is zero (as for all INs, for instance), data is exchanged directly between the RAM and the Functional Units (FUs).
- An Address Computation Unit (ACU), which aided by the ROM tables selects the memory row to be read/written as well as the offset to be used by the cyclic shifter. The addresses and offsets computed during the read phase are stored locally in the ACU and reused during the writing phase.
- $M = 360$ Functional Units that implement the functionality of an IN, PN and CN, and that work in parallel, updating all the BP messages corresponding to a node group at the same time. When IN group q is being processed, FU p executes the operations of IN $M\,q + p$, while when CN/PN group q is processed FU p executes the operations of CN/PN $q + p\,Q$. A functional unit reads messages on the incident edges and computes an outgoing message for each of the incident edges, according to the chosen variant of the BP algorithm. Each processing unit has enough local memory to store all incoming messages (maximum of 22, as per the last row of Table I) and computes all outgoing messages in parallel.
  Our architecture is independent of the specific algorithm used for computation at the CNs. Hence, the internal arithmetic inside the FUs will vary depending on the chosen variant of the belief propagation algorithm [15]: Sum-Product, Min-Sum, λ-Min, etc. Due to space constraints, we do not report about our implementation of the FUs here; these will be reported in a subsequent expanded journal submission.

A. RAM Memory

Next we describe the organization of the RAM in more detail. A single-port RAM is used, and we divide it into two blocks: one to store the input LLRs and another one to store edge messages passed around during belief propagation. Our decoder uses the flooding schedule. Hence, the second block holds the CN $\rightarrow$ VN messages after the first half of each decoding iteration, and the VN $\rightarrow$ CN messages after the second half.

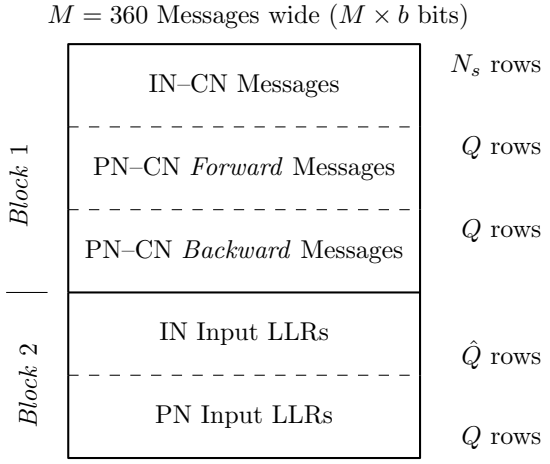

Fig. 3. RAM organization.

Messages are stored in RAM in two's complement format. If we denote by b the number of bits used to represent each message, then the width of the memory is $M\,b$ bits and its depth is arranged as follows (see Fig. 3).

- The first block, holding the extrinsic messages, consists of three parts:
  - A block of N_s rows holds the IN–CN messages. Strand i, $i = 0, 1, \ldots, N_s - 1$, is stored in row i, where the p-th element in the strand occupies bits $\{p\,b,\ p\,b+1,\ \ldots,\ (p+1)b-1\}$, $p = 0, 1, \ldots, M-1$.
  - The next Q rows hold the messages for the *forward* connection of CNs: row $N_s + q$ correspond to the *forward* connection of CN group q, $q = 0, 1, \ldots, Q-1$. Thus bits $\{p\,b,\ p\,b+1,\ \ldots,\ (p+1)b-1\}$ correspond to the message between CN $q + p\,Q$ and PN $q + p\,Q$.
  - The next Q rows hold the messages for the *backward* connection of the CNs: row $N_s + Q + q$ correspond to the *backward* connection of CN group q, $q = 0, 1, \ldots, Q-1$. Thus bits $\{p\,b,\ p\,b+1,\ \ldots,\ (p+1)b-1\}$ correspond to the message between CN $q + p\,Q$ and PN $q + p\,Q - 1$.
- The second block, holding the input LLRs, is divided in two parts:
  - The input LLRs corresponding to IN group q are stored in row $N_s + 2Q + q$, $q = 0, 1, \ldots, \hat{Q} - 1$.

– The input LLRs corresponding to PN group q are stored in row $N_s + 2Q + \hat{Q} + q$, $q = 0, 1, \ldots, Q-1$.

In the last decoding iteration the contents of this block of memory are overwritten with the final *a-posteriori* LLRs, and hard decisions can be obtained based on their sign bit.

The total RAM depth is thus given by $N_s + 3Q + \hat{Q}$. With the parameters from Tables I and II we check that the RAM requirement is dictated by the $(N = 64800, R = 3/5)$ code and evaluates to a maximum depth of 972 words. If we assume $b = 4$ bits per message, the total RAM requirement for our decoder is $972 \times M \times b \approx 1.4$ Mbit.

B. ROM Memory

For each block length and code rate we need to store in ROM the parameters that describe the parity-check matrix of the code. From the point of view of INs it is enough to store the degree w for those groups for which it is different from 3, and the IN group index g_w starting at which IN groups have degree 3. The strand numbers can be immediately computed given those two parameters and the IN group number q. Since the *offset* is always 0 for INs, it does not need to be stored.

From the point of view of the CNs, for each of the Q CN groups we need to store its degree[2] d_q^C, and two lists of d_q^C elements each: one to store the strand numbers and another one for the corresponding offsets.

All in all, for the description of the 6 long block-length and the 7 short block-length LDPC codes defined in DVB-T2 the total ROM requirements amount to 75 Kbit.

IV. Sequence of decoding operations

During *initialization*, the input LLRs computed by the soft de-mapper are written to RAM (in its second block). Since in the first iteration of belief propagation INs and PNs simply transfer these initial LLRs to their adjacent CNs, we also include this copy as part of the initialization, and appropriately fill in the first block of the RAM as well.

Next, we proceed with the iterative decoding. Each iteration consist of a first half in which CN $\rightarrow$ VN messages are computed and a second half in which VN $\rightarrow$ CN messages are computed. Each iteration begins by sequentially processing the Q groups of CNs, one group at a time. For each group, the ACU reads the ROM and computes the strands to be read, and the offset (if any) for the cyclic shifter. These addresses and offsets are stored locally in the ACU to be reused during the writing phase. The RAM is then read one strand at a time. If the offset is non-zero, then the word contents are circularly shifted to the left and then written to the FUs' internal memory. Then the FUs compute the outgoing messages. If the offset is non-zero, then the word contents are circularly shifted to the right before being written to the RAM (using the addresses stored locally in the ACU). All address computation, read/write, message computation and cyclic shift operations are pipelined.

The second half iteration processes first the $\hat{Q}$ groups of INs and then the Q groups of PNs in a similar way, but naturally the calculations carried out by the FUs to compute the VN $\rightarrow$ CN messages are different from those carried out during the first half iteration to compute the CN $\rightarrow$ VN messages.

If the maximum number of iterations has been reached (or some other stop criterion is met), then this second half of the iteration is different: instead of computing extrinsic messages, the FUs calculate the final *a-posteriori* LLRs, which are stored in the second RAM block, overwriting the input LLRs which are not necessary any more.

We note that the RAM is read/written one strand at a time. Also, for any node group, all the incident edges in a strand correspond to successive nodes in the group. As a consequence of these facts, we do not have memory conflicts. Our arrangement of the memory in accordance with the strands is a key component not found in earlier works, that makes our architecture clean, conflict-free and flexible.

V. Results

Although we defer to a subsequent journal publication the formal proof of the correctness of our decoding scheme and of the absence of memory conflicts, we provide some simulation results that functionally validate our architecture and asses its performance. To this aim we constructed three different LDPC software decoders:

- The first one is a completely generic floating-point decoder that implements the Min-Sum and Sum-Product variants of the BP algorithm. It can work with any LDPC code, without relying in any specific properties of the parity-check matrix.
- The second simulator is also a floating-point decoder, but its memory arrangement and sequence of decoding operations accurately follow the architecture we proposed, reflecting the limited hardware resources described in Section III. Thus, it is tailored to the specific properties of the LDPC codes defined in DVB-T2.
- The third software decoder is also architecture-aware, as the second one, but it is fully implemented in fix-point (the messages are stored in RAM in fix-point format and the operations inside the FUs are also carried out with finite precision).

1) Functional validation: The functional validation of the proposed architecture was carried out by comparing the outputs of the two first software decoders when fed with the same inputs, and this for both code lengths and for all code rates, under different channel conditions. As a double check we also compared our two floating-point decoders with the generic LDPC decoder implementation provided in Matlab's Communication Toolbox (which unfortunately implements the Sum-Product algorithm, but not the Min-Sum).

2) Fix-point performance validation: Once the functional validation had been passed, we moved on to validate the performance of the fix-point implementation of our architecture.

[2]For $N = 64800$ all codes have constant row weight, so only one value per code rate needs to be stored; this is not the case for $N = 16200$, where for every code rate the degree of each CN group needs to be stored.

Our fix-point software simulator aims to be bit-accurate with the Verilog implementation. In fact, we used it to generate input/output test vectors with which to validate the Verilog implementation of the decoder.

We next show simulation results for our decoder using two different algorithms and quantization strategies for CN message processing: the Sum-Product algorithm with $b = 5$ bits per message, and the Min-Sum algorithm with $b = 4$. The results shown in Fig. 4 have been obtained integrating our bit-accurate fix-point decoder into a complete simulator of the whole DVB-T2 transmit/receive chain, and using an AWGN channel model.

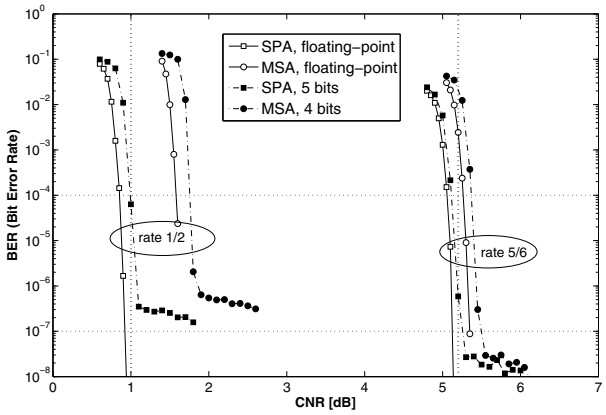

Fig. 4. Bit Error Rate (BER) at the output of the LDPC decoder in the DVB-T2 receiving chain. AWGN channel, QPSK modulation, 50 decoding iterations. SPA stands for Sum-Product Algorithm, MSA for Min-Sum Algorithm.

These results have been obtained under the same conditions as the reference results in the implementation guidelines published by the DVB consortium [16, Ch. 14], and are therefore immediately comparable. In fact, the vertical dotted lines shown in Fig. 4 correspond to the target carrier-to-noise ratios (CNR) for which a bit-error rate (BER) of 10^{-4} should be obtained using an optimal floating-point decoder. Hence we see that our fix-point decoder using the Sum-Product algorithm performs within 0.2 dB of the ideal floating-point decoder. We observe the appearance of error floors due to the quantization of the messages, but they are well below 10^{-6} and they will be absorbed by the BCH decoder that comes right after LDPC decoding in the DVB-T2 receiving chain [14]. The Min-Sum implementation is less sensitive to quantization effects, even using 4 bits instead of 5 to encode the messages: error floors are below 10^{-7} and the shift of the BER curves with respect to floating-point performance is around 0.1 dB. Of course, overall performance is worse due to its sub-optimality compared to the Sum-Product algorithm.

Due to space constraints, we have presented a limited set of results for illustration, but these results are representative of the behavior of our decoder for other code rates, modulation types and channel models.

A preliminary synthesis of the proposed decoder has been carried out in CMOS 130nm technology, using a clock frequency of 200 MHz. The target throughput for the LDPC decoder derived from the DVB-T2 standard is 60.4 Mbps. Assuming a fix number of iterations equal to 50 (value used in the reference results of [16]), with the clock-frequency of 200 MHz, we reach a maximum throughput of 59 Mbps for the $(N = 64800,\, R = 3/5)$ combination, which turns out to be the limiting case. For all other codes, the maximum achievable throughput with 50 iterations is well above the required target. Reducing the number of iterations from 50 to 45 is enough to meet the throughput requirement in the limiting case, while the effect on performance is minimal.

VI. Conclusions

We have presented a flexible architecture for an LDPC decoder for the DVB-T2 system, which can be easily extended to support DVB-S2 and DVB-C2 as well. We use the flooding schedule and propose a semi-parallel architecture which uses 360 processing units. We identify a particular way to partition the parity check-matrices in node groups and in strands of 360 elements each. By arranging the memory so that a word in RAM corresponds to a strand we completely avoid the memory access conflicts present in previous solutions, and hence we do not require any extra logic or *ad hoc* solutions to handle these exceptional circumstances. The clean structure offers great flexibility to the designer: new codes can be added by simple addition of small ROM tables and it is easy to trade off performance and memory usage without any modifications to the RAM arrangement or general decoding structure. Besides, our proposal is not tailored to any particular variant of the BP algorithm, and any check-node processing algorithm can be employed without further redesign. We have shown some simulations and preliminary synthesis results that proof the viability of our solution, both from a functional point of view and performance-wise: throughput and decoding performance requirements specified in the DVB-T2 standard and implementation guidelines are satisfied by our decoder.

Acknowledgment

The authors would like to thank Prof. Rüdiger Urbanke and Prof. Bixio Rimoldi at EPFL, and Dr. Mauro Lattuada, Dr. Matteo Buttusi and Pascal Peguet at Abilis Systems for numerous valuable discussions.

References

[1] R. Gallager, "Low-density parity-check codes," Ph.D. dissertation, MIT Press, Cambridge, Massachussets, Jul. 1963.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: turbo-codes," in *IEEE ICC '93*, May 1993, pp. 1064–1070.

[3] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[4] E. Boutillon, J. Castura, and F. R. Kschingang, "Decoder-first code design," in *Pro. 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sep. 2000, pp. 459–462.

[5] M. M. Mansour and N. Shanbhag, "Architecture-aware low-density parity-check codes," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'03)*, vol. 2, May 2003, pp. II–57 – II–60.

[6] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sep. 2000, pp. 1–8.

[7] F. Kienle and N. Wehn, "Design methodology for IRA codes," in *Proc. Asia South Pacific Design Automation Conference (ASP-DAC'04)*, Jan. 2004, pp. 459–462.

[8] F. Kienle, T. Brack, and N. Wehn, "A synthesizable IP core for DVB-S2 LDPC code decoding," in *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE'05)*, 2005, pp. 100–105.

[9] M. Gomes, G. Falcao, V. Silva, V. Ferreira, A. Sengo, and M. Falcao, "Flexible parallel architecture for DVB-S2 LDPC decoders," in *IEEE Global Telecomm. Conf. (GLOBECOM'07)*, Nov. 2007, pp. 3265–3269.

[10] J. Dielissen, A. Hekstra, and V. Berg, "Low cost LDPC decoder for DVB-S2," in *Proc. IEEE Conference on Design, Automation and Test in Europe (DATE'06)*, 2006, pp. 130–135.

[11] D. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Workshop on Signal Processing Systems (SIPS'04)*, Austin, USA, Oct. 2004, pp. 107–112.

[12] C. Marchand, J.-B. Doré, L. Conde-Canencia, and E. Boutillon, "Conflict resolution by matrix reordering for DVB-T2 ldpc decoders," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM'09)*, December 2009, pp. 1–6.

[13] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for LDPC decoding," *IEEE Trans. Inf. Theory*, vol. 53, no. 11, pp. 4076–4091, Nov. 2007.

[14] "Digital video broadcasting (DVB); frame structure, channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)," ETSI, Tech. Rep. EN 302 755 V1.2.1, Feb. 2011.

[15] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[16] "Implementation guidelines for a second generation digital terrestrial television broadcasting system (DVB-T2)," ETSI, Tech. Rep. 102 831 V0.10.4, Jun. 2010.