A DVB-S2 compliant LDPC decoder integrating the Horizontal Shuffle Scheduling.

Arthur Segard*, François Verdier*, David Declercq* and Pascal Urard†
*ETIS laboratory
University of Cergy-Pontoise
6, avenue du Ponceau - 95014 Cergy France
E-mail: {lastname}@ensea.fr
†ST Microelectronics - Crolles France
E-mail: {firstname.lastname}@st.com

***Abstract*— Low-density parity check codes (LDPC) are a class of channel decoding codes used in digital communications. Very high error correcting performances can be reached with such codes but they require both a great computing effort and randomly constructed decoding matrices. LDPC codes are used to perform the channel coding of the satellite television broadcast standard DVB-S2. This paper proposes a way to design massively parallel hardware architecture of DVB-S2 compliant LDPC decoders. It is based on a particular way to schedule all the algorithm's calculations. The proposed architecture speeds-up the decoding process, allowing the algorithm to converge faster with no significant performance loss. Moreover, a particular data update mechanism has been developed in order to avoid all data conflicts inherent to DVB-S2 matrices even for highly parallel implementations. This paper describes our hardware LDPC decoder architecture and its processing elements. Estimated silicon area of this decoder is 11 mm in ST 90 nm technology and the decoding throughput reaches 591 Mbps @ 300MHz for rate 1/2 and code size of 64800.**

I. Introduction

Low Density Parity Check codes (LDPC) are a class of channel decoding codes published in 1962 [1]. They were forgotten during decades mainly because of the implementation complexity of the decoder, until recently, when deep-submicron technologies allowed the implementation of far more complex algorithms than standard forward error correction codes: if combined with the Belief Propagation algorithm, LDPC codes can reach performance near the theoretical Shannon limit.

A binary LDPC code can be built with a parity check matrix H, containing only zeros and ones. This matrix is very sparse and the ones should be randomly allocated, to ensure the efficiency of the code. The lack of uniformity of this matrix is a central problem for the hardware implementation of the decoder.

Several hardware implementations of LDPC decoders have been described in the literature. For example [3] is trying to get the best performance possible out of one particular matrix. Their model allows a very high degree of parallelism to obtain very high data throughput, with low power dissipation. However, this method exhibits a strong lack of flexibility because important parameters cannot be changed (codeword size, code rate, etc.). Another way to obtain a parallel decoder is to give the decoding matrix a regular structure, which allows the implementation of multiple identical processors [4]. In this case, the problem consists in finding a method for building pseudo-random matrices that still have good performance (as in [5]). The DVB-S2 standard [6] describes partitioned matrices requiring a highly parallel decoding architecture. Some architectures were described to work with those specific matrices ([8], [9], and [10]).

Ways of modifying the decoder exist that can improve its performance, apart from choosing the good random matrix. The scheduling of all calculations, for example, is also an important issue. The scheduling has consequences on the error correction and on the throughput. In [7], the authors show how a good choice of scheduling can, theoretically, speed-up the decoding. There are very few attempts in the literature of hardware implementation of any other schedule than the Flooding Schedule (FS). In [10], a parallel implementation of the shuffled schedule applied on DVB-S2 matrices and highlights of some of its consequences concerning data conflicts are described. These data update conflicts are inherent to the structure of DVB-S2 matrices and are avoided in [10] both by defining the right computation order and by reducing the degree of parallelism thus reducing the expected throughput.

This paper describes our architecture combining a parallel, DVB-S2 compliant, LDPC decoder and the Horizontal Shuffle Schedule (HSS). This schedule allows a faster convergence, compared with the FS allowing then a higher throughput. Moreover we detail a particular mechanism to avoid all data conflicts that permits a massively parallel hardware implementation reaching very high throughput.

The paper is organized as follows: after this introduction, the second section describes LDPC codes and points out the scheduling issues. The third section shows how the hardware could be if the DVB-S2 standard was strictly applied and how we could improve this architecture. The fourth section shows the improved architecture and give some performance results obtained by simulations.

II. LDPC CODES FOR CHANNEL ERROR CORRECTION.

A. LDPC matrix and decoding graph.

Let us take an LDPC code, defined with its parity check matrix H. A graph G can be deduced from this matrix, to illustrate the decoding process. Each row of H (and also each parity equation) corresponds to a checknode in G. Each column of H (and also each bit of the code word) corresponds to a bitnode of G. Finally, every one in the matrix H is a link between a bitnode and a checknode in G. The decoding algorithm manipulates the probabilities of each bit being 0 or 1. These probabilities are used in the Log Likelihood Ratio (LLR) and are computed this way:

$$LRR_i(bit_i) = \log\left(\frac{P(bit_i = 0)}{P(bit_i = 1)}\right)$$

Every single LRR_i is inserted in the graph through its corresponding bitnode. Every element of the graph has a fixed purpose in the decoding:

- The edges propagate messages between bitnodes and checknodes. The messages at the beginning of the decoding are inserted in the graph through the bitnodes to the checknodes.
- Each checknode computes the probability that its equation is verified. To do that, it updates all u_j messages going to the bitnodes linked to it.
- Each bitnode computes the probability of the value of the i^{th} bit. To do that, it updates the v_j messages going to the checknodes linked to it.

To obtain a parallelized decoder, each $N \times M$ DVB-S2 matrix is generated from a sub-matrix of size $\frac{N-M}{360} \times \frac{M}{360}$. From each column of this sub-matrix can be deduced 360 columns of its corresponding DVB-S2 matrix. The $N - M$ first columns can be obtained this way. The last remaining M columns constitute the systematic part of the DVB-S2 matrix.

The sub-matrices are described in the DVB-S2 standard, and allow up to 360 processors to work at the same time during the decoding, with the FS.

B. Decoding flow

The decoding process is an iterative process. An iteration is ended when all messages of the graph have been updated at least once. The whole decoding process ends when a valid code word is found, or when the maximum number of iterations has been reached. The final decision on a bit is taken at the end the decoding process, by computing the sign of the local a posteriori log density ratio:

$$S_i = LLR_i + \sum_{j=1}^{d_v} u_k \tag{1}$$

where d_v is the degree of the i^{th} bitnode. If S_i is positive or null, the bit i is considered as a 0 (as a 1 otherwise).

The schedule has some effect on the decoder performance. In the DVB-S2 standard, the easiest way to perform the decoding consists in using the FS. One iteration of the FS, combined with the Corrected Min-Sum algorithm ([11]), includes the following steps:

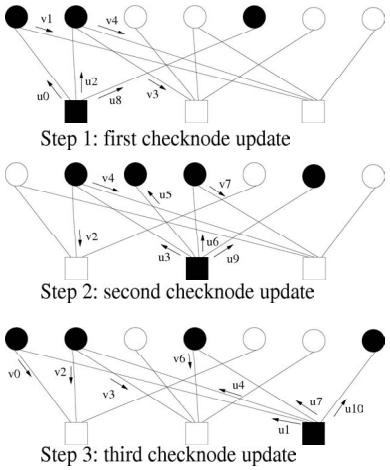

Fig. 1. Horizontal Shuffle Schedule

- step one: u_j messages updating. For a degree d_c checknode:

$$u_j = \bigoplus_{k=1, k\neq j}^{d_c} v_k \tag{2}$$

where $\oplus$ operator is defined as follows:

$$|v_i| \oplus |v_j| = \min(|v_i|, |v_j|) - offset \tag{3}$$

and

$$sign(v_i \oplus v_j) = sign(v_i) \times sign(v_j) \tag{4}$$

- step two: messages v_j updating. For a degree d_v bitnode:

$$v_j = LLR_i + \sum_{k=1, k\neq j}^{d_v} u_k \tag{5}$$

However, if the FS is convenient, we consider it is not the most powerful schedule. We propose in this paper to use the Layered Scheduling or Horizontal Shuffle Scheduling. With the HSS, the updating process is quite different from the FS (figure 1). For each checknode:

- first, the u_j messages coming out of the checknode are updated by using equations (2), (3) and (4).
- then, all v_j messages coming out of the bitnodes linked to this checknode are updated with equation (5).

During a FS, in the l^{th} iteration, the update of the messages is based only on messages updated in the $(l-1)^{th}$ iteration. During an HSS decoding, the computation can be done on messages already updated on the l^th iteration, by a different checknode.

As an example, one can see in figure 1 that the message v_4 was already updated two times before being processed by the third checknode. Globally, the number of updates of the v

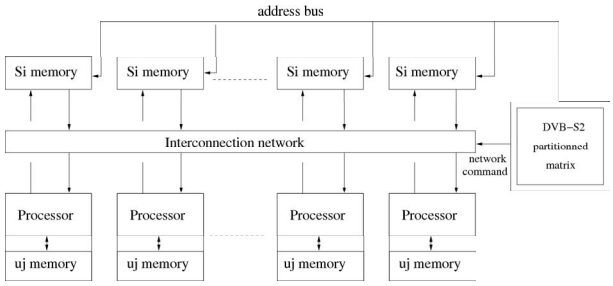

Fig. 2. parallel organization of processors for HSS

messages during one iteration depends on the bitnode degree. A v message coming out of a d_v degree bitnode is updated $d_v - 1$ times. It would be updated only once with a Flooding Schedule. This higher update frequency per iteration allows the decoder to converge faster.

III. Architecture for the DVB-S2 standard.

The channel decoding part of the DVB-S2 standard is based on partitioned LDPC decoding matrices. This partitioning allows a parallel implementation, leading to a high throughput decoding. However, we show in the following that this construction is not well adapted to a parallel implementation of HSS.

A. Applying the HSS

The particular way of building DVB-S2 matrices leads naturally to the use of the flooding schedule, as shown in [8] or [9]. To obtain an HSS compliant architecture, we decided to adapt the decoder structure to the S_i sums (equation (1)) as in [10]. To perform a checknode update with equations (2) and (3), when the v_j messages are needed, the S_i sums will be used this way:

$$v_j = LLR_i + \sum_{k=1,k\neq j}^{d_v} u_k = S_i - u_j \tag{6}$$

During the l^{th} iteration, each checknode update will be performed as follows:

- The S_i sum of all bitnodes connected to the checknode are read.
- Each v_j^l message is computed with equation (6). The u_j^{l-1}, stored in the processor, are initialized with zero values at the beginning of the decoding.
- The u^l messages are computed (equation (2), (3) and (4)).
- The updated S_i are obtained by adding all u_j^l with their corresponding v_j^l. The processor keeps the u_j^l messages for doing the subtraction during the next iteration.
- Each updated S_i sum is written back in memory.

In the proposed architecture (figure 2), the processor performs both bitnode and checknode updating at the same time. This is a real advantage to obtain a high throughput. The memories S_i are initialized with the LLR_i obtained from the communication channel, and the u_j memories are set to zero. A state machine generates the S_i memories addresses, selecting the S_i for bitnode/checknode update. A 360×360 barrel shifter allows the processors to catch the data from any memory.

However, this structure also creates a few memory access conflicts, whitch occur when a given S_i is read by two different processors, processing two different checknodes at the same time.

B. Managing the memory access conflicts

Because of the way the DVB-S2 matrices are partitioned, all processors, at any clock cycle of the decoding, request a different S_i, avoiding direct memory access conflict. During an iteration, because of the HSS, an S_i sum is read, updated and written back in memory d_v times. A conflict occurs when a S_i is read by a first processor, and then read again by a second processor before it has been updated and written back by the first processor. In this case, the first updated S_i is written back too late to be taken into account by the second processor. This, mandatory data to keep the decoder stability is lost.

This situation concerns all DVB-S2 matrices. It occurs particularly when two (or more) checknodes connected to the same bitnode are processed at the same time. It means that these processors have to update the same S_i in a parallel way. The solution used in [10] when such event occurs consists in suspending the memory read cycles until the updated values are written back. Unfortunately, we have determined that these no-operation cycles would become more frequent as the level of parallelism increases. For example, with the r=1/2 DVB-S2 matrix, such a case happens 720 times in a 40 processors architecture and 2880 times in a 360 processors architecture.

To solve this problem, we have implemented in our architecture the following strategy:

At the begining of the l^{th} iteration:

$$S_i = LLR_i + u_1^{l-1} + u_2^{l-1} + u_3^{l-1} \tag{7}$$

- S_i is read by two processors
- S_i is updated by the first processor as $S_i^1 = LLR_i + u_1^l + u_2^{l-1} + u_3^{l-1}$
- S_i is updated by the second processor as $S_i^2 = LLR_i + u_1^{l-1} + u_2^l + u_3^{l-1}$
- both S_i^1 and S_i^2 are writen in two different memory adresses

When a third processor needs S_i to update u_3, in a subsequent checknode processing:

- S_i, S_i^1 and S_i^2 are read by the processor
- an updated S_i is computed:

$$S_i^{updated} = S_i^1 + S_i^2 - S_i = LLR_i + u_1^l + u_2^l + u_3^{l-1} \tag{8}$$

- S_i is updated again as $S_i = LLR_i + u_1^l + u_2^l + u_3^l$ and writen back in memory in its regular memory address.

This two steps update process can be used when more than two processors access the same sum. The only change would be in equation (8):

$$S_i^{updated} = \sum_{k=1}^{k=m} S_i^k - (m-1) \times S_i \qquad (9)$$

with m being the number of processors.

C. Memory occupation

As illustrated in figure 2, this architecture has two types of memories. The u_j memories are local to each processor. The number of different values to store in these local memories is equal to the number of one in the DVB-S2 matrix.

We can notice that equation (3) is similar to a sort operator producing only two different absolute values. The first one corresponds to the first minimum among all the incoming $|v_j|$. The second one is the second minimum. Moreover, only one output is associated to the second minimum whatever is the degree d_c of the checknode. However, all of the produced values can have different signs. As a consequence, the amount of local memory can be drastically reduced by reserving only the space needed for storing d_c signs (d_c bits), two absolute values and an index to identify the output associated with the second minimum.

Let us suppose that all checknodes have a degree of 8, and that the messages are coded with six signed bits. For each checknode in the graph, only two absolute values coded with five bits and eight bits will be stored for the signs. The index number adds three more bits. We can see that only 21 bits are necessary for each checknode, instead of 48. This new architecture will exhibit a 56% reduction on the u_j messages memory. In our architecture, it appears that 2 Mbit of RAM are necessary to decode any matrix in the DVB-S2 standard.

IV. Simulation results

Our decoder was simulated with 6 bits data buses and a 360 × 360 barrel shifter. Only 25 iterations are sufficient to reach the same error correcting performance as in [8] using the r=1/2 matrix (figure 3). For this code rate the throughput is 394 Mbit/s at 200MHz and 591 Mbit/s at 300MHz. To obtain the standard's 135Mbit/s throughput, a 69MHz clock is enough, which could help to lower the power consumption. Table I shows the maximum throughput corresponding to each simulated matrix.

An estimation of silicon area in 90nm ST technology has been also done, and predicts that 11 mm^2 is necessary to implement our decoder. Compared with [8] (15.8 mm^2) fewer area is necessary, mainly due to a more simple processor and advances in the memory implementation technology. We need more area than [10] (4.1 mm^2) because of the high difference between the levels of parallelism (360 instead of 40).

V. Conclusion

We have described in this article a new architecture to decode DVB-S2 LDPC codes, using the Horizontal Shuffle Schedule. Combined with the MinSum algorithm, it allows a great reduction of the amount of memory necessary to run properly. The proposed architecture has a very simple structure, allowing a drastic simplification of the implementation.

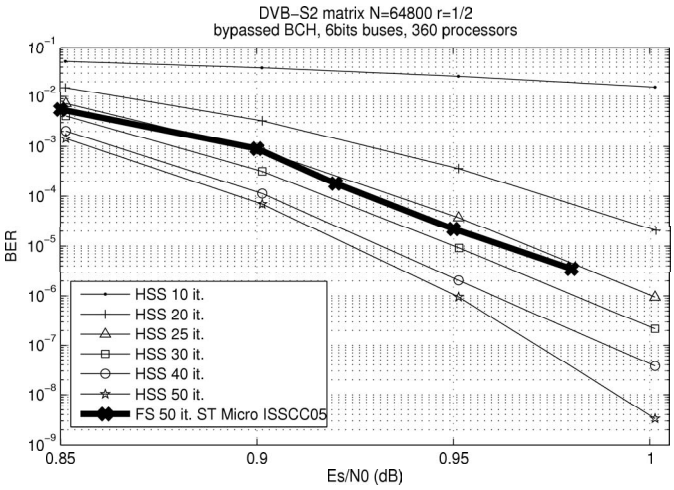

Fig. 3. Performance comparison between HSS and FS

RATE	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{3}{4}$	$\frac{3}{5}$
@ 200 MHz (Mbit/s)	363	402	394	465	246
@ 300 MHz (Mbit/s)	544	604	591	697	369

TABLE I

Maximum decoder throughput (64800 bits codeword)

References

[1] R.G. Gallager. *Low Density Parity-Check Codes*. M.I.T. Press, Cambridge, 1963.

[2] D.J.C. MacKay and R.M. Neal. *Near Shannon Limit Performance of Low Density Parity-Check Codes*. IEEE Electronnics Letters, vol. 32, no. 18, pp. 1645-1655, August 1996.

[3] A.J. Blanksby and C.J. Howland *A 690-mW 1Gb/s 1024-b, rate-1/2 low-density parity-check code decoder*. IEEE J. Solid-State Circuits, vol. 37, no. 3, pp. 404-412, March 2002.

[4] E. Boutillon and J Catsura and F.R. Kschischang *Decoder-first code design*. Symp. on Turbo Codes and Related Topics, pp. 459-462, Septembre 2000.

[5] Dale E. Hocevar *LDPC code construction with flexible hardware implementation*. ICC '03. IEEE International Conference on Communication, vol. 4, pp. 2708-2712, May 2003.

[6] ETSI EN 302 307 *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications* http://www.dvb.org/documents/ /en302307.v1.1.1.draft.pdf

[7] J. Zhang and M. Fossorier *Shuffled Belief Propagation Decoding* IEEE Trans.Commun., vol. 53, pp. 209-213, Feb. 2005.

[8] P. Urard, E. Yeo, L. Paumier, P. Georgelin, T. Michel, V. Lebars, E. Lantreibecq and B. Gupta *A 135Mb/s DVB-S2 Compliant Codec Based on 64800b LDPC and BCH Codes* ISSCC 2005, february 2005, San Francisco, CA, USA.

[9] Frank Kienle, Torben Brack and Norbert WehnM. Abramowitz. *A synthesizable IP Core for DVB-S2 LDPC Code Decoding*. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE05), vol. 3, pp. 100-105, May 2005

[10] John Dielissen, Andries Hekstra and Vincent Berg *Low cost LDPC decoder for DVB-S2* Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE06), pp. 130-135, March 2006

[11] Jinghu Chen and Marc P. C. Fossorier *Density Evolution for Two Improved BP-Based Decoding Algorithms of LDPC Codes* IEEE Communications Letters, vol. 6, pp. 208-210, no. 5, may 2002