Simplified check node processing in nonbinary LDPC decoders

E. Boutillon and L. Conde-Canencia
Lab-STICC, CNRS UMR 3192
Université Européenne de Bretagne - UBS
Lorient - FRANCE
Email: {emmanuel.boutillon, laura.conde-canencia}@univ-ubs.fr

***Abstract*—This paper deals with low-complexity algorithms for the check node processing in nonbinary LDPC decoders. After a review of the state-of-the-art, we focus on an original solution to significantly reduce the order of complexity of the Extended Min-Sum decoder at the elementary check node level. The main originality of the so-called Bubble Check algorithm is the two-dimensional strategy for the check node processing, which leads to a reduction of the number of comparisons. The simulation results obtained for the Bubble Check show that this complexity reduction does not introduce any performance loss and that it is even possible to further reduce the number of comparisons. This motivated the search of a simplified architecture and led to the L-Bubble Check, which is the main contribution of the paper. The implementation of a forward/backward check node as a systolic architecture of L-Bubble elementary checks is also described. Finally, some FPGA synthesis results of a whole GF(64)-LDPC decoder implementation are presented.**

***Index Terms*—Nonbinary low-density parity-check decoders, check node processing, simplified architecture, decoder implementation, FPGA synthesis.**

I. Introduction

The extension of binary LDPC codes to high-order Galois Fields (GF(q), with $q > 2$), aims at further close the gap of performance with the Shannon limit [1] when using small or moderate codeword lengths. This new family of codes is currently considered as a real competitor of binary LDPC and Turbo-codes for some future digital communications standards that aim at higher spectral efficiency with smaller packet length. However, the expected performance gain of Non-Binary (NB) LDPC codes comes at the expense of increased hardware complexity and a considerable effort is currently dedicated to the design of NB iterative receivers and NB link adaptation strategies with a good complexity/performance trade-off [2] [3].

In this paper, we focus on the design of low-complexity NB-LDPC decoders. Our work is dedicated to the complexity reduction of these, especially for high-order fields, which is a key feature for practical hardware implementation. In [4], we presented a detailed complexity comparison of the different existing iterative decoding algorithms applied to NB-LDPC codes. The interest of the Extended Min-Sum (EMS) algorithm [5] was then highlighted because of the significant complexity reduction it introduces compared to Belief Propagation (BP) and its derivated algorithms. The principle of the EMS decoder is to use truncated messages for both the storage and the computation of variable and check nodes. Hence the BP $O(q^2)$-complexity is theoretically reduced to the order of $n_m \times \log n_m$, where n_m is the size of the truncated messages ($n_m \ll q$), and the performance loss compared to BP decoding is small (around 0.1dB) to negligible, depending on the decoder complexity which is tuned by the value of n_m [4] [5].

In the EMS NB-LDPC decoder, one of the bottlenecks of the computations is at the Check Node (CN) level. To be specific, the complexity of the the CN is dominated by $O(n_m^2)$, due to the parallel insertion that is needed to reorder the vector messages at the elementary check node. In this paper, we review the Bubble Check algorithm, recently presented in [9] as a low-complexity original solution that significantly reduces the order of complexity of the CN. The complexity of the Bubble Check algorithm is dominated by $O(n_m \times \sqrt{n_m})$, which represents a complexity reduction of $\sqrt{n_m}$ compared to the EMS.

The simulation results obtained for the Bubble Check show that this complexity reduction does not introduce any performance loss. Moreover, for practical purposes, we propose a simplified version of the Bubble Check that improves the complexity/performance trade-off by further reducing complexity without performance loss. This simplified Bubble Check has been named L-Bubble Check and constitutes the main contribution of this paper.

The paper is organised as follows: Section II explains the state-of-the-art in EMS CN processing of NB-LDPC decoders. Section III reviews the Bubble-Check algorithm, provides some theoretical discussion on the complexity reduction and presents simulation results that motivate the search of a simplified architecture. Section IV proposes the L-Bubble Check and describes its associated architecture. The global forward/backward CN architecture is also presented. Finally, Section V draws conclusions.

II. NB-LDPC Check Node Processing

An NB-LDPC code is a linear block code defined on a very sparse parity-check matrix whose nonzero elements belong to a finite field GF(q). The construction of these codes can then be expressed as a set of parity-check equations over GF(q).

978-1-4244-6746-4/10/$26.00 ©2010 IEEE

A. Elementary Check Node Processing

The CN processing, which dominates the decoding complexity of NB-LDPC, can be implemented using the forward-backward algorithm [6]: a check node of degree d_c is decomposed in $3(d_c-2)$ Elementary Check Nodes (ECN) where an ECN has two input messages $\mathbf{U}$ and $\mathbf{V}$ and one output message $\mathbf{E}$.

In the BP algorithmn, each message $\mathbf{U}^P$, $\mathbf{V}^P$ and $\mathbf{E}^P$ contains the probability distribution of the associated variable u, v and e (respectively). In other words, let $\mathbf{U}^P$ be the probability density function defined by the q probabilities $\mathbf{U}^P(\alpha) = P(u=\alpha)_{\alpha\in\mathbf{GF}(q)}$, such that:

$$\sum_{\alpha\in\mathbf{GF}(q)} \mathbf{U}^P(\alpha) = 1$$

The messages $\mathbf{V}^P$ and $\mathbf{E}^P$ follow the same notation and u, v and e must verify the parity equation: $u \oplus v \oplus e = 0$, where $\oplus$ stands for addition in GF(q). Assuming that $\mathbf{U}^P$ and $\mathbf{V}^P$ are known, the extrinsic information $\mathbf{E}^P$, can be computed for every value $\gamma \in$ GF(q) as:

$$\mathbf{E}^P(\gamma) = \sum_{(\alpha,\beta)\in\mathbf{GF}(q)^2 | \alpha\oplus\beta=\gamma} \mathbf{U}^P(\alpha) \times \mathbf{V}^P(\beta) \qquad (1)$$

The direct computation of equation (1) requires q^2 multiplications and additions. This is why, considering values of $q > 16$, results in prohibitive complexity. Performing the computation in the frequency domain [7] reduces the order of complexity to $q\times \log(q)$ multiplications and additions. The EMS [2] [5] further reduces complexity by:

- only considering the n_m highest values of vectors $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{E}$,
- using Log-Likelihood Ratio (LLR) vectors instead of probability vectors and thus performing operations in the logarithm domain,
- approximating the addition operation by the maximum operation in the logarithm domain.

Hence the complexity of the EMS is in the order of $O(n_m \log n_m)$, and thus reasonable enough to compete with binary LDPC decoders.

B. Implementation-friendly CN processing

Considering the opposite of the classical LLR definition makes that there is no need to renormalize the messages after each computation [8]. For this reason, we define $\mathbf{U}$ as $\mathbf{U}= [U(1), U(2), \ldots, U(n_m)]$ with $U(1) = 0$ and

$$U(i) = -\log \frac{P(U^{\mathbf{GF}}(i) \mid L)}{P(U^{\mathbf{GF}}(1) \mid L)} \qquad (2)$$

where L are the local hypotheses and $\mathbf{U}^{\mathbf{GF}} = [U^{\mathbf{GF}}(1), U^{\mathbf{GF}}(2), \ldots, U^{\mathbf{GF}}(n_m)]$ is the vector of the GF(q) indexes associated to the n_m LLR values in $\mathbf{U}$. Note that the same notation applies for $\mathbf{V}$ and $\mathbf{E}$ and that the truncated messages now contain the n_m smallest LLR values sorted in increasing order. Also, the **max** operator of the classical EMS [2] becomes a **min**, and equation (1) is then replaced by:

$$E(k) = \min_{i,j\in[1,n_m]^2} \{U(i)+V(j)\}_{U^{\mathbf{GF}}(i)\oplus V^{\mathbf{GF}}(j)=E^{\mathbf{GF}}(k)} \qquad (3)$$

where the indexes $k, i, j \in \{1, \ldots, n_m\}$. Note that equation (3) does not indicate which $E^{\mathbf{GF}}(k)$ is associated to the computed $E(k)$. This is, in fact, the task of the ECN: to provide the elements of vector $\mathbf{E}$ as well as their associated $E^{\mathbf{GF}}(k)$ indexes.

C. EMS ECN processing

The EMS ECN generates the output vector $\mathbf{E}$ that contains the n_m smallest values of a matrix $\mathbf{T}_\Sigma$ defined as $\mathbf{T}_\Sigma(i,j) = U(i) + V(j)$, for $(i,j) \in [1, n_m]^2$. As the elements of $\mathbf{U}$ and $\mathbf{V}$ are sorted in increasing order, the elements of $\mathbf{T}_\Sigma$ have the following property:

Property 1: $\forall(i,j) \in [1,n_m]^2, \forall(i',j') \in [1,n_m]^2$, if $i \leq i'$ and $j \leq j' \Rightarrow \mathbf{T}_\Sigma(i,j) \leq \mathbf{T}_\Sigma(i',j')$

To obtain $\mathbf{E}$, the authors in [5] use a sorter that contains n_m competing elements from $\mathbf{T}_\Sigma$. This sorter is initialised with the first column of $\mathbf{T}_\Sigma$ and dynamically updated at each step of the algorithm. At each clock cycle, the smallest value of the sorter is extracted. For this, a parallel insertion of the new incoming value from $\mathbf{T}_\Sigma$ is needed (this implies, roughly, n_m comparisons). To obtain n_m LLR values without redundant GF(q) indexes, the number of cycles (i.e., the number of operations, denoted by n_{op}) is chosen to be (slightly) larger than n_m (typically $n_{op} = n_m + 2$). Thus, the complexity of this operation is in the order of $n_m \times n_{op}$.

III. The Bubble Check algorithm for the EMS ECN

The Bubble Check [9] reduces the complexity of the EMS ECN by exploiting the properties of the values in $\mathbf{T}_\Sigma$. This yields to a reduction of the size of the sorter and thus of the number of comparisons needed for the parallel insertion of the new incoming value.

A. The principle

From *Property 1* it follows that $\mathbf{T}_\Sigma(1,1)$ is the minimum value of $\mathbf{T}_\Sigma$ and it will thus be extracted to occupy the first position of $\mathbf{E}$ (i.e., $E(1) = \mathbf{T}_\Sigma(1,1) = 0$). For the second position of $\mathbf{E}$, there are two candidates: $\mathbf{T}_\Sigma(2,1)$ and $\mathbf{T}_\Sigma(1,2)$. If, for example, $\mathbf{T}_\Sigma(2,1)$ is extracted (i.e., $E(2) = \mathbf{T}_\Sigma(2,1) < \mathbf{T}_\Sigma(1,2)$), then the two candidates for the third position are $\mathbf{T}_\Sigma(3,1)$ and $\mathbf{T}_\Sigma(1,2)$, and so on. Note that all the candidates for a given position k belong to a different row and a different column in $\mathbf{T}_\Sigma$.

Fig. 1 presents all the possibilities for the kth position of $\mathbf{E}$, with $k = 5$. In this figure, a grey circle represents a value already extracted from $\mathbf{T}_\Sigma$ to $\mathbf{E}$ and a white circle represents a candidate (or a *bubble*) for the kth position. The name Bubble Check comes from this graphical representation, i.e., the algorithm uses *bubbles* to go through the elements

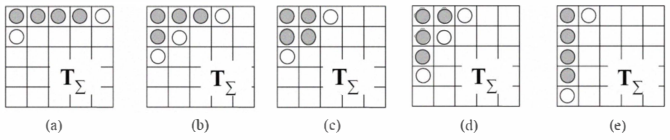

Fig. 1. Candidate values (or *bubbles*) in $\mathbf{T}_\Sigma$ to occupy the $k = 5$ position of $\mathbf{E}$. A grey circle represents a value already extracted from $\mathbf{T}_\Sigma$ to $\mathbf{E}$ and a white circle represents a candidate (or a *bubble*) for the kth position.

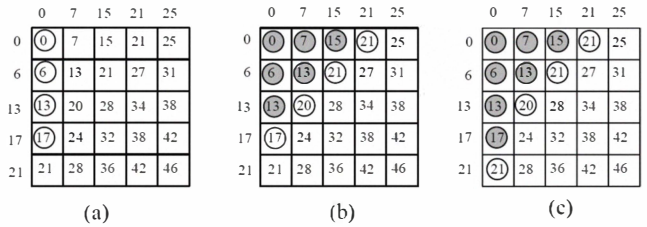

Fig. 2. Example of ECN processing with the Bubble Check algorithm

of matrix $\mathbf{T}_\Sigma$. Let n_b be the number of bubbles for the kth position, in Fig.1(a), (c), (e), n_b = 2 and in Fig. 1(b), (d), n_b = 3. At this point, the key question is: what is the maximum number of bubbles needed to perform the algorithm? For each k, the maximum value of n_b corresponds to the worst case, i.e., the already extracted values (or grey circles) have a triangular shape in $\mathbf{T}_\Sigma$. If $k-1$ is a triangular number, (i.e., $k-1 = t\times(t-1)/2$, for a given integer t) then the maximum value of n_b is t. Consequently, for all k, n_b can be bounded by the triangular root as:

$$n_b = \psi(k) = \left\lceil \frac{1+\sqrt{1+8\times(k-1)}}{2} \right\rceil \qquad (4)$$

where $\lceil x \rceil$ represents the smallest integer greater than or equal to x.

B. Complexity reduction

The maximum theoretical size of the sorter is given by equation (4) which means that, if we reduce the size of the sorter from n_m to $\psi(n_m)$, the complexity of the EMS ECN is no longer dominated by $n_m \times n_{op}$ but by $\sqrt{n_m} \times n_{op}$, which constitutes a complexity reduction of $\sqrt{n_m}$. For example, in a practical implementation of a GF(64) EMS ECN, a typical value of n_m is 15, and the number of compare operations can then be reduced by a factor of 3.

C. Description of the Bubble Check algorithm

The originality of the Bubble Check is the two-dimensional (2D) strategy to replace the extracted value in the sorter: once an element $\mathbf{T}_\Sigma(i,j)$ is moved from the sorter to $\mathbf{E}$, it can be replaced by either $\mathbf{T}_\Sigma(i+1,j)$ or $\mathbf{T}_\Sigma(i,j+1)$. To implement this, we introduce a flag, H. If $H = 1$, then $\mathbf{T}_\Sigma(i,j)$ is replaced by $\mathbf{T}_\Sigma(i,j+1)$ in the sorter; if $H = 0$, then $\mathbf{T}_\Sigma(i,j)$ is replaced by $\mathbf{T}_\Sigma(i+1,j)$.

Let n_b be the size of the sorter or the fixed number of bubbles. The algorithm can then be described as follows:

1) Initialize the n_b elements of the sorter with the first n_b values of the first column of $\mathbf{T}_\Sigma$.
2) Extract the smallest value in the sorter to the output vector $\mathbf{E}$. This element is denoted as $\mathbf{T}_\Sigma(i,j)$.
3) Flag control: change the value of H.
   a) if $(i = 1)$ then $H = 1$, $\bar{H} = 0$.
   b) if $(j = 1$ and $i = n_b)$ then $H = 0, \bar{H} = 1$.
4) Replacing rule
   a) If $\mathbf{T}_\Sigma(i+\bar{H}, j+H)$ has never been introduced in the sorter then include $\mathbf{T}_\Sigma(i+\bar{H}, j+H)$ in the sorter.
   b) else include $\mathbf{T}_\Sigma(i+H, j+\bar{H})$ in the sorter.
5) Go to 2, until $\mathbf{E}$ is completed or the maximum number of operations n_{op} is reached.

Note that, for the sake of simplicity and because our work does not introduce any novelty concerning this aspect, we have not considered the non-redundancy control of the GF(q) indexes in the description of the algorithm (see [5] for further details).

D. An example of Bubble Check processing

Fig. 2 shows an example of the EMS ECN Bubble Check for inputs $\mathbf{U} = \{0, 7, 15, 21, 25, \ldots\}$ and $\mathbf{V} = \{0, 6, 13, 17, 21, \ldots\}$. The size of the sorter is n_b = 4 and the output vector for n_{op} = 8 is $\mathbf{E}$ = {0, 6, 7, 13, 13, 15, 17, 20}. Fig. 2(a) shows the initial configuration: $k = 1$, $E(k) = \mathbf{T}_\Sigma(1,1) = 0, H = 1$. In Fig. 2(b), $k = 7, E(k) = \mathbf{T}_\Sigma(4,1) = 17$. Since $i = n_b = 4, H = 0$, then the next bubble in the sorter is $\mathbf{T}_\Sigma(5,1)$. Finally, in Fig. 2(c) $k = 8$, $E(k) = \mathbf{T}_\Sigma(3,2) = 20, H = 0$ and the next bubble in the sorter is $\mathbf{T}_\Sigma(4,2) = 24$.

E. Simulation results for the Bubble Check algorithm

We simulated the EMS decoder in [5] and the EMS ECN Bubble Check over the Additive White Gaussian Noise (AWGN) channel. The NB-LDPC that we considered are ultra-sparse and designed in GF(64). They are characterised by a fixed variable node degree d_v = 2 and constructed as in [10]. The decoder architecture follows an horizontal shuffle scheduling and a forward/backward processing at the check node level.

We considered performance comparison of the EMS and the Bubble Check for different values of n_b (see Fig. 3). For both algorithms the size of the truncated messages was chosen to be $n_m = 16$, which represents an good complexity/performance trade-off for $q = 64$ [5]. Simulations considered codewords of length $N = 192$ symbols, a code rate $R = 1/2$ and 20 decoding iterations. For the Bubble Check curves were obtained for n_b = 2, 3, 4, 5 and $n_b = \psi(n_m) = 6$. Fig. 3 shows that the Bubble Check presents no performance loss for $n_b \geq 4$. For n_b = 3 and 2, the performance loss is around 0.04 dB and 0.4 dB, respectively. The simulation of other code lengths and rates (not shown in this paper) confirms

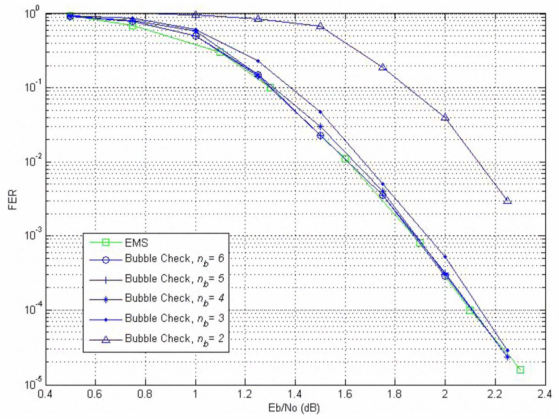

Fig. 3. Performance comparison of the EMS and the Bubble Check over the AWGN channel for different values of n_b. The simulation parameters are $N = 192$ symbols, $R = 1/2$ and 20 decoding iterations

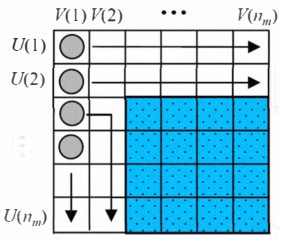

Fig. 4. L-Bubble Check exploration of matrix $\mathbf{T}_\Sigma$

that the performance of the Bubble Check algorithm with $n_b = 4$ bubbles remains identical to the performance of the EMS algorithm. This shows that, in practice, complexity can be chosen to be even lower than the theoretical by using $n_b < \psi(n_m)$.

IV. Simplified Bubble-Check or L-Bubble Check

The simulation results obtained for the Bubble Check algorithm show that, in practice, no performance loss is introduced when considering $n_b < \psi(n_m)$. This motivated the search of a simplified algorithm, whose complexity could be chosen to be smaller than the theoretical. Hence, we propose the L-Bubble Check algorithm.

A. Description of the L-Bubble Check algorithm

This algorithm is characterised by a size of the sorter always fixed to $n_b = 4 < \psi(n_m)$. The other simplification is that the flag H is not used. This means that the *paths* to follow in $\mathbf{T}_\Sigma$ are predetermined, or, in other words, only a portion of the 2D matrix is explored. The n_b = 4 values in the sorter are initialized with the *bubbles* $\mathbf{T}_\Sigma(1,1)$, $\mathbf{T}_\Sigma(2,1)$, $\mathbf{T}_\Sigma(3,1)$ and $\mathbf{T}_\Sigma(4,1)$. As depicted in Fig. 4, the first two bubbles are replaced horizontally, the third bubble is first replaced horizontally then vertically; finally, the fourth bubble is replaced vertically. Note that the values $\mathbf{T}_\Sigma(i,j)$ with $i > 2$ and $j > 2$ are ignored.

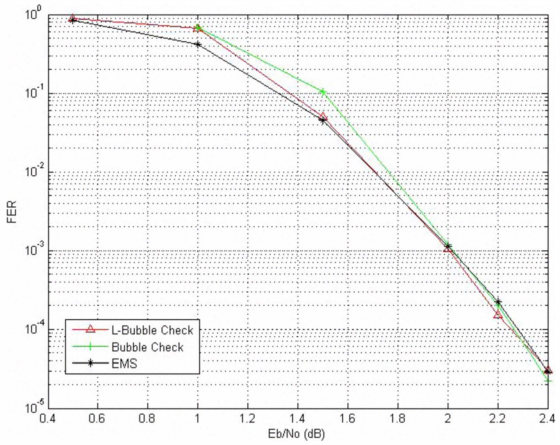

Fig. 5. Performance comparison of the EMS, the Bubble Check with $n_b = 4$ and L-Bubble Check for N = 192 symbols, R = 1/2, $n_m = 12$, $n_{op} = 24$ and 20 decoding iterations

B. Simulation results for the L-Bubble Check algorithm

Fig. 5 presents simulation results over the AWGN channel for $N = 192$, $R = 1/2$, $n_m = 12$, $n_{op} = 24$ and 20 decoding iterations. These curves confirm the no performance loss of the L-Bubble Check compared to Bubble Check or EMS. From these results we may conclude that the L-Bubble Check architecture is the one to be chosen for a practical implementation of the EMS ECN.

C. The L-Bubble Check architecture

The L-Bubble Check architecture is depicted in Fig. 6. In this figure the input values are stored in the memories **U** and **V**. The value $\mathbf{T}_\Sigma(i',j')$ is computed from two values (one of each memory) and feeds the sorter. This sorter is composed of four registers (B0, ..., B3), four multiplexers and one **min** operator. The output of the architecture is vector **E**.

The values $U(i')$ and $V(j')$ to be fetched in the memories are determined by the position that the last extracted value had in the sorter, represented by **@ind** $\in \{0,1,2,3\}$. The adresses of the bubbles in the sorter are as follows: @0 : $(1,j)$, @1 : $(2,j)$, @2 : (i,j), @3 : $(i,1)$.

The critical path is composed of the following steps:

1) Read $U(i')$ and $V(j')$ from memories **U** and **V**.
2) Compute $\mathbf{T}_\Sigma(i',j') = U(i') + V(j')$, this value becomes the **ind**'th bubble (position of the bubble extracted in the preceding cycle) and the corresponding register is thus bypassed.
3) Determine the minimum value in the sorter and its associated index **@ind** (**min** operator).
4) From **@ind**, update the address of the ith bubble and store it for the next cycle. The replacing rule is:
   a) if **@ind** = 0 or 1, then $j' = j + 1$
   b) elsif (**@ind** = 2 & $j = 1$) then $j' = 2$
   c) else $i' = i + 1$

This architecture was implemented on an FPGA device (Virtex XC4VLX200, speed 11). The results of the synthesis

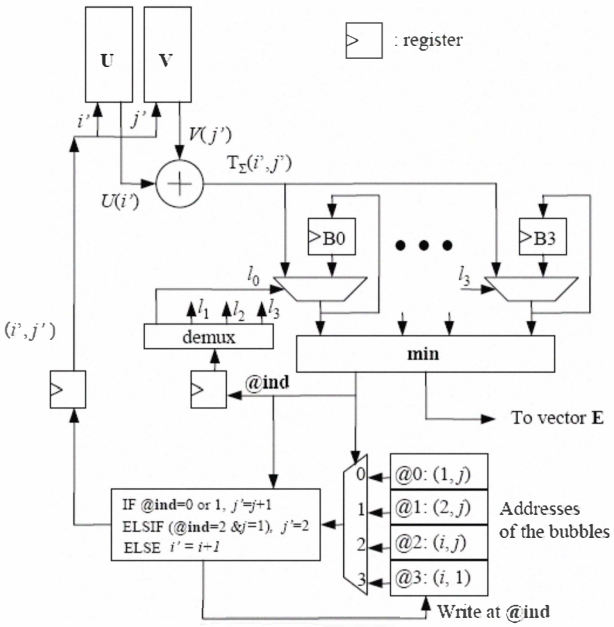

Fig. 6. Architecture scheme of the L-Bubble Check

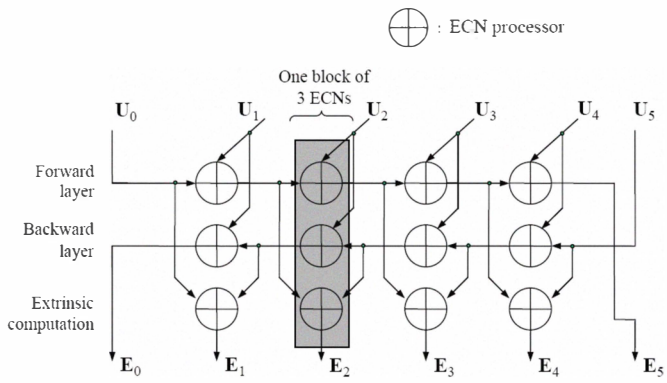

Fig. 7. Architecture scheme of a forward/backward CN processor, $d_c = 6$

were that each L-Bubble ECN requires a total of 498 slices and that the maximum clock frequency is 75 MHz. Note that this FPGA contains a total of 89,000 slices.

D. The global forward/backward CN architecture

Fig. 7 presents the forward/backward implementation of a CN as a systolic architecture. In this figure, the check node degree is $d_c = 6$ and the number of ECN processors is $3 \times (d_c - 2) = 12$. These ECNs are structured in $d_c - 2 = 4$ blocks and three layers: forward, backward and extrinsic computation.

The initialisation phase is completed when each ECN receives the first 4 elements of its local **U** and first element of **V**. Then, at each clock cycle, each ECN receives two new inputs (one from each vector **U** and **V**) and generates a new element of vector **E**. The latency of the global CN to provide the first valid data is $d_c + 2$ clock cycles. Then, $n_{op} - 1$ cycles are required to complete the whole CN processing.

The main advantage of this architecture is that it can be easily modified to implement other values of d_c (i.e., to support different code rates).

V. Conclusion

This paper was dedicated to low-complexity implementation of the elementary check node processing in NB-LDPC decoders. After a brief review of the state-of-the-art, we described in detail the Bubble Check algorithm, which consitutes an original solution to significantly reduce complexity. After the analysis of the simulation results obtained for the Bubble Check, we proposed the L-Bubble Check, as a simplified version of the Bubble Check suitable for practical implementation. We described the associated architecture and presented simulation results showing that no performance loss is introduced with the use of the simplified architecture. The L-Bubble Check architecture was adopted in a first implementation of a GF(64) rate-2/3 NB-LDPC decoder, for a frame length of 192 symbols and $n_m = 12$. The FPGA synthesis results on a Virtex4 were the following: 22 Kslices and a maximum clock frequency of 75 MHz. The decoding thoughput was in the order of 3.8 Mbps for 8 decoding iterations.

Acknowledgment

This work is supported by INFSCO-ICT-216203 DAVINCI "Design And Versatile Implementation of Non-binary wireless Communications based on Innovative LDPC Code" (www.ict-davinci-codes.eu) funded by the European Commission under the Seventh Framework Program (FP7). The authors would like to thank Ali Al Ghouwayel for his work on the decoder implementation and the FPGA synthesis.

References

[1] M. C. Davey and D. J. C. Mackay, *Low density parity check codes over GF(q)*, Information Theory Workshop, 1998, pp. 70-71.

[2] D. Declercq and M. Fossorier, *Decoding algorithms for nonbinary LDPC codes over GF(q)*, IEEE Trans. on Commun., vol. 55, pp. 633- 643, April 2007

[3] S. Pfletschinger and M. Navarro, *Link Adaptation with Retransmissions for Non-Binary LDPC Codes*, accepted in Future Network & Mobile Summit 2010, Florence, Italy, June 2010

[4] L. Conde-Canencia, A. Al Ghouwayel and E. Boutillon, *Complexity Comparison of Non-Binary LDPC Decoders*, the Proc. of ICT Mobile Summit, Santander, Spain, June 2009

[5] A. Voicila, D. Declercq, F. Verdier, M. Fossorier and P. Urard, *Low Complexity, low memory EMS algorithm for non-binary LDPC codes*, the Proc. of IEEE Intern. Conf. on Commun., ICC'2007, Glasgow, England, June 2007.

[6] H. Wymeersch, H. Steendam and M. Moeneclaey, *Log-Domain Decoding of LDPC Codes over GF(q)*, the Proc. of IEEE Intern. Conf. on Commun., ICC'2004, Paris, France, June 2004, pp. 772-776.

[7] H. Song and J. R. Cruz, *Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording*, IEEE Trans. Magn., vol. 39, pp. 1081-1087, Mar. 2003

[8] V. Sevin, *Min-Max decoding for non binary LDPC codes*, the Proc. Intern. Symposium on Information Theory, ISIT'2008, Toronto, Canada, July 2008, pp. 960-964

[9] E. Boutillon and L. Conde-Canencia, *Bubble check: a simplified algorithm for elementary check node processing in Extended Min-Sum non-binary LDPC decoders*, IEE Electronics Letters, Vol. 46, no.9, pp.633-634, April 29 2010

[10] C. Poulliat , M. Fossorier and D. Declercq, *Design of Regular (2, dc)-LDPC Codes over GF(q) Using Their Binary Images*, IEEE Trans. Commun., 2008, 56, (10), pp. 16261635.