Flexible, Cost-Efficient, High-Throughput Architecture for Layered LDPC Decoders with Fully-Parallel Processing Units

Thien T. Nguyen-Ly, Tushar Gupta, Manuel Pezzin, Valentin Savin, David Declercq, Sorin Cotofana

Flexible, Cost-Efficient, High-Throughput Architecture for Layered LDPC Decoders with Fully-Parallel Processing Units

***Abstract*—In this paper, we propose a layered LDPC decoder architecture targeting flexibility, high-throughput, low cost, and efficient use of the hardware resources. The proposed architecture provides full design time flexibility, *i.e.*, it can accommodate any Quasi-Cyclic (QC) LDPC code, and also allows redefining a number of parameters of the QC-LDPC code at the run time. The main novelty of the paper consists of: (1) a new low-cost processing unit that merges in an efficient way the logical functionalities of the Variable-Node Unit (VNU) and the A Posteriori Log-Likelihood Ratio (AP-LLR) unit, (2) a high speed, low-cost Check-Node Unit (CNU) architecture, which is executed twice in order to complete the computation of the check-node messages at each iteration, (3) a splitting of the iteration processing in two perfectly symmetric stages, executed in two consecutive clock cycles, each one using exactly the same processing resources; the processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. Synthesis results targeting a 65nm CMOS technology for a $(3, 6)$-regular $(648, 1296)$ Quasi-Cyclic LDPC code and for the WiMax $(1152, 2304)$ irregular QC-LDPC code show significant improvements in terms of area and throughput compared to the baseline architecture discussed in this paper, as well as several state of the art implementations.**

I. Introduction

Low Density Parity Check (LDPC) codes are a class of error correction codes known to closely approach to the Shannon limit under iterative message-passing (MP) decoding algorithms. MP architectures are composed of processing units that perform the desired computation by passing messages to each other. The way such architecture applies to LDPC decoding is closely related to the bipartite graph representation of LDPC codes [1]. It comprises two types of nodes, known as variable-nodes and check-nodes, corresponding respectively to coded bits and parity-check equations. Accordingly, an LDPC decoder comprises two types of processing units, namely Variable-Node Units (VNUs) and Check-Node Units (CNUs), which exchange messages according to the structure of the bipartite graph.

MP decoders may deal with different scheduling strategies, according to the order in which variable and check-node messages are updated during the message passing iterative process. The classical convention is that, at each iteration, all check-nodes and subsequently all variable-nodes pass new messages to their neighbors. This message-passing schedule is usually referred to as flooding scheduling [2]. A different approach is to split the parity check matrix in several horizontal layers, then process horizontal layer sequentially, while check-nodes (rows) within the same layer are processed by using a flooding schedule strategy. Each time a layer is processed the decoder updates the neighbor variable-nodes, so as to profit from the propagated messages, and then proceeds to the next layer. This message scheduling, known as layered scheduling [3], propagates information faster and converges in about half the number of iterations compared to the fully parallel scheduling [4], thus yielding a lower decoding latency. Layered scheduling advantageously applies to Quasi-Cyclic (QC) LDPC codes [5], which are naturally equipped with a layered structure, and also known to significantly reduce the complexity of the interconnection network. Due to their benefits in terms of area/throughput/flexibility, layered QC-LDPC decoders have been widely adopted, and can be considered as a de facto standard solution in most applications [6]. Additional considerations may address different optimizations at the processing unit level, e.g., implementing different decoding algorithms or processing the input data in either a serial or a parallel manner [7]. Regarding the MP decoding algorithm, hardware implementations of LDPC decoders mostly rely on the Min-Sum (MS) algorithm [8], since the corresponding VNUs and CNUs can be implemented by very simple arithmetic operations (additions and comparisons).

In this work, we propose a layered MS decoder architecture targeting (i) flexibility, (ii) high-throughput, and (ii) low cost and efficient use of the hardware resources. Highest flexibility can be achieved by using serial processing units: VNUs and CNUs process incoming messages in a serial manner, which makes their implementation independent of the variable or check-node degree. However, this comes at the cost of a reduced throughput. Thus, in this paper we focus on layered LDPC decoder architectures with fully parallel processing units. Such architecture has some inherent limitations in terms of flexibility, mainly concerning the number of incoming messages into VNUs and CNUs, corresponding to the degrees (i.e., number of connections) of the corresponding variable and check nodes in the Tanner graph. To ensure the highest possible flexibility, the proposed architecture can accommodate any QC-LDPC code, and also allows redefining a number of parameters at the run time, e.g., number of rows of the QC base matrix, as well as the positions and values of the non-negative entries within each row.

The classical solution to increase throughput and to also ensure an efficient use of hardware resources in layered architectures is to pipeline the datapath. However, the number of stages in the datapath may impose specific constraints on

the base matrix of the QC-LDPC code, in order to ensure that no memory conflicts occur during the read/write operations from/to the memory storing the exchanged messages or the a posteriori logarithmic likelihood ratios (AP-LLR) values. Moreover, pipelined architectures violate the layered scheduling principle, in the sense that each layer processing starts before completing processing the previous layer, thus reducing the convergence speed. To avoid such limitations, the proposed architecture does not use pipeline. Instead, we propose a specific design of the datapath processing units (VNUs, CNUs, and AP-LLR units) that allow an efficient reuse of the hardware resources, thus yielding significant cost reduction. Accordingly, the main novelty of the paper consists of: (1) A low-cost VNU/AP-LLR processing unit that merges in an efficient way the logical functionalities of the VNU and AP-LLR units, and can be executed by selecting either the VNU or the AP-LLR mode. (2) A high-speed, low-cost CNU architecture, which only computes the first minimum (`min1`) and index of the first minimum (`indx_min1`), instead of first two minima and `indx_min1` as required by the MS decoding algorithm. To compute the second minimum (`min2`), the CNU is executed a second time with `indx_min1` input set to the maximum value (according to the bit-length of the exchanged messages). Due to a specific organization of the datapath, the second execution of the CNU does not induce any penalty in terms of throughput, as explained below. (3) We split the iteration processing in two perfectly symmetric stages, executed in two consecutive clock cycles, each one using the same processing resources. In the first clock cycle we perform read operations, then execute the VNU/AP-LLR unit in VNU mode, and the CNU to compute `min1` and `indx_min1`. In the second clock cycle we execute the CNU to compute `min2`, the VNU/AP-LLR unit in AP-LLR mode, and perform write back operations. The processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. In particular, the second execution of the CNU during the second clock cycle does not impose any penalty on the operating clock frequency.

The paper is organized as follows. In Section II we briefly review QC-LDPC codes and the MS decoding algorithm. Section III details the proposed low-cost, high-throughput flexible architecture for the layered MS decoder. We discuss first the baseline architecture, and then the main enhancements that we are incorporating into this architecture. Implementation results are provided in Section IV, and Section V concludes the paper.

II. Layered MS Decoding for QC-LDPC Codes

We consider a QC-LDPC code defined by a base matrix B of size $R \times C$, with integer entries $b_{i,j} \geq -1$. The parity-check matrix H is obtained by expanding the base-matrix B by an expansion factor Z; thus, each entry of B is replaced by a square matrix of size $Z \times Z$, defined as follows: -1 entries are replaced by the all-zero matrix, while $b_{i,j} \geq 0$ entries are replaced by a circulant matrix, obtained by right-shifting the identity matrix by $b_{i,j}$ positions. Hence, H has

Algorithm 1 Layered MS decoding algorithm

Input: $(\gamma_1, \ldots, \gamma_N)$ ▷ input LLRs
Output: $(\hat{x}_1, \ldots, \hat{x}_N)$ ▷ estimated codeword
[Initialization]
for all $n = 1, \ldots, N$ **do** $\tilde{\gamma}_n = \gamma_n$;
for all $m = 1, \ldots, M$ and $n \in \mathcal{N}(m)$ **do** $\beta_{m,n} = 0$;

[Decoding Iterations]
for all iter $= 1, \ldots,$ iter_max **do** ▷ Iteration loop
  for all $r = 1, \ldots, R$ **do** ▷ Loop over horizontal layers
    for all $m \in \mathcal{M}_r$ and $n \in \mathcal{N}(m)$ **do** ▷ **VNU**
      $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$;
    for all $m \in \mathcal{M}_r$ and $n \in \mathcal{N}(m)$ **do** ▷ **CNU**
      $\beta_{m,n} = \prod_{n' \in H(m) \setminus n} \text{sign}(\alpha_{m,n'}) \cdot \min_{n' \in H(m) \setminus n} \left(|\alpha^{\text{SAT}}_{m,n'}| \right)$;
      // where $\alpha^{\text{SAT}}_{m,n}$ is the value of $\alpha_{m,n}$ saturated to q bits
    for all $m \in \mathcal{M}_r$ and $n \in \mathcal{N}(m)$ **do** ▷ **AP-LLR**
      $\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}$;
  end (horizontal layers loop)
  for all $n = 1, \ldots, N$ **do** $\hat{x}_n = \text{sign\_bit}(\tilde{\gamma}_n)$; ▷ hard decision
  if $H \cdot \hat{x}_1^N = 0$ **then** exit iteration loop; ▷ syndrome check
end (iteration loop)

$M = R \times Z$ rows and $N = C \times Z$ columns. We also denote by $\mathcal{M}_r$ the set of Z consecutive rows of H corresponding to the r-th row in B. $\mathcal{M}_r$ is further referred to as a *(decoding) layer* of H. Finally, we denote by $\mathcal{N}(m)$ the set of columns of H having a non-zero ('1') entry in the m-th row, for any $m = 1, \ldots, M$. In the bipartite graph, representation, check and variable nodes correspond respectively to rows and columns of H, and they are connected by edges according the the non-zero entries of H. The number of edges incident to each check or variable node (or equivalently, the weight of the corresponding row/column) is referred to as the node degree.

Let $(x_1, \cdots, x_N)$ denote a codeword that is sent over a binary input channel, and $(y_1, \cdots, y_N)$ be the received word. The following notation for MP decoders will be used throughout the paper:

- $\gamma_n = \log\left(\Pr(x_n = 0|y_n) / \Pr(x_n = 1|y_n)\right)$, the LLR value of x_n according to the received y_n value; it is also referred to as the a priori LLR of variable node n;
- $\tilde{\gamma}_n$: the a posteriori (AP) LLR of variable node n;
- $\alpha_{m,n}$: message sent from variable-node n to check-node m;
- $\beta_{m,n}$: message sent from check-node m to variable-node n;

The layered MS decoding algorithm is described in Algorithm 1. To match to the hardware implementation that will be discussed in the next section, we assume that input LLRs γ_n and check-to-variable node messages $\beta_{m,n}$ are quantized on q bits, while AP-LLR values $\tilde{\gamma}_n$ are quantized on $\tilde{q}$ bits, with $q < \tilde{q}$. Subtractions and additions used in the VNU and AP-LLR steps are implemented through the use of $\tilde{q}$-bit saturated adders. Hence, variable-to-check messages $\alpha_{m,n}$ computed at the VNU step are quantized on $\tilde{q}$ bits, and they are saturated to q bits just before entering the CNU. The $\alpha_{m,n}$ values used at the AP-LLR step are the unsaturated $\tilde{q}$-bit values.

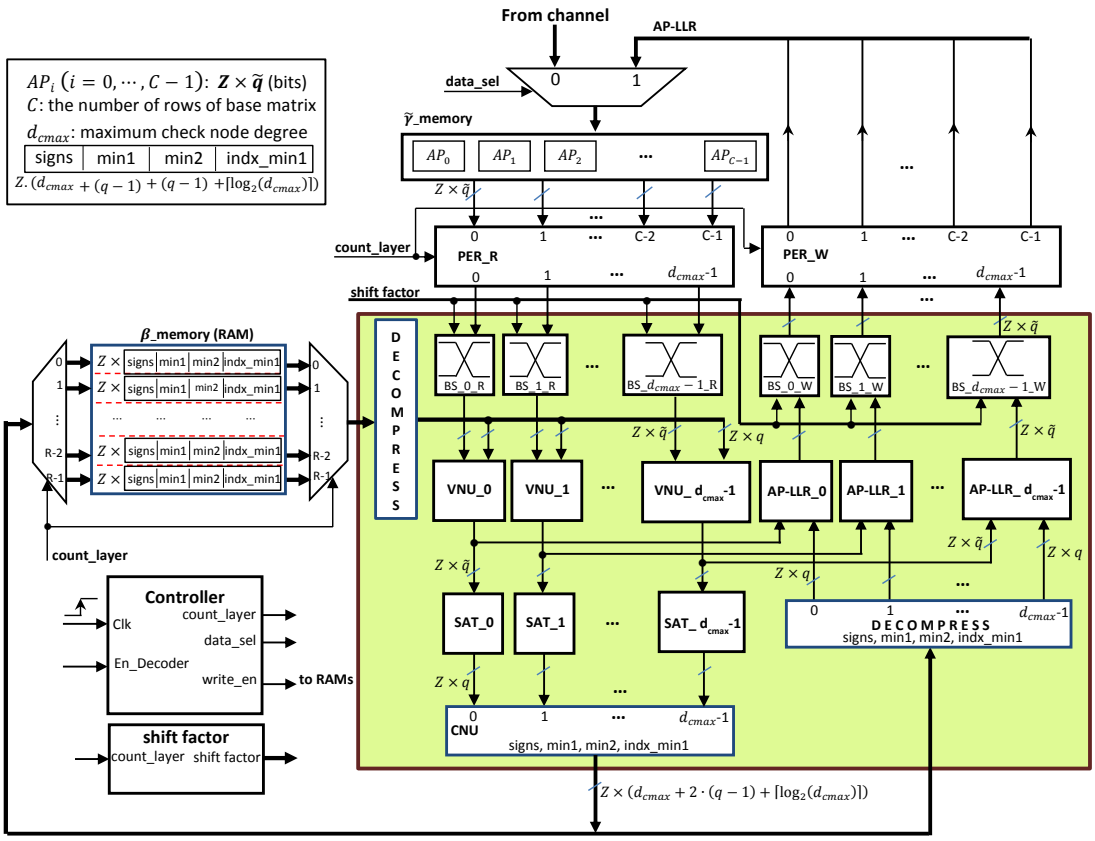

Figure 1. Block diagram of the baseline layered MS decoder architecture

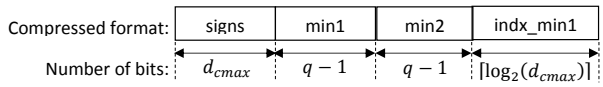

Figure 2. Compressed β-message

It is worth noting that for a given m, the absolute values of the $\beta_{m,n}$ messages computed at the CNU step are equal to either the first or the second minimum of the input messages' absolute values $|\alpha_{m,n}^{\text{SAT}}|$. Moreover, there is only one $\beta_{m,n}$ message whose absolute value is equal to the second minimum, with the variable-node index corresponding to the first minimum. In the sequel, we shall denote by `min1` and `min2` the first and second minimum, and by `indx_min1` the index of the first minimum. Thus, $\beta_{m,n}$ messages can be stored in a *compressed* format [9] to reduce memory requirements, by storing only their signs, `min1`, `min2`, and `indx_min1` values, as shown in Figure 2.

III. Layered MS Decoder Architecture

For the sake of simplicity, we shall first assume that all the check-nodes have the same degree, which will be denoted in the sequel by d_{cmax}. No further assumptions are made regarding the base matrix B. The case of check-node irregular codes will be discussed in Section III-C. We start by discussing the baseline architecture, then the proposed enhancements are discussed in Section III-B.

A. Baseline Architecture

Figure 1 illustrates the baseline architecture of the layered MS decoder, whose main blocks are further discussed below. Each decoding iteration takes two clock cycles. All data are read and processed at the first rising edge clock, then written at the second rising edge clock.

Memory blocks. Two memory blocks are used, one for the $\tilde{\gamma}_n$ values ($\tilde{\gamma}$*_memory*) and one for the $\beta_{m,n}$ messages (β*_memory*). $\tilde{\gamma}_n$ values are quantized on $\tilde{q}$ bits, and $\beta_{m,n}$ messages on q bits. $\tilde{\gamma}$*_memory* is implemented by registers, in order to allow massively parallel read or write operations. The memory is organized in C blocks, denoted by AP_i ($i = 0, \ldots, C-1$) corresponding to the number of columns of base matrix, each one consisting of $Z \times \tilde{q}$ bits. Data are read from/write to blocks corresponding to non-negative entries in the row of B (layer) being processed. β*_memory* is implemented as a Random Access Memory (RAM). Each memory word consists of Z compressed β-messages, corresponding to one row of B.

Permutations for Reading and Writing (PER_R, PER_W). PER_R permutation is used to rearrange the data read from $\tilde{\gamma}$*_memory*, according to the processed layer, so as to ensure processing by the proper VNU/CNU. PER_W block operates oppositely to PER_R.

Barrel Shifter for Reading and Writing (BS_R, BS_W).

Barrel shifters are used to implement the cyclic (shift) permutations corresponding to the non-negative entries of the base matrix B. We use d_{cmax} BS_R and d_{cmax} BS_W blocks, corresponding to the check-node degree, each of them having Z $\tilde{q}$-bit inputs and Z $\tilde{q}$-bit outputs.

Decompress. This block is used to convert $\beta_{m,n}$ messages from the compressed format to the uncompressed one.

Variable Node Units (VNUs). These processing units compute the $\alpha_{m,n}$ messages. The inputs of the VNUs are read from $\tilde{\gamma}$*_memory* and β*_memory*. Each VNU_i block ($i = 0, \ldots, d_{cmax} - 1$) in Figure 1 consists of Z $\tilde{q}$-bit saturated subtractors for the parallel execution of Z variable-nodes (one column of B).

Saturators (SATs). Prior to CNU processing, $\alpha_{m,n}$ values are saturated to q bits.

Check Node Units (CNUs). These processing units compute the $\beta_{m,n}$ messages. For simplicity, Figure 1 shows one CNU block with d_{cmax} inputs, each one of size $Z \times q$ bits. Thus, this block actually includes Z computing units, used to process in parallel the Z check-nodes within one layer. The check-node processing consists of computing the signs of the β-messages, as well as `min1`, `min2` and `indx_min1` value, and is implemented by using the high-speed low-cost (tree-structure) TS approach proposed in [10].

AP-LLR Units. These units compute the $\tilde{\gamma}_n$ values. Each AP_LLR_i block ($i = 0, \ldots, d_{cmax} - 1$) in Figure 1 consists of Z $\tilde{q}$-bit saturated adders, for the parallel execution of Z variable-nodes (one column of B).

Controller. This block generates control signals such as *count_layer* for indicating which layer is being processed, *En_read* and *En_write* for reading and writing data, etc. It also controls the synchronous execution of the other blocks.

B. Enhanced Architecture

In this Section we discuss the main enhancements that we are incorporating into the baseline architecture, which consist of (1) a low-cost VNU/AP-LLR processing unit that merges in an efficient way the logical functionalities of the VNU and AP-LLR units, (2) a low-cost CNU architecture, which is executed twice in order to complete computation of the check-node messages, (3) a splitting of the iteration processing in two perfectly symmetric stages, yielding an optimal clock frequency. VNU/AP-LLR unit and the new CNU substitute to the VNU, AP-LLR, and the old CNU units in the baseline architecture, as shown in Figure 3 (where VNU/AP-LLR is shortened to VN/AP). All the other blocks of the architecture remain the same.

1) VNU/AP-LLR Unit: The main difference between VNU and AP-LLR processing units is that subtractors are used within the first, while adders are used within the second. We propose a new VNU/AP-LLR processing unit that merges their logical functionalities, controlled by a specific signal (`sel`) to allow selecting between the VNU or AP-LLR mode. The control signal is generated by the controller, such that VNU mode is selected during the first clock, and AP-LLR mode during the second.

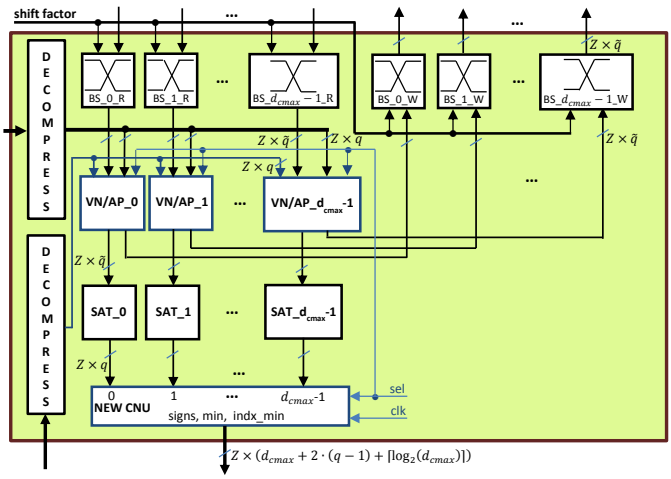

Figure 3. New processing units for the layered MS decoder architecture

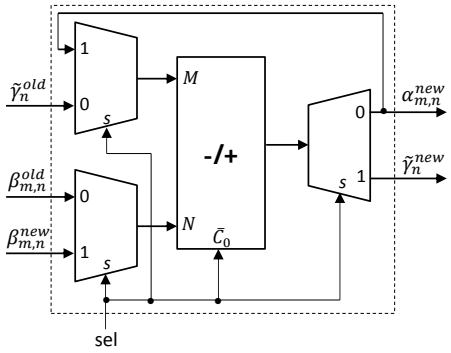

Figure 4. VNU/AP-LLR processing unit

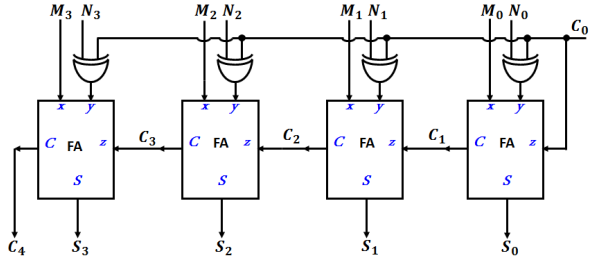

Figure 5. Adder/subtractor block used within the VNU/AP-LLR unit

The block diagram of the VNU/AP-LLR unit is detailed in Figure 4. At the input, two multiplexers are used to select the input data according to either the VNU or AP-LLR mode. Similarly, at the output, a de-multiplexer is used to choose the value of either $\alpha_{m,n}$ or $\tilde{\gamma}_n$, depending on the `sel` signal. The block in the middle, which may acts as either a subtractor or an adder is detailed in Figure 5 (by the sake of simplicity, we illustrate this block for $\tilde{q} = 4$ bits). It consist of a modified Ripple Carry Adder (RCA) with carry in given by the complement of the `sel` signal ($C_0 = \overline{\texttt{sel}}$), and which is further XORed to all the bits of the second input. It can be easily seen that the VNU/AP-LLR unit operate in VNU mode if $\texttt{sel} = 0$ ($C_0 = 1$), or in AP-LLR mode if $\texttt{sel} = 1$ ($C_0 = 0$).

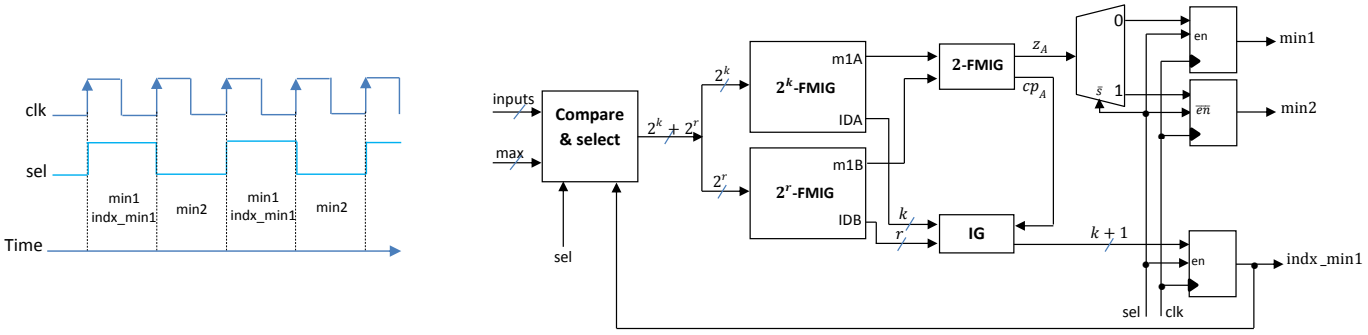

Figure 6. Block diagram of the proposed CNU architecture

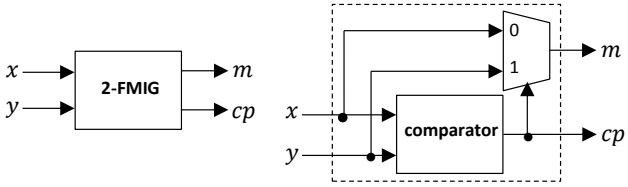

Figure 7. 2-FMIG architecture

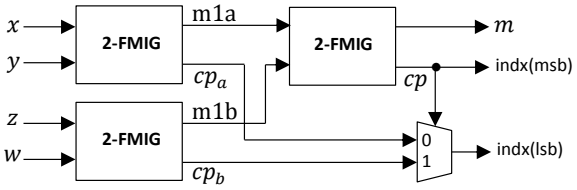

Figure 8. 4-FMIG architecture

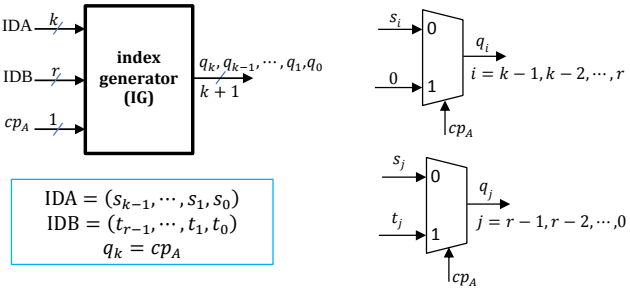

Figure 9. IG (Index Generator) architecture

2) CNU Unit: We focus only on the computation of `min1`, `min2`, and `indx_min1`, as the signs of the output messages can be simply computed by XORing the adequate signs of input messages. We propose a high-speed low-cost CNU architecture inspired by the TS architecture proposed in [10], which is further simplified so as to compute only the value and the index of the first minimum. As shown in Figure 6, our CNU is executed during the first clock cycle to compute `min1` and `indx_min1`, then it is re-executed during the second clock cycle with `indx_min1` input set to the maximum value, so that to compute `min2`. The `sel` control signal is used to indicate whether the CNU is in first or second minimum mode (first or second clock cycle). The compare and select block is used to set the `indx_min1` input to the maximum value, in case that the `sel` signal indicates that the second minimum is being computed (second clock cycle).

The proposed CNU architecture is detailed in Figure 6 for a number of inputs ($2^k + 2^r$) equal to the sum of two powers of 2. The general case can be worked out by decomposing the number of inputs as a sum of powers of 2, then combining corresponding blocks similarly to the technique used in [10]. The 2^k-FMIG (First Minimum and Index Generator) block computes the value and the index of the first minimum among the 2^k input values. The 2-FMIG block includes one comparator and one multiplexer, as shown in Figure 7. The 4-FMIG consists of three 2-FMIG blocks for finding the minimum value and one multiplexer for indicating its index, as shown in Figure 8. Similarly, the 2^{k+1}-FMIG block can be constructed from three 2^k-FMIG blocks and one multiplexer. The IG (Index Generator) block in Figure 6 is used to determine the index of the minimum value, and is further detailed in Figure 9

3) Iteration Processing Split: As shown in Figure 3, in the new architecture the clock signal is fed to the CNU. This allows splitting the iteration processing in two perfectly symmetric stages, executed in two consecutive clock cycles, each one using the same processing units, but in different mode. In the first clock cycle we perform read operations, then execute the VNU/AP-LLR unit in VNU mode, and the CNU to compute `min1` and `indx_min1`. In the second clock cycle we execute the CNU to compute `min2`, the VNU/AP-LLR unit in AP-LLR mode, and perform write back operations. The processing load is perfectly balanced between the two clock cycles, thus yielding an optimal clock frequency. In particular, the second execution of the CNU during the second clock cycle does not impose any penalty on the operating clock frequency. The baseline CNU (i.e. computing `min1`, `min2`, and `indx_min1`) executed in one of the two clock cycles would lead to an increased critical path, and therefore a reduced clock frequency, while splitting its execution between the two clock cycles would have resulted in an inefficient use of the hardware resources.

C. Case of Check-Node Irregular Codes

To accommodate QC-LDPC codes with variable check-node degree $d_c \in [d_{cmin}, d_{cmax}]$, some extra control logic is

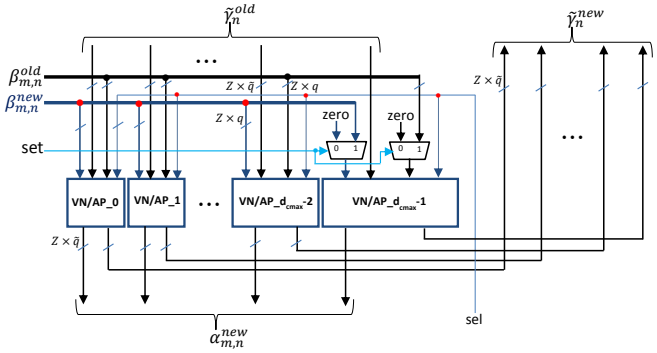

Figure 10. Modified VNU to accommodate variable check-node degree (example for $d_{cmin} = d_{cmax} - 1$)

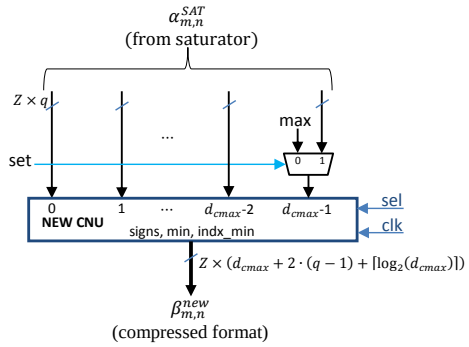

Figure 11. Modified CNU to accommodate variable check-node degree (example for $d_{cmin} = d_{cmax} - 1$)

required in order to "inactivate" the last $d_{cmax} - d_c$ VNU/AP-LLR units, as well as the last $d_{cmax} - d_c$ inputs of the CNU, for check-nodes of degree d_c. If the check node degree d_c varies between d_{cmin} and d_{cmax}. A VNU/AP-LLR unit is inactivated by setting the corresponding β-inputs to 0, while an input of the CNU is inactivated by setting it to the maximum value ($2^{q-1} - 1$, where q is the number of quantization bits of input $\alpha_{m,n}^{\text{SAT}}$ values, including the sign bit). The modified VNU/AP-LLR and CNU architectures are shown in Figure 10 and Figure 11, respectively, for $d_{cmin} = d_{cmax} - 1$.

D. Design and Run Time Flexibility

Figure 12 details the flowchart of the QC-LDPC decoder generation. The VHDL inputs consist of two configuration files, for the base-matrix related parameters and the user-defined parameters. Base-matrix parameters relate to either the matrix size (number of rows and columns, expand factor) or to the number, position and values of the non-negative entries (d_{cmin}, d_{cmax}, positions and values on non-negative entries per row). While some of these parameters are fixed, meaning that they cannot be overwritten at run time, the number of rows of the base matrix as well as the positions and values on non-negative entries per row can be overwritten at run time, while still ensuring proper operation of the decoder using the redefined base-matrix. This property is particularly useful to achieve flexibility of the implemented decoder with respect to the coding rate. Note also that it would also be possible to achieve flexibility with respect to the expansion factor (Z) value, by including some extra control logic. However, such control logic has not been included in our current implementation, so we report this parameter as being fixed.

The RPL parameter shown in Figure 12 allows defining the number of base matrix Rows Per Layer. For the sake of simplicity, we have assumed so far that one decoding layer corresponds to one row of the base matrix B. However, in general it is also possible to define a decoding layer as RPL consecutive rows of the base matrix, as long as each column of B has at most one non-negative entry in each layer. This feature has been integrated to our design. If RPL > 1, the number of decoding layers is equal to R/RPL, with RPL $\times\, Z$ check nodes per each layer.

Finally, the user-defined parameter allows specifying the quantization parameters $(q, \tilde{q})$, and the number of decoding iterations.

IV. Implementation Results

We have implemented the baseline and enhanced layered MS decoder architectures for a regular QC-LDPC code with variable-nodes of degree $d_v = 3$, and for the irregular WiMAX QC-LDPC code with rate $1/2$ [11]. For both codes, the size of the base is equal to $R \times C = 12 \times 24$. For the regular code, the base matrix B is shown in Figure 13. It can be divided in 3 horizontal layers, with each layer corresponding to RPL $= 4$ consecutive rows of B. For the WiMAX code, the RPL value is set to 1, thus the number of decoding layers is equal to 12. Configuration parameters of the two decoders are further detailed in Table I.

ASIC synthesis results targeting a 65nm CMOS technology are shown in Table II. The top part of the table reports the maximum operating frequency, the corresponding throughput, and the area. The reported throughput is given by the formula:

$$\text{Throughput} = \frac{N \times f_{max}}{\text{iter\_number} \times \text{cyc\_iter}},$$

where $N = C \times Z$ is the codeword length, and cyc_iter $= 2 \times (R/\text{RPL})$ is the number of clock cycles to complete one iteration (2 clock cycles per layer, times the number of layers). First, we note that the enhanced architecture provides a significant increase in the maximum operating frequency compared to the baseline architecture, by a factor of $\times 2.25$ and $\times 3$, for the $(3, 6)$-regular and the WiMAX code, respectively. This is due to the proposed increased-speed CNU together with the proposed split of the iteration processing. Regarding the area, it can be seen that the enhanced architecture provides a significant area reduction for the $(3, 6)$-regular code, by 24.2% compared to the baseline architecture. However, the area reduction is of only 2.27% for the WiMAX code. In oder to keep the area comparison on an equal basis with respect to synthesis timing constraints, in the bottom part of Table II we report area figures when the same timing constraints are applied to both the baseline and the enhanced architecture. We consider timing constrains corresponding to the maximum operating frequency for the baseline architecture. In this case, it can be seen that the proposed cost-efficient VNU/AP-LLR

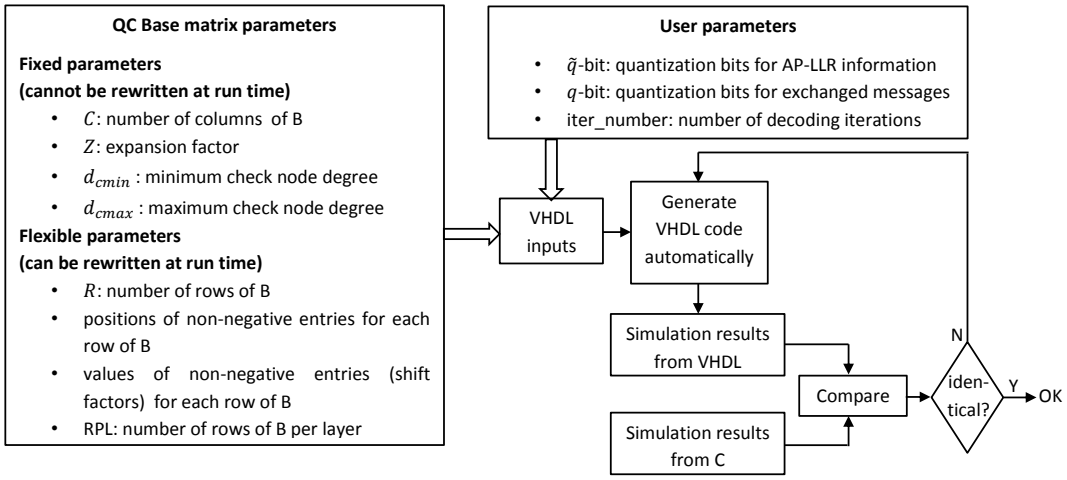

Figure 12. Flowchart for QC-LDPC decoder generation

49	-1	-1	-1	-1	**43**	-1	-1	-1	-1	**50**	-1	-1	-1	-1	**2**	-1	**27**	-1	-1	-1	-1	-1	**49**
-1	-1	-1	**10**	**41**	-1	-1	-1	-1	**52**	-1	-1	**32**	-1	-1	-1	-1	-1	**50**	-1	**50**	-1	-1	-1
-1	-1	**20**	-1	-1	-1	-1	**20**	-1	-1	-1	**51**	-1	**10**	-1	-1	**47**	-1	-1	-1	-1	-1	**33**	-1
-1	**24**	-1	-1	-1	-1	**22**	-1	**53**	-1	-1	-1	-1	-1	**31**	-1	-1	-1	-1	**18**	-1	**47**	-1	-1
10	-1	-1	-1	**15**	-1	-1	-1	-1	-1	**2**	-1	-1	-1	-1	**50**	-1	**13**	-1	-1	-1	-1	-1	**53**
-1	-1	**44**	-1	-1	**6**	-1	-1	-1	-1	-1	**29**	-1	**40**	-1	-1	**16**	-1	-1	-1	**13**	-1	-1	-1
-1	**2**	-1	-1	-1	-1	-1	**13**	**41**	-1	-1	-1	-1	-1	**42**	-1	-1	-1	-1	**48**	-1	**49**	-1	-1
-1	-1	-1	**36**	-1	-1	**24**	-1	-1	**50**	-1	-1	**12**	-1	-1	-1	-1	-1	**10**	-1	-1	-1	**48**	-1
-1	-1	**47**	-1	**50**	-1	-1	-1	-1	-1	**0**	-1	-1	-1	-1	**9**	-1	**7**	-1	-1	-1	-1	-1	**28**
-1	**24**	-1	-1	-1	-1	-1	**51**	-1	**38**	-1	-1	-1	-1	**6**	-1	-1	-1	-1	**23**	-1	**16**	-1	-1
6	-1	-1	-1	-1	-1	**5**	-1	-1	-1	-1	**13**	-1	**3**	-1	-1	**29**	-1	-1	-1	**16**	-1	-1	-1
-1	-1	-1	**35**	-1	**16**	-1	-1	**37**	-1	-1	-1	**4**	-1	-1	-1	-1	-1	**24**	-1	-1	-1	**29**	-1

Figure 13. Base matrix of the $(3, 6)$-regular QC-LDPC code

Table I
PARAMETERS OF THE QC-LDPC CODES

	R	C	Z	RPL	d_{cmin}	d_{cmax}	$\tilde{q}$	q	iter_number
(3,6)-regular	12	24	54	4	6	6	6	4	20
WiMAX	12	24	96	1	6	7	6	4	20

and CNU processing units yield an area reduction by 25.26% for the $(3, 6)$-regular code, and by 13.64% for the WiMAX code.

For the WiMAX QC-LDPC code, the proposed enhanced architecture is further compared with other state of the art implementations in Table III. We also report throughput and area figures scaled to 65nm [12], as well as the Throughput to Area Ratio (TAR) and the Normalized TAR (NTAR) metrics [13], so as to keep the throughput comparison on an equal basis with respect to technology, area, and number of iterations. To scale throughput and area to 65nm, we use scale factors $(\text{technology\_size}/65)$ and $(65/\text{technology\_size})^2$, as suggested in [12]. The computation of the TAR and NTAR metrics is detailed in the footnote to Table III. Note that for all the reported implementations, the achieved throughput is inversely proportional to the number of iterations, hence the NTAR metric corresponds to the TAR value assuming that only one decoding iteration is performed. We mention that the decoder proposed in [13] is a reconfigurable decoder that supports the IEEE 802.16e (WiMAX) and and the IEEE 802.11n (WiFi) wireless standards. The reported throughput is the maximum achievable coded throughput for the $(1152, 2304)$ WiMAX code with 5 decoding iterations. From Table III it can be seen that the proposed enhanced architecture compares favorably with state of the art implementations, yielding a NTAR value of 27.9 Gbps/mm^2/iteration.

Table II
COMPARISON BETWEEN ENHANCED AND BASELINE ARCHITECTURES FOR $(3, 6)$-REGULAR AND WIMAX QC-LDPC CODES

	$(3, 6)$-regular QC-LDPC		WiMAX QC-LDPC	
	Baseline	Enhanced	Baseline	Enhanced
Max. Freq. (MHz)	111	250	83	250
Throughput (Mbps)	1198	2700	398	1200
Area (mm^2)	0.95	0.72	0.88	0.86
Frequency (MHz)	111		83	
Area (mm^2)	0.95	0.71	0.88	0.76

Finally, we mention that for the $(3, 6)$-regular QC-LDPC code, the proposed enhanced architecture achieves an NTAR value of 75 Gbps/mm^2/iteration.

V. CONCLUSION

In this paper we proposed a low-cost and flexible architecture for high-throughput layered LDPC decoders with fully-parallel processing units. To do so, we proposed new processing unit architectures that allow a more efficient hardware usage, thus yielding a significant cost reduction. The proposed CNU further allows splitting the iteration processing in two perfectly symmetric stages, resulting in a significant increase in the maximum operating frequency. The proposed enhanced architecture allows full design time flexibility, and also provides good run time flexibility, by allowing the same architecture being executed with different base matrices sharing a number of common characteristics. Finally, the benefits of the proposed architecture have been demonstrated through comparison with a baseline layered architecture with fully-parallel processing units, as well as several state of the art implementations of layered LDPC decoders.

Table III
COMPARISON BETWEEN THE PROPOSED ENHANCED ARCHITECTURE AND STATE OF THE ART IMPLEMENTATIONS FOR THE WIMAX QC-LDPC CODE

	Y. Ueng (2008) [14]	K. Zhang (2009) [15]	T. Heidari (2013) [16]	K. Kanchetla (2016) [13]	Proposed decoder
Code length	2304	2304	2304	576-2304	2304
Technology (nm)	180	90	130	90	65
Frequency (MHz)	200	950	100	149	250
Iterations	4.6 (average)	10	10	5	20
Throughput (Mbps)	106	2200	183	955	1200
Tput. scaled to 65nm (Mbps)	294	3036	366	1318	1200
Area (mm^2)	-	2.90 $^{(*)}$	6.90 $^{(**)}$	11.42 $^{(*)}$	0.86 $^{(*)}$
Area scaled to 65nm (mm^2)	-	1.51 $^{(*)}$	1.73 $^{(**)}$	5.94 $^{(*)}$	0.86 $^{(*)}$
TAR (Mbps/mm^2)	-	2010.60	211.56	221.89	1395.35
NTAR (Mbps/mm^2/iter)	-	20106	2115.6	1109.45	27907

$^{(*)}$ only core area is reported

$^{(**)}$ total chip area is reported

TAR = (Throughput scaled to 65nm) / (Area scaled to 65nm)

NTAR = TAR $\times$ Iterations

REFERENCES

[1] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[2] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219–230, 1998.

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

[4] J. Zhang, Y. Wang, M. P. Fossorier, and J. S. Yedidia, "Iterative decoding with replicas," *IEEE Transactions on Information Theory*, vol. 53, no. 5, pp. 1644–1663, 2007.

[5] M. P. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[6] E. Boutillon and G. Masera, "Hardware design and realization for iteratively decodable codes," in *Channel Coding: Theory, Algorithms, and Applications*, D. Declercq, M. Fossorier, and E. Biglieri, Eds. Academic Press Library in Mobile and Wireless Communications, Elsevier, June 2014.

[7] O. Boncalo, A. Amaricai, A. Hera, and V. Savin, "Cost efficient FPGA layered LDPC decoder with serial AP-LLR processing," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, Munich, Germany, September 2014, pp. 1–6.

[8] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Communications*, vol. 47, no. 5, pp. 673–680, 1999.

[9] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 483–488, 2007.

[10] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin, "Algorithms of finding the first two minimum values and their hardware implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3430–3437, 2008.

[11] IEEE-802.16e, "Physical and medium access control layers for combined fixed and mobile operation in licensed bands," 2005, amendment to Air Interface for Fixed Broadband Wireless Access Systems.

[12] J. R. Hauser, "MOSFET device scaling," in *Handbook of Semiconductor Manufacturing Technology*. Boca Raton, FL: CRC Press, 2008, pp. 8–21.

[13] V. K. Kanchetla, R. Shrestha, and R. Paily, "Multi-standard high-throughput and low-power quasi-cyclic low density parity check decoder for worldwide interoperability for microwave access and wireless fidelity standards," *IET Circuits, Devices & Systems*, vol. 10, no. 2, pp. 111–120, 2016.

[14] Y.-L. Ueng, C.-J. Yang, Z.-C. Wu, C.-E. Wu, and Y.-L. Wang, "VLSI decoding architecture with improved convergence speed and reduced decoding latency for irregular LDPC codes in WiMAX," in *IEEE International Symposium on Circuits and Systems, ISCAS 2008.*, 2008, pp. 520–523.

[15] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic ldpc codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985–994, 2009.

[16] T. Heidari and A. Jannesari, "Design of high-throughput qc-ldpc decoder for wimax standard," in *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, 2013, pp. 1–4.