A Deeply Pipelined, Highly Parallel and Flexible LDPC Decoder

Jérémy Nadal*, Mickaël Fiorentino*, Elsa Dupraz† and François Leduc-Primeau*
*École Polytechnique de Montréal, Montréal, Québec, Canadam, †IMT Atlantique, Lab-STICC, UBL, Brest, France

***Abstract*—A deeply pipelined and parallel LDPC decoder architecture is proposed in this paper. The main feature of this architecture is the Δ-update scheme, which relaxes the data dependency requirement and allows for deeper pipelines than typical decoders. The proposed architecture also has the flexibility to handle a large number of codes. Frame error rate performance is shown for three codes with different quantization parameters. Finally, the impact of pipeline depth on processing time and on the energy-delay product (EDP) is evaluated from post-synthesis results. The results show that the ability to have deeper pipelines can lead to large reductions in EDP.**

I. Introduction

Low-density parity check (LDPC) is the key correcting code for several communication standards, such as 801.11n (WiFi) and 5$^{\text{th}}$ generation new radio (5GNR). Building hardware implementations of LDPC decoders is more and more challenging due to the increasing demands in data rate, quality-of-service, and flexibility while keeping the latency and the energy consumption as low as possible. This is for instance the case for the upcoming 5GNR standard, where new services such as ultra-reliable low-latency communication (URLLC) are introduced. Some internet-of-thing applications may also require central units with very powerful decoding capabilities, due to the important amount of data they receive from thousands of machines or sensors.

In this context, we propose a novel highly parallel and deeply pipelined LDPC decoder architecture for high data rate and low latency applications, which we have implemented in a 65 nm technology node. Deep pipeline is supported thanks to a novel Δ-update mechanism, introduced in [1], which allows to ignore some of the data dependencies of the layered schedule. This paper provides the first hardware implementation of a decoder architecture based on the Δ-update, including post-synthesis results. It demonstrates for the first time the benefits of using these techniques in terms of latency, processing throughput, and energy consumption, for several codes having different sparsity and regularity characteristics. It also shows that the proposed architecture is flexible enough to support many type of LDPC codes.

This paper is structured as follows. Section II describes LDPC codes and the offset min-sum (OMS) decoding algorithm with the Δ-update mechanism. Then, Section III introduces the proposed Δ-update LDPC decoder architecture. Section IV presents the post-synthesis and frame error rate (FER) performance results for different pipeline depth and quantization choice. The impact of the pipeline on the processing time and the energy-delay product (EDP) is also analysed in this section. Finally, Section V concludes the paper.

II. LDPC Codes and Decoder

A. LDPC Codes

The quasi-cyclic (QC)-LDPC codes [2] are considered in this paper due to their suitability for hardware implementation. The QC-LDPC code construction starts with a base matrix (BM) of size $N_r \times N_c$. Each non-zero element of this BM is expanded by a Z-by-Z cyclically-shifted identity matrix, where Z is the lifting size. The circular shift values depend on Z and on the edge positions in the BM. This family of codes allows for efficient hardware implementations with a high degree of parallelism and a high processing throughput [3]. In this work, we consider 3 types of LDPC codes: *i)* 5GNR code [4] with BM index 1 (BM1), code rate (CR) $1/3$, and $Z = 32$, *ii)* 802.11n WiFi code [5] with CR $1/2$, and $Z = 27$, and *iii)* the code proposed in [1], with a 9-color constraint and $Z = 18$, referred as $C_9 - Z_{18}$ in the rest of paper.

B. Quantized OMS-LDPC Decoder

After encoding and transmitting the message over the noisy channel, the received signal at time i is $y_i = x_i + w_i$, where x_i is the i^{th} transmitted modulated symbol, and w_i is the additive Gaussian noise with variance σ^2. For simplicity, the BPSK modulation is employed: $x_i \in \{-1, +1\} = 2c_i - 1$, where c_i is the i^{th} transmitted bit. For each channel output y_i, the decoder first computes Q_μ-bits belief outputs μ_i as $\mu_i = \text{sat}_{Q_\mu}(\lfloor \alpha y_i \rceil)$, where $\lfloor . \rceil$ define the rounding operator, α is a belief scaling parameter depending on Q_μ and σ^2, and

$$\text{sat}_Q(x) = \max(\min(x, 2^{Q-1} - 1, -2^{Q-1}))$$

is the Q-bits saturation operator. To describe the decoding algorithm, the following variables are introduced: $\mu_{i\to j}^{(t)}$ corresponds to the variable node (VN) i to check node (CN) j message at iteration number t, $\lambda_{j\to i}^{(t)}$ is the CN j to VN i message at t, Λ_i is to the current sum of the incoming messages at VN index i. The Λ_i are referred to as the VN belief sums, V_j is the set of VN indexes connected to the CN j, N_r is the number of rows in the BM, β is a positive integer coefficient corresponding to the offset parameter, Q_λ and $Q_\Lambda > Q_\mu$ respectively represent the number of quantization bits for the $\lambda_{j\to i}^{(t)}$ variables and for the Λ_i variables.

The OMS LDPC decoding algorithm [6] with row-layered scheduling works as follows. At the first iteration $t = 1$, the VN belief sums Λ_i are initialized with the channel beliefs μ_i,

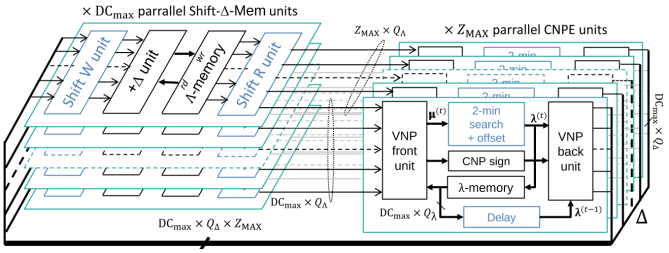

Fig. 1. Proposed LDPC decoder hardware architecture.

and the messages $\mu_{i\to j}^{(1)}$ are set to 0. Then, at each iteration $t > 1$, the messages $\mu_{i\to j}^{(t)}$ and the belief sums Λ_i are sequentially updated for each $j \in \{0, ..., Z \times (N_r - 1)\}$ according to the following equations, $\forall i$:

$$\mu_{i\to j}^{(t)} = \text{sgn}(\Lambda_i - \lambda_{j\to i}^{(t-1)})\text{sat}_{Q_\lambda}(|\Lambda_i - \lambda_{j\to i}^{(t-1)}|), \quad (1)$$

$$\lambda_{j\to i}^{(t)} = \prod_{l\in V_j\setminus i} \text{sgn}(\mu_{l\to j}^{(t)}) \max\Big(\min_{l\in V_j\setminus i} |\mu_{l\to j}^{(t)}| - \beta, 0\Big), \quad (2)$$

$$\Lambda_i = \text{sat}_{Q_\Lambda}\big(\lambda_{j\to i}^{(t)} + \mu_{i\to j}^{(t)}\big), \quad (3)$$

where sgn(.) denotes the sign operator. The algorithm stops when a given number of iterations is reached or when an early termination (ET) criterion is fulfilled, for instance if all parity-check equations are satisfied. When considering QC parity-matrices, the VN belief sums Λ_i are typically processed in parallel per sub-block of Z rows, called *layer*, due to the absence of data dependency in these sub-blocks.

C. Δ-Update Technique

Data dependencies between successive layers cause update conflicts which can highly damage the FER performance or the processing rate of the decoder. The sparsity of the LDPC parity-check matrix often limits the data dependencies. However, some irregular codes, such as the ones used in 5GNR or 802.11n standards, can be very dense. To avoid such conflicts, the Δ-update mechanism was proposed in [1]. The main idea is to compute the difference Δ_i between the updated VN belief sum and its current value. Once the computation is finished, Δ_i is used to update the next VN belief sum Λ_i. If a VN is involved in several concurrent CN computations, its message-update schedule is simply altered, while other VNs can still be updated normally. To incorporate the Δ-update mechanism in the layered OMS LDPC decoder algorithm described in Section II-B, equation (3) is replaced by

$$\Delta_i = \lambda_{j\to i}^{(t)} - \lambda_{j\to i}^{(t-1)},$$

and the VN belief sums are updated by adding Δ_i to the previous Λ_i value: $\Lambda_i \leftarrow \text{sat}_{Q_\Lambda}(\Delta_i + \Lambda_i)$. Equations (1) and (2) remain unchanged.

III. Proposed Hardware Architecture

A. Delta-Update LDPC Decoder Architecture

The top-level architecture of the proposed LDPC decoder is shown in Fig. 1. To allow high-throughput and low latency decoding, two parallelism axes are considered:

- up to DC_{MAX} VN belief sum Λ_i in the BM (i.e. entire check nodes) are processed in parallel,
- up to Z_{MAX} rows (i.e. all check nodes in a layer) are processed in parallel.

Therefore, the architecture is composed of DC_{MAX} `Shift-Δ-Mem` units connected to Z_{MAX} check node processing elements (CNPEs). The `Shift-Δ-Mem` unit number $l \in \{0, ..., DC_{\text{MAX}} - 1\}$ takes as input, after the first decoding iteration, a vector $\boldsymbol{\Delta}$ of Z_{MAX} Δ-update signals (Q_Δ bits) coming from the l^{th} output of the Z_{MAX} `CNPEs`. Each `shift` unit is composed of two barrel shifters in parallel followed by a last multiplexer stage for selecting the output [7]. The `Λ-memories`, each of depth N_{subVN} store the Z_{MAX} VN belief sums of Q_Λ bits associated with a subset of N_{subVN} variable nodes. Each row of the memory contains variable nodes associated with a single block-column of the parity-check matrix.

The novelty of the architecture resides in the $+\Delta$ units that add the shifted Δ-update and the VN belief sums to update, read from the `Λ-memories`. The outputs of the $+\Delta$ units corresponds to the updated VN belief sums vector, and is written back into the `Λ-memories`. As described in Section II-C, this scheme enables to support deep pipeline configurations without update conflicts, at the cost of additional decoding iterations. This is particularly needed for highly parallel architectures where several pipeline stages are required to reduce the critical path of the `2-min` search computation tree. It is worth noting that, contrary to most state-of-the-art architectures, it is not possible to avoid the use of the `shift W` unit in Fig. 1 due to the Δ-update scheme [1]. It is however a minor drawback as it only introduces a slight complexity increase and deeper pipeline depth can be supported.

In the CNPE architecture, the variable node processor (VNP) front unit subtracts the VN Belief sums Λ_i to the Q_λ bits message values $\lambda^{(t-1)}$ of the previous iteration. After sign and magnitude separation, the magnitudes of the obtained DC_{MAX} messages are saturated to $Q_\lambda - 1$ bits and fed to the `2-min` search unit. This block is composed of cascaded sort and merge structures described in [8]. The major change compared to typical CNPE architectures comes from the VNP back unit. It outputs the Δ-update values of $Q_\Delta = Q_\lambda + 1$ bits obtained by subtracting the old message values λ^{t-1} with the newly obtained message values λ^t. These new messages are finally stored into the `λ-memories`, each of depth $N_{r,\text{MAX}}$ which corresponds to the maximum number of BM rows that the decoder can support.

B. Pipeline Configuration

Several pipeline configurations are supported on blocks having a recursive structure. This is the case for the `shift` and the `2-MIN` search units, highlighted in blue in Fig. 1, where a pipeline register can be instantiated (or not) between each stage. Furthermore, the clock cycle latency due to the `Λ-memory` write cycle is counted as a pipeline stage, and one mandatory pipeline register stage is added before the $+\Delta$ unit. Let $d_{\texttt{shift}} \in \{1, ..., \log_2(Z_{\text{MAX}})\}$ and $d_{\texttt{2-MIN}} \in$

TABLE I
SIMULATION PARAMETERS FOR THE CODES AND THE DECODING.

Code type	5GNR			801.11.n			$C_9 - Z_{18}$		
Code rate	1/3			1/2			1/2		
Code length	2176			648			2592		
Lift. size Z	32			27			18		
$(E_b/N_0)_{WF}$	1.3 dB			2.5 dB			1.7 dB		
$(E_b/N_0)_{EF}$	2 dB			3.2 dB			2.4 dB		
Q_λ	5	6	7	5	6	7	5	6	7
α_{WF}	4	4.5	4.5	3.2	4.2	4.2	4	5.7	5.7
α_{EF}	3	5.2	4.2	2.5	2.5	2.5	2.5	5.7	4.2

$\{1, ..., \lceil \log_2(\text{MAX\_DC}) \rceil + 1\}$ respectively be the pipeline depth of these two blocks, where $\lceil . \rceil$ denotes the ceiling function. Therefore, the total pipeline depth of the LDPC decoder is: $d_{\texttt{DEC}} = d_{\texttt{2-MIN}} + 2 \times d_{\texttt{shift}} + 2$.

It is worth mentioning that the read from $\Lambda-\texttt{memory}$ and the $+\Delta$ computation must be performed during the same clock cycle. Otherwise, update conflicts may occur, and techniques to avoid them will be considered in future work. Therefore, the clock period will be ultimately limited by the propagation time of the Δ-update path.

IV. SIMULATIONS AND RESULTS

A. Performance Evaluation

This section aims to analyze the FER performance of the proposed decoder architecture when considering $Q_\lambda \in \{5, 6, 7\}$ and the three LDPC codes presented in Section II-A. To support all these cases, the decoder architecture is implemented with MAX_DC $= 26$, $Z_{\text{MAX}} = 32$, $it_{max} = 25$ (with ET), $N_{r,\text{MAX}} = 96$, $N_{\text{subVN}} = 8$, $Q_\mu = Q_\lambda$ and $Q_\Lambda = 8$. The pipeline depth $d_{\texttt{DEC}}$ is set to 15 for the 5GNR code and the $C_9 - Z_{18}$ codes, and to 4 for the 802.11n code.

For all the results presented in this section, two Monte-Carlo simulations are performed for each considered case. The first simulation considers a belief scaling parameter α_{WF} that minimizes the FER at a given E_b/N_0 value in the waterfall region (noted $(E_b/N_0)_{WF}$). The second simulation considers the optimized belief scaling parameter α_{EF} at a $(E_b/N_0)_{EF}$ value in the error floor region. All the code and decoding parameters are presented in Table I. For all the simulations, we have found the optimal offset to be $\beta = 1$.

Figure 2 shows the FER results for each code and each Q_λ. Each simulation curve is obtained by selecting, for each E_b/N_0 value, the best FER from the waterfall and error floor simulation setups. It can be observed that $Q_\lambda = 6$ and $Q_\lambda = 7$ achieve similar FER performance for the 5GNR and $C_9 - Z_{18}$ codes, and the FER is degraded when $Q_\lambda = 5$. Particularly, we observe a FER increase before the error floor. This can be explained by the fact that the belief scaling is not optimised in this region, and that this parameter has a strong influence when the number of quantization bits is too low. Therefore, α must be optimised for all E_b/N_0 values when case $Q_\lambda = 5$, requiring an accurate noise power estimator in practice. For the remaining of this section, $Q_\lambda = 6$ will be considered.

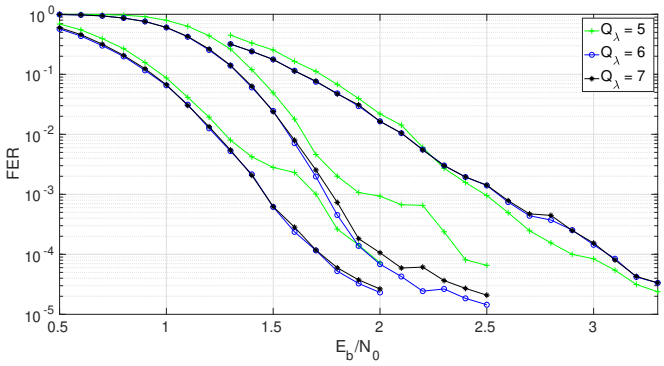

Fig. 2. FER simulation of the LDPC decoders with different Q_λ values and codes.

TABLE II
SYNTHESIS RESULTS FOR DIFFERENT PIPELINE DEPTH WITH $Q_\lambda = 6$.

Pipeline depth	5	7	9	13
Cell count ($\times 10^6$)	1.01	1.06	1.18	1.23
Cell area (mm^2)	4.31	4.34	5.00	5.38
Net area (mm^2)	1.80	1.87	2.01	2.07
Total area (mm^2)	6.12	6.22	7.01	7.45
Clock period (ns)	2	1.4	1.2	1.1
Estimated power (W)	1.32	2.07	2.87	3.63

B. Synthesis Results

The synthesis flow that we have developed to perform the synthesis of the reported architectures relies on the Cadence Genus™ tool and the TSMC65GP 65nm design kit. Table II compares the results obtained for different pipeline depth $d_{\texttt{DEC}}$, in terms of design complexity, clock period and dynamic power consumption. The clock period is a measure of the maximum speed achievable by the design. For each pipeline depth configuration, it is obtained by successively running the synthesis with incremented clock period targets, until the obtained slack is positive. The architecture parameters are the same as described in Section IV-A, with $Q_\lambda = 6$. The pipeline stage locations are chosen to balance the propagation delay between each stage.

Results show that the maximum clock frequency is almost doubled when comparing $d_{\texttt{DEC}} = 5$ and $d_{\texttt{DEC}} = 13$ cases. This increase in speed is achieved at the cost of area and power consumption: the cell count and the cell area are respectively increased by 21% and 25%, and the power consumption is increased by a factor of 2.75. When the pipeline depth reaches 13, the propagation path between each register becomes shorter than the Δ-update path discussed in Section III-B. Therefore, there is no reason to increase the pipeline depth above 13 as it would results in a waste of area and power consumption. The dynamic power consumption significantly increases with the pipeline depth, which is due to: *i)* the faster switching activity of the transistors, *ii)* increased area resulting from added pipeline registers. In the following section, we evaluate the trade-offs between speed and power consumption,

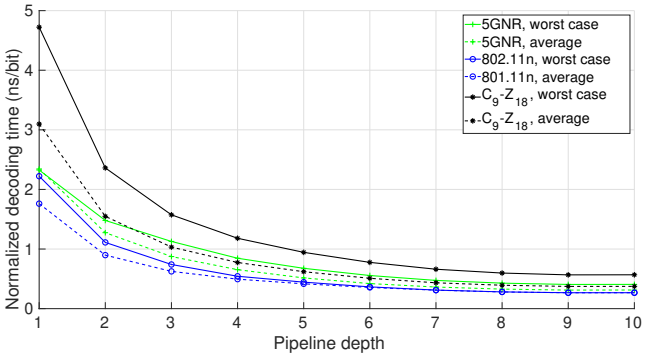

Fig. 3. Average and worst-case decoding time for different pipeline depths and codes, for a FER target of 10^{-2} at $(E_b/N_0)_{WF}$ values (given in Table I).

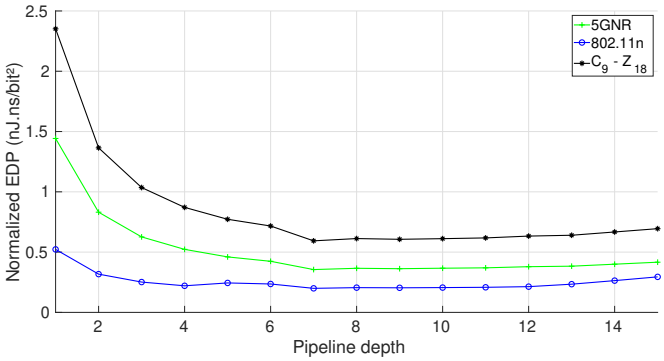

Fig. 4. Normalized EDP for different pipeline depths and codes, for a FER target of 10^{-2} at $(E_b/N_0)_{WF}$ values (given in Table I).

using the EDP and the decoding time metrics.

C. Pipeline Depth Analysis

The Δ-update mechanism described in Section II-C enables the use of architectures with deep pipeline. Higher clock frequency can be reached, at the cost of an increased number of decoding iteration, in-between the layered and the flooding schedules. In this section, the impact of the pipeline depth on the decoding time and the EDP is analyzed when considering the synthesis results obtained in IV-B.

We define the average normalized decoding time (NDT) as $\text{NDT}_{\text{avg}} = \tilde{n}_{\text{clk}} T_{\text{clk}}/L$, where $\tilde{n}_{\text{clk}}$ is the average number of clock cycle to perform a decoding (with ET), T_{clk} is the clock period of the circuit and L is the code length. The worst-case NDT is defined as $\text{NDT}_{\text{MAX}} = N_r t_{max} T_{\text{clk}}/L$. Note that $1/\text{NDT}_{\text{MAX}}$ gives the worst-case throughput, while $1/\text{NDT}_{\text{avg}}$ gives the best throughput that can be achieved by the architecture if inputs can be buffered. To measure EDP, we assume that ET is used to calculate the energy, while the delay corresponds to NDT_{MAX}. Therefore we have $\text{EDP} = \text{NDT}_{\text{avg}} \cdot \text{NDT}_{\text{MAX}} \cdot P_D$, where P_D is the dynamic power estimated by the synthesis tool.

Fig. 3 shows the average and worst-case NDT when the pipeline depth d varies from 1 to 15, for all the codes. These results are obtained for a 10^{-2} FER target at $(E_b/N_0)_{WF}$. Results are extrapolated for $d_{\text{DEC}} \leq 5$, assuming that the clock period perfectly scale with the pipeline depth. Through successive Monte-Carlo simulation, t_{max} is obtained when the FER target is reached while the ET mechanism is disabled. Then, the ET is activated to deduce $\tilde{n}_{\text{clk}}$. Pipeline stage locations have been chosen to minimize the critical path delay, and T_{clk} is obtained from the synthesis results presented in Section IV-B. It can be seen that increasing d_{DEC} always results in improved decoding times for all the codes. In fact, we observed that the clock period decreases faster than the increase in number of iterations. However, for very deep pipelines ($d_{\text{DEC}} \geq 9$), latency gains are marginal while penalizing the power and area. This is particularly visible in Fig. 4, where the EDP slowly increases when $d_{\text{DEC}} \geq 9$. The 5GNR and 802.11n codes cannot be pipelined with more than 1 stage when considering a parallel architecture with a strict row-layered schedule. Therefore, the Δ-update scheme with $d_{\text{DEC}} = 7$ reduces the EDP by 75% for the 5GNR code and 62% for the 802.11n code. The $C_9 - Z_{18}$ code is designed to support pipeline depth up to $d_{\text{DEC}} = 5$ [1], which results in a reduction of 23% of the EDP (when $d_{\text{DEC}} = 7$). This demonstrates that deeper pipelines can reduce the EDP.

V. Conclusion

A novel highly parallel LDPC decoder architecture is proposed. Thanks to its Δ update scheme, deep pipelines can be supported without update conflicts, at the cost of additional decoding iterations. This scheme also allows to easily support many different codes without complex control mechanisms. We observed that the ability to increase the pipeline depth allows a significant reduction of the decoding time and of the EDP. This demonstrate the efficiency of the Δ-update scheme to improve the processing speed and the EDP of highly-parallel LDPC decoders.

References

[1] E. Dupraz, F. Leduc-Primeau, and F. Gagnon, "Low-latency LDPC decoding achieved by code and architecture co-design," in *2018 IEEE 10th International Symposium on Turbo Codes Iterative Information Processing (ISTC)*, Dec 2018, pp. 1–5.

[2] Seho Myung, Kyeongcheol Yang, and Jaeyoel Kim, "Quasi-cyclic LDPC codes for fast encoding," *IEEE Trans. Inf. Theory*, vol. 51, no. 8, pp. 2894–2901, Aug 2005.

[3] S. Kim, G. E. Sobelman, and H. Lee, "a reduced-complexity architecture for LDPC layered decoding schemes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*.

[4] 3GPP TS 38.212 , "3rd generation partnership project; technical specification group radio access network; multiplexing and channel coding," https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3214.

[5] "IEEE Standard for Information technology, Local and metropolitan area networks, Part 11: Wireless LAN MAC and PHY Specifications," *IEEE Std 802.11n-2009*, pp. 1–565, 2009.

[6] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Commun.*

[7] X. Chen, S. Lin, and V. Akella, "QSN—a simple circular-shift network for reconfigurable quasi-cyclic LDPC decoders," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 57, no. 10, pp. 782–786, Oct 2010.

[8] F. Leduc-Primeau, F. R. Kschischang, and W. J. Gross, "Modeling and energy optimization of LDPC decoder circuits with timing violations," *IEEE Trans. on Commun.*, vol. 66, no. 3, pp. 932–946, March 2018.