# Analysis and Design of Cost-Effective, High-Throughput LDPC Decoders

Thien Truong Nguyen-Ly, Valentin Savin, Khoa Le, David Declercq, *Senior Member, IEEE*, Fakhreddine Ghaffari, and Oana Boncalo

***Abstract*— This paper introduces a new approach to cost-effective, high-throughput hardware designs for low-density parity-check (LDPC) decoders. The proposed approach, called nonsurjective finite alphabet iterative decoders (NS-FAIDs), exploits the robustness of message-passing LDPC decoders to inaccuracies in the calculation of exchanged messages, and it is shown to provide a unified framework for several designs previously proposed in the literature. NS-FAIDs are optimized by density evolution for regular and irregular LDPC codes, and are shown to provide different tradeoffs between hardware complexity and decoding performance. Two hardware architectures targeting high-throughput applications are also proposed, integrating both Min-Sum (MS) and NS-FAID decoding kernels. ASIC post synthesis implementation results on 65-nm CMOS technology show that NS-FAIDs yield significant improvements in the throughput to area ratio, by up to 58.75% with respect to the MS decoder, with even better or only slightly degraded error correction performance.**

***Index Terms*— Error correction, high throughput, low-density parity-check (LDPC) codes, low cost, nonsurjective finite alphabet iterative decoder (NS-FAID).**

## I. Introduction

THE increasing demand of massive data rates in wireless communication systems will require significantly higher processing speed of the baseband signal, as compared with conventional solutions. This is especially challenging for forward error correction (FEC) mechanisms, since FEC decoding is one of the most computationally intensive baseband processing tasks, consuming a large amount of hardware resources and energy. The use of very large bandwidths will also result in stringent, application-specific, requirements in terms of both throughput and latency. The conventional approach to increase throughput is to use massively parallel architectures. In this context, low-density parity-check (LDPC) codes are recognized as the foremost solution, due to the intrinsic capacity of their decoders to accommodate various degrees of parallelism. They have found extensive applications in modern communication systems, due to their excellent decoding performance, high-throughput capabilities [1]–[4], and power efficiency [5], [6], and have been adopted in several recent communication standards.

This paper targets the design of cost-effective, high-throughput LDPC decoders. One important characteristic of LDPC decoders is that the memory and interconnect blocks dominate the overall area/delay/power performance of the hardware design [7]. To address this issue, we build upon the concept of finite alphabet iterative decoders (FAIDs) introduced in [8]–[10]. While FAIDs have been previously investigated for variable-node (VN) regular LDPC codes over the binary symmetric channel, this paper extends their use to any channel model and to both regular and irregular LDPC codes.

The approach considered in this paper, referred to as nonsurjective finite FAIDs (NS-FAIDs), is to allow storing the exchanged messages using a lower precision (a smaller number of bits) than that used by the processing units. The basic idea is to reduce the size of the exchanged messages, once they have been updated by the processing units. Hence, to some extent, the proposed approach is akin to the use of imprecise storage, which is seen as an enabler for cost and throughput optimizations. Moreover, NS-FAIDs are shown to provide a unified framework for several designs previously proposed in the literature, including the normalized and offset Min-Sum (OMS) decoders [11], [12], the partially OMS (POMS) decoder [13], the MS-based decoders proposed in [14] and [15], or the recently introduced dual-quantization domain MS decoder [16].

This paper refines and extends some of the concepts we previously introduced in [17] and [18]. In particular, the definition of NS-FAIDs [17] is extended such as to cover a larger class of decoders, which is shown to significantly improve the decoding performance in case that the exchanged messages are quantized on a small number of bits (e.g., 2 bits per exchanged message). We show that NS-FAIDs can be optimized by using the density evolution (DE) technique, so as to obtain

Manuscript received April 10, 2017; revised July 13, 2017 and October 2, 2017; accepted November 4, 2017. Date of publication December 14, 2017; date of current version February 22, 2018. This work was supported in part by the European H2020 Work Program under Project Flex5Gware and in part by the Franco-Romanian (ANR-UEFISCDI) Joint Research Program Blanc-2013 under Project DIAMOND. *(Corresponding author: Valentin Savin.)*

T. T. Nguyen-Ly is with CEA-LETI, MINATEC Campus, 38054 Grenoble, France, and also with ETIS ENSEA/UCP/CNRS UMR-8051, 95014 Cergy-Pontoise, France (e-mail: thientruong.nguyen-ly@cea.fr).

V. Savin is with CEA-LETI, MINATEC Campus, 38054 Grenoble, France (e-mail: valentin.savin@cea.fr).

K. Le, D. Declercq, and F. Ghaffari are with ETIS ENSEA/UCP/CNRS UMR-8051, 95014 Cergy-Pontoise, France (e-mail: khoa.letrung@ensea.fr; declercq@ensea.fr; ghaffari@ensea.fr).

O. Boncalo is with the Computers and Information Technology Department, University Politehnica Timisoara, 300006 Timisoara, Romania (e-mail: boncalo@cs.upt.ro).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2776561

the best possible decoding performance for given hardware constraints, expressed in terms of memory size reduction. The DE optimization is illustrated for both regular and irregular LDPC codes, for which we propose a number of NS-FAIDs with different tradeoffs between hardware complexity and decoding performance.

To assess the benefits of the NS-FAID approach, we further extend the hardware architectures proposed in [18] to cover the case of irregular codes, and provide implementation results targeting an ASIC technology, which is more likely to reflect the benefits of the proposed NS-FAID approach in terms of throughput/area tradeOFF. The proposed architectures target high throughput and efficient use of the hardware resources. Both architectures implement layered decoding with fully parallel processing units. The first architecture is pipelined, so as to increase throughput and ensure an efficient use of the hardware resources, which in turn imposes specific constraints on the decoding layers,[1] in order to ensure proper execution of the layered decoding process. The second architecture does not make use of pipelining, but allows maximum parallelism to be exploited through the use of full decoding layers,[2] thus resulting in significant increase in throughput. Both MS and NS-FAID decoding kernels are integrated into each of the two proposed architectures, and compared in terms of area and throughput. ASIC postsynthesis implementation results on 65-nm CMOS technology show a throughput to area ratio improvement by up to 58.75%, when the NS-FAID kernel is used, with even better or only slightly degraded error correction performance.

The rest of this paper is organized as follows. NS-FAIDs are introduced in Section II, which also discusses their expected implementation benefits and the DE analysis. The optimization of regular and irregular NS-FAIDs is presented in Section III. The proposed hardware architectures, with both MS and NS-FAID decoding kernels, are discussed in Section IV. Numerical results are provided in Section V, and Section VI concludes this paper.

## II. Nonsurjective Finite Alphabet Iterative Decoders

### A. Preliminaries

LDPC codes are defined by sparse bipartite graphs, comprising a set of VNs, corresponding to coded bits, and a set of check nodes (CNs), corresponding to parity-check equations. FAIDs are message-passing LDPC decoders that have been introduced in [8]–[10]. We state below the definition of a subclass of FAID decoders, which is less general than the one proposed in [10]. Let $Q$ be a positive integer. A $(2Q+1)$-level FAID is a 4-tuple $(\mathcal{M}, \Gamma, \Phi_v, \Phi_c)$, where we have the following.

1) $\mathcal{M} = \{-Q, \ldots, -1, 0, +1, \ldots, +Q\}$ is the alphabet of the exchanged messages, and is also referred to as the decoder alphabet.
2) $\Gamma \subseteq \mathcal{M}$ is the input alphabet of the decoder, i.e., the set of all possible values of the quantized soft information supplied to the decoder.
3) $\Phi_v$ and $\Phi_c$ denote the update rules for VNs and CNs, respectively.

We will use $\mathfrak{m} \in \mathcal{M}$ and $\gamma \in \Gamma$ to denote the elements of $\mathcal{M}$ and $\Gamma$, respectively. The CN-update function $\Phi_c$ is the same for any FAID decoder, and is equal to the update function used by the MS decoder. Precisely, for a CN of degree $d_c$, the update function $\Phi_c : \mathcal{M}^{d_c-1} \to \mathcal{M}$ is given by

$$\Phi_c(\mathfrak{m}_1, \ldots, \mathfrak{m}_{d_c-1}) = \left(\prod_{i=1}^{d_c-1} \operatorname{sgn}(\mathfrak{m}_i)\right) \min_{i=1,\ldots,d_c-1} |\mathfrak{m}_i|. \quad (1)$$

The VN-update function $\Phi_v : \Gamma \times \mathcal{M}^{d_v-1} \to \mathcal{M}$, for a VN of degree $d_v$, is defined as

$$\Phi_v(\gamma, \mathfrak{m}_1, \ldots, \mathfrak{m}_{d_v-1}) = F\left(\gamma + \sum_{j=1}^{d_v-1} \mathfrak{m}_j\right) \quad (2)$$

where the function $F : \mathbb{Z} \to \mathcal{M}$ is defined based on a set of threshold values $\mathcal{T} = \{T_0, T_1, \ldots, T_{Q+1}\} \subset \bar{\mathbb{R}}_+$, with $T_0 = 0$, $T_{Q+1} = +\infty$, and $T_i < T_j$ for any $i < j$

$$F(x) = \operatorname{sgn}(x)\mathfrak{m}, \quad \text{where } \mathfrak{m} \text{ is s.t. } T_{\mathfrak{m}} \le |x| < T_{\mathfrak{m}+1}. \quad (3)$$

In (2), the variable $\gamma$ represents the *channel contribution*, i.e., the quantized soft information that has been supplied to the decoder for the corresponding VN. The quantization method and its impact on FAIDs' decoding performance will be discussed in Section II-E. It is also worth noting that in [10], FAIDs are introduced with a more general VN-update function $\Phi_v$, but the simpler definition (2) that we use has many hardware implementation benefits, which will be described in Section IV. Moreover, it can be easily seen that any nondecreasing odd function satisfies (3). Precisely, Proposition 1 holds.

*Proposition 1:* For any function $F : \mathbb{Z} \to \mathcal{M}$, there exists a threshold set $\mathcal{T}$ such that $F$ is given by (3), if and only if $F$ satisfies the following two properties.

1) $F$ is an odd function, i.e., $F(-x) = -F(x)$, $\forall x \in \mathbb{Z}$.
2) $F$ is nondecreasing, i.e., $F(x) \le F(y)$ for any $x < y$.

We note that Proposition 1 also implies that $F(0) = 0$ and $F(x) \ge 0, \forall x > 0$. In this paper, we further extend the definition of FAIDs by allowing $F(0)$ to take on nonzero values. To ensure symmetry of the decoder, we shall write $F(0) = \pm\lambda$, with $\lambda \ge 0$, meaning that $F(0)$ takes on either $-\lambda$ or $+\lambda$ with equal probability. In the following, $F$ will be referred to as *framing function*.

As the focus of this paper is on practical implementations, we will further assume that the sum $\gamma + \sum_{j=1}^{d_v-1} \mathfrak{m}_j$ in (2) is saturated to $\mathcal{M}$, prior to applying $F$ on it. Consequently, in the sequel, we shall only consider framing functions $F : \mathcal{M} \to \mathcal{M}$, and the VN-update function $\Phi_v$ from (2) is redefined as

$$\Phi_v(\gamma, \mathfrak{m}_1, \ldots, \mathfrak{m}_{d_v-1}) = F\left(s_{\mathcal{M}}\left(\gamma + \sum_{j=1}^{d_v-1} \mathfrak{m}_j\right)\right) \quad (4)$$

[1] A decoding layer may consist of one or several rows of the base matrix of the quasi-cyclic (QC)-LDPC code, assuming that they do not overlap.

[2] A decoding layer is said to be full if each column of the base matrix has one nonnegative entry in one of the rows composing the layer.

**Algorithm 1** FAID Decoding With Framing Function $F$

Input: $\boldsymbol{y} = (y_1, \ldots, y_N)$ ▷ received word
Output: $\hat{\underline{\boldsymbol{x}}} = (\hat{x}_1, \ldots, \hat{x}_N)$ ▷ estimated codeword
**Initialization**
**for all** $n = 1, \ldots, N$ **do** $\gamma_n = \text{quant}\,(\text{LLR}(x_n \mid y_n))$;
**for all** $m = 1, \ldots, M$ and $n \in H(m)$ **do** $\beta_{m,n} = 0$;
**Iteration Loop**
**for all** $n = 1, \ldots, N$ and $m \in H(n)$ **do** ▷ **VN-processing**
$\alpha_{m,n} = \Phi_v\left(\gamma_n, \beta_{m',n} \mid m' \in H(n) \setminus m\right)$;
**for all** $m = 1, \ldots, M$ and $n \in H(m)$ **do** ▷ **CN-processing**
$\beta_{m,n} = \Phi_c\left(\alpha_{m,n'} \mid n' \in H(m) \setminus n\right)$;
**for all** $n = 1, \ldots, N$ **do** ▷ **AP-update**
$\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$;
**for all** $n = 1, \ldots, N$ **do** $\hat{x}_n = \text{sign}(\tilde{\gamma}_n)$; ▷ hard decision
**if** $\hat{\underline{\boldsymbol{x}}}$ is a codeword **then** exit the iteration loop ▷ syndrome check
**End Iteration Loop**

Notation: $H$ – bipartite graph of the LDPC code, with $N$ VNs and $M$ CNs; $H(n)$ – set of CNs connected to VN $n$; $H(m)$ – set of VNs connected to CN $m$; $\alpha_{m,n}$ – message from VN $n$ to CN $m$; $\beta_{m,n}$ – message from CN $m$ to VN $n$; $\gamma_n$ and $\tilde{\gamma}_n$ – a priori and a posteriori LLR of VN $n$; quant – quantization map (see also Section II-E).

where $s_{\mathcal{M}} : \mathbb{Z} \to \mathcal{M}$, $s_{\mathcal{M}}(x) = \text{sgn}(x)\min(|x|, Q)$, is the saturation function. Since $F(-\mathfrak{m}) = -F(\mathfrak{m}), \forall \mathfrak{m} \in \mathcal{M}$, $F$ is completely determined by the vector $[|F(0)|, F(1), \ldots, F(Q)]$, further referred to as the lookup table (LUT) of $F$, which satisfies the following inequalities (Proposition 1):

$$0 \le |F(0)| \le F(1) \le \cdots \le F(Q) \le Q. \tag{5}$$

Summarizing, the subclass of FAIDs considered in this paper is defined by (4), where $F$ is a framing function satisfying (5). Furthermore, for any integer $q > 0$, the expression *$q$-bit FAID* is used to refer to a $(2Q+1)$-level FAID, with $Q = 2^{q-1} - 1$. It follows that messages exchanged within the FAID decoder are $q$-bit messages (including 1 bit for the sign). Finally, the message-passing iterative decoding process for an FAID with framing function $F$ is shown in Algorithm 1.

### B. Nonsurjective FAIDs

The finite alphabet MS decoder is a particular example of FAID, with framing function $F : \mathcal{M} \to \mathcal{M}$ being the identity function. Using (5), it can be easily verified that this is the only FAID for which the framing function $F$ is surjective (or equivalently bijective, since $\mathcal{M}$ is finite). For any other framing function $F$, there exists at least one element of $\mathcal{M}$, which is not in the image of $F$. The class of FAIDs defined by nonsurjective framing functions[3] is investigated in this section.

[3]Although we restrict the study in this paper to nonsurjective framing functions $F : \mathcal{M} \to \mathcal{M}$, it can be readily extended to nonsurjective framing functions $F : \mathbb{Z} \to \mathcal{M}$. While such an extension would widen the class of NS-FAIDs, our preliminary investigations have shown that the best NS-FAIDs are actually those within the subclass of NS-FAIDs defined by $F : \mathcal{M} \to \mathcal{M}$, investigated in this paper.

TABLE I
EXAMPLES OF 4-bit FRAMING FUNCTIONS OF WEIGHT $W = 4$

| $\mathfrak{m}$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $F_1(\mathfrak{m})$ | 0 | 1 | 1 | 3 | 3 | 3 | 7 | 7 |
| $F_2(\mathfrak{m})$ | $\pm 1$ | 1 | 1 | 3 | 3 | 4 | 4 | 7 |

*Definition 2:* The *weight* of a framing function $F : \mathcal{M} \to \mathcal{M}$, denoted by $W$, is the number of distinct entries in the vector $[|F(0)|, F(1), \ldots, F(Q)]$. It follows that $1 \le W \le Q+1$. By a slight abuse of terminology, we shall also refer to $W$ as the weight of the NS-FAID. We further define the *framing bit-length* as $\omega = \lceil \log_2(W) \rceil + 1$.

*Definition 3:* An NS-FAID is an FAID of weight $W < Q+1$. Hence, the framing function $F$ is nonsurjective, meaning that the image set of $F$ is a strict subset of $\mathcal{M}$.

Table I provides two examples of $q = 4$-bit NS-FAIDs (hence $Q = 7$), both of which are of weight $W = 4$, hence framing bit-length $\omega = \lceil \log_2 4 \rceil + 1 = 3$. Note that $F_1$ maps 0 to 0, while $F_2$ maps 0 to $\pm 1$. The image sets of $F_1$ and $F_2$ are $\text{Im}(F_1) = \{0, \pm 1, \pm 3, \pm 7\}$ and $\text{Im}(F_2) = \{\pm 1, \pm 3, \pm 4, \pm 7\}$.

The main motivation for the introduction of NS-FAIDs is that they allow reducing the size of the memory required to store the exchanged messages. Clearly, for an NS-FAID with framing bit-length $\omega$, the exchanged messages can be represented using only $\omega$ instead of $q$ bits (including 1 bit for the sign). Moreover, as a consequence of the message size reduction, the size of the interconnect network that carries the messages from the memory to the processing units is also reduced.

*Proposition 4:* The number of $(2Q+1)$-level NS-FAIDs of weight $W$ is given by

$$N_{\text{NS-FAID}}(Q, W) = \binom{Q}{W-1}\binom{Q+1}{W} \tag{6}$$

where $\binom{y}{x}$ denotes the binomial coefficient.

### C. Examples of NS-FAIDs

As mentioned above, if the framing function $F$ is the identity function, then the corresponding FAID is equivalent to the MS decoder with finite alphabet $\mathcal{M}$. Some examples of NS-FAIDs are provided in the following.

*Example 1:* Let $F : \mathcal{M} \to \mathcal{M}$ be defined by

$$F(\mathfrak{m}) = \text{sgn}(\mathfrak{m})\max(|\mathfrak{m}| - \theta, 0) \tag{7}$$

where $\theta \in \{1, \ldots, Q-1\}$. Then, the corresponding NS-FAID is the OMS decoder with offset factor $\theta$.

*Example 2:* Let $F : \mathcal{M} \to \mathcal{M}$ be defined by

$$F(\mathfrak{m}) = \begin{cases} \mathfrak{m}, & \text{if } |\mathfrak{m}| \text{ is even} \\ \text{sgn}(\mathfrak{m})(|\mathfrak{m}| - 1), & \text{if } |\mathfrak{m}| \text{ is odd.} \end{cases} \tag{8}$$

Then, the corresponding NS-FAID is the POMS decoder from [13].

Moreover, it can be seen that the MS-based decoders proposed in [14] and [15] and the dual-quantization domain decoder proposed in [16] are particular realizations of NS-FAIDs.

While the main reason behind the NS-FAIDs definition consists in their ability to reduce memory and interconnect requirements, we can also argue that they may allow improving the error correction performance (with respect to MS). This is the case of both OMS and POMS decoders mentioned above. Given a target message bit-length $w$ (e.g., corresponding to some specific memory constraint), one may try to find the framing function $F$ of corresponding weight $W$, which yields the best error correction performance. The optimization of the framing function can be done by using the DE technique, which will be discussed in Section II-E.

Since $F$ is a nondecreasing function, it can be shown that the framing function $F$ can alternatively be applied at the CN-processing step (instead of VN-processing), while resulting in an equivalent decoding algorithm. Whether $F$ is applied at the VN-processing or the CN-processing step is rather a matter of implementation. When $F$ is applied at the VN-processing step, both VN- and CN-messages belong to a strict subset of the alphabet $\mathcal{M}$, namely, $\mathcal{M}' = \mathrm{Im}(F) \subset \mathcal{M}$. When $F$ is applied at the CN-processing step, only CN-messages belong to $\mathcal{M}'$. However, it is worth noting that many hardware implementations of QC LDPC decoders rely on a layered architecture, which only requires storing the CN messages [7].

### D. Irregular NS-FAIDs

In case of irregular LDPC codes, irregular NS-FAIDs are NS-FAIDs using different framing functions $F_{d_v}$ for VNs of different degrees $d_v$. Framing functions $F_{d_v}$ may have different weights $W_{d_v}$. In this case, messages outgoing from degree-$d_v$ VNs can be represented by using only $w_{d_v} = \lceil \log_2(W_{d_v}) \rceil + 1$ bits. However, the message size reduction does not necessarily apply to CN-messages, due to the fact that a CN may be connected to VNs of different degrees. Let $\mathcal{M}'_{d_v} = \mathrm{Im}(F_{d_v})$. Then, messages outgoing from a CN $c$ can be represented by using

$$\left\lceil \log_2 \left( \left| \cup_{d_v \in \mathcal{D}_c} \mathcal{M}'_{d_v} \right| \right) \right\rceil \text{ bits} \tag{9}$$

where $\mathcal{D}_c$ is the set of degrees of VNs connected to $c$, and $|\cdot|$ is used to denote the number of elements of a set.

Alternatively, it is also possible to define CN-irregular NS-FAIDs in a similar manner. However, in this paper, we only deal with VN-irregular NS-FAIDs, since most of the practical irregular LDPC codes are irregular on VNs, while almost regular (or semiregular) on CNs. In order to reduce the size of the CN-messages, in Section III-B, we will further impose certain conditions on the framing functions $F_{d_v}$, by requiring their images being included in one another.

### E. Density Evolution Analysis

For the sake of simplicity, we only consider transmission over binary-input memoryless noisy channels. We assume that the channel input alphabet is $\mathcal{X} = \{-1, +1\}$, with the usual convention that $+1$ corresponds to 0-bit and $-1$ corresponds to 1-bit, and denote by $\mathcal{Y}$ the output alphabet of the channel. We denote by $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ the transmitted and received symbols, respectively.

We further consider a function $\varphi : \mathcal{Y} \to \Gamma$ that maps the output alphabet of the channel to the input alphabet of the decoder, and set $\gamma = \varphi(y)$. Hence, $\varphi$ encompasses both the computation of the soft (unquantized) log-likelihood ratio (LLR) value and its quantization. We shall refer to $\varphi$ as quantization map and to $\gamma$ as the input LLR of the decoder.

For transmission over the binary-input additive white Gaussian noise (AWGN) channel, we shall consider that the decoder's input information as well as the exchanged messages are quantized on the same number of bits; therefore $\Gamma = \mathcal{M}$ unless otherwise stated. In this case, $y = x + z$, where $z$ is the white Gaussian noise with variance $\sigma^2$, and the quantization map $\varphi : \mathcal{Y} \to \mathcal{M}$ is defined by

$$\varphi(y) = [\mu \cdot y]_{\mathcal{M}} \tag{10}$$

where $\mu > 0$ is a constant referred to as gain factor and $[x]_{\mathcal{M}}$ denotes the closest integer to $x$ that belongs to $\mathcal{M}$ (see also [19] and the gain factor quantizer defined therein).

The objective of the DE technique is to recursively compute the probability mass function of the exchanged messages, through the iterative decoding process. This is done under the assumption that exchanged messages are independent, which holds in the asymptotic limit of the code length. In this case, the decoding performance converges to the cycle free case. DE equations for the NS-FAID decoder can be derived in a similar way as for the finite-alphabet MS decoder [19]. Similar to [19], the DE is used to compute the asymptotic error probability, defined as

$$p_e^{(+\infty)} = \lim_{\ell \to +\infty} p_e^{(\ell)} \tag{11}$$

where $p_e^{(\ell)}$ is the bit error probability at iteration $\ell$.

For a target bit error probability $\eta > 0$, the $\eta$-threshold is defined as the worst channel condition for which decoding error probability is less than $\eta$. Assuming the binary-input AWGN channel model, the $\eta$-threshold corresponds to the maximum noise variance $\sigma^2$ (or equivalently minimum SNR), such that the asymptotic error probability is less than $\eta$

$$\sigma^2_{\text{thres}}(\eta) = \sup\{\sigma^2 \mid p_e^{(+\infty)} \leq \eta\}. \tag{12}$$

In case that $\eta = 0$, the $\eta$-threshold is simply referred to as DE threshold [20]. However, the asymptotic decoding performance of finite-precision MS-based decoders is known to exhibit an error floor phenomenon at high SNR [19]. This makes the $\eta$-threshold definition more appropriate in practical cases, when the target bit-error rate (BER) can be fixed to a practical nonzero value.

Finally, it is worth noting that the $\sigma^2_{\text{thres}}(\eta)$ value depends on: 1) the irregularity of the LDPC code, parametrized as usual by the degree distribution polynomials $\lambda(x)$ and $\rho(x)$ [20]; 2) the NS-FAID, i.e., the size of the decoder alphabet and the framing function $F$; and 3) the channel quantizer $\varphi$ or equivalently the gain factor $\mu$ used in (10). Therefore, assuming that the degree distribution polynomials $\lambda(x)$ and $\rho(x)$ and the size of the decoder alphabet are fixed, we use the DE technique to jointly optimize the framing function $F$ and channel quantizer.

TABLE II
BEST NS-FAIDs FOR $(3, 6)$-REGULAR LDPC CODES

| | | F | SNR-thres (dB) |
|---|---|---|---|
| $w = 4$ | MS | $[0, 1, 2, 3, 4, 5, 6, 7]$ | $1.643$ $(\mu = 5.6)$ |
| $w = 3$ | $F(0) = 0$ | $[\mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{3}, \mathbf{3}, \mathbf{3}, \mathbf{7}, \mathbf{7}]$ | $\mathbf{1.409}$ $(\mu = \mathbf{3.8})$ |
| | $F(0) = \pm 1$ | $[\pm 1, 1, 1, 3, 3, 4, 4, 7]$ | $1.412$ $(\mu = 5.1)$ |
| | $F(0) = \pm 2$ | $[\pm 2, 2, 2, 3, 3, 3, 4, 7]$ | $1.712$ $(\mu = 7.1)$ |
| | $F(0) = \pm 3$ | $[\pm 3, 3, 3, 3, 3, 4, 5, 7]$ | $2.227$ $(\mu = 10.0)$ |
| $w = 2$ | $F(0) = 0$ | $[0, 0, 0, 0, 0, 6, 6, 6]$ | $2.251$ $(\mu = 8.6)$ |
| | $F(0) = \pm 1$ | $[\pm \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{6}, \mathbf{6}, \mathbf{6}]$ | $\mathbf{1.834}$ $(\mu = \mathbf{6.4})$ |
| | $F(0) = \pm 2$ | $[\pm 2, 2, 2, 2, 2, 2, 2, 7]$ | $1.911$ $(\mu = 8.3)$ |
| | $F(0) = \pm 3$ | $[\pm 3, 3, 3, 3, 3, 3, 3, 7]$ | $2.014$ $(\mu = 9.4)$ |

## III. DENSITY EVOLUTION OPTIMIZATION OF NS-FAIDs

Throughout this section, we consider $q = 4$-bit NS-FAIDs (hence, $Q = 7$). To illustrate the tradeoff between hardware complexity and decoding performance, we consider the optimization of both regular and irregular NS-FAIDs.

### A. Optimization of Regular NS-FAIDs

In this section, we consider the optimization of regular NS-FAIDs for ($d_v = 3$ and $d_c = 6$)-regular LDPC codes. We consider $q = 4$-bit NS-FAIDs, with framing bit-length parameter $w \in \{2, 3\}$. According to Proposition 4, the number of NS-FAIDs is given by $N_{\text{NS-FAID}}(q = 4, w = 3) = N_{\text{NS-FAID}}(Q = 7, W = 4) = \binom{7}{3}\binom{8}{4} = 2450$, and $N_{\text{NS-FAID}}(q = 4, w = 2) = N_{\text{NS-FAID}}(Q = 7, W = 2) = \binom{7}{1}\binom{8}{2} = 196$.

All regular NS-FAIDs have been evaluated by using the DE technique from Section II-E. Table II summarizes the best NS-FAIDs according to $w$ and $F(0)$ values,[4] for $0 \leq |F(0)| \leq 3$; DE thresholds ($\eta = 0$) and corresponding gain factors ($\mu$) are also reported. Best NS-FAIDs for $w = 2$ and $w = 3$ are emphasized in bold. For comparison purposes, the DE threshold of the $q = 4$-bit MS decoder is also reported: MS threshold is equal to 1.643 dB, for $\mu = 5.6$. For $w = 3$, it can be observed that best NS-FAIDs with $F(0) = 0$ or $F(0) = \pm 1$ have better DE thresholds than the 4-bit MS decoder. The best NS-FAID is given by the framing function $F = [0, 1, 1, 3, 3, 3, 7, 7]$ and its DE threshold is equal to 1.409 dB ($\mu = 3.8$), representing a gain of 0.23 dB compared with 4-bit MS. For $w = 2$, the best NS-FAID is given by the framing function $F = [\pm 1, 1, 1, 1, 1, 6, 6, 6]$ and its DE threshold is equal to 1.834 dB ($\mu = 6.4$), which represents a performance loss of only 0.19 dB compared with 4-bit MS. To emphasize the benefits of the proposed NS-FAIDs extension, we note that that for $w = 2$, best NS-FAIDs with $F(0) = \pm 1$, $F(0) = \pm 2$, or $F(0) = \pm 3$ have better DE thresholds than the best NS-FAIDs with $F(0) = 0$. The latter is given by the framing function $F = [0, 0, 0, 0, 0, 6, 6, 6]$ and its DE threshold is equal to 2.251 dB ($\mu = 8.6$), thus resulting in a performance loss of approximately 0.61 dB compared with 4-bit MS.

[4]NS-FAIDs with $|F(0)| > 3$ have worse DE thresholds, and thus they have not been included in the table.

### B. Optimization of Irregular NS-FAIDs

As a case study, we consider the optimization of irregular NS-FAIDs for the WiMAX irregular LDPC codes with rate $1/2$ [21] (of course, the proposed method can be applied to any other irregular codes in the same manner). The edge-perspective degree distribution polynomials are given by $\lambda(x) = 0.2895x + 0.3158x^2 + 0.3947x^5$ and $\rho(x) = 0.6316x^5 + 0.3684x^6$. Hence, VNs are of degree $d_v \in \{2, 3, 6\}$. For each VN-degree $d_v$, we consider that the corresponding framing function $F_{d_v}$ may be of any weight $W_{d_v} \in \{2, 4, 8\}$, corresponding to a framing bit-length $w_{d_v} \in \{2, 3, 4\}$. Hence, the total number of framing functions is given by $N_{\text{NS-FAID}}(7, 2) + N_{\text{NS-FAID}}(7, 4) + N_{\text{NS-FAID}}(7, 8) = 2645$ (see Proposition 4). Since a different framing function may be applied for each VN-degree, it follows that the total number of irregular NS-FAIDs is equal to $2645^3 = 18\,504\,486\,125$. Clearly, even though we rely on DE, it is practically impossible to evaluate the decoding performance of all the irregular NS-FAIDs. To overcome this problem, we proceed as described in the following.

*1) Optimization Procedure:* First, we evaluate the DE thresholds of NS-FAIDs applying one and the same framing function to all the VNs, irrespective of their degree, which for simplicity will be referred to as uniform NS-FAIDs throughout this section. Uniform NS-FAIDs with framing bit-length $w = \{2, 3, 4\}$ are then sorted with increasing DE threshold value, from the best to the worst decoder. Note that the case $w = 4$ represents a slight abuse of terminology, since there is only one such decoder, corresponding to the original MS decoder. We further denote by $\mathcal{U}^{(\text{best})}$-NS-FAID-$w$ the set of best uniform NS-FAIDs with framing bit-length $w$, determined as follows.

1) $\mathcal{U}^{(\text{best})}$-NS-FAID-2 is comprised of the uniform NS-FAIDs with $w = 2$, whose DE threshold is less than or equal to 5 dB; this represents 121 decoders out of the total of $N_{\text{NS-FAID}}(Q = 7, w = 2) = 196$ decoders.
2) $\mathcal{U}^{(\text{best})}$-NS-FAID-3 is comprised of the uniform NS-FAIDs with $w = 3$, whose DE threshold is less than or equal to 3 dB; this represents 946 decoders out of the total of $N_{\text{NS-FAID}}(Q = 7, w = 3) = 2450$ decoders.
3) $\mathcal{U}^{(\text{best})}$-NS-FAID-4 is comprised of the MS decoder only; its DE threshold is equal to 1.374 dB.

The limiting values of the DE thresholds for $\mathcal{U}^{(\text{best})}$-NS-FAID-2 and $\mathcal{U}^{(\text{best})}$-NS-FAID-3 are chosen such that the number of selected NS-FAIDs in each set is small enough.

For irregular NS-FAIDs, we denote by NS-FAID-$w_2w_3w_6$ the ensemble of NS-FAIDs defined by a triplet of framing functions ($F_2$, $F_3$, and $F_6$), corresponding to VN degrees $d_v = 2, 3, 6$, with framing bit-lengths $w_2, w_3, w_6$. Since $w_2, w_3, w_6 \in \{2, 3, 4\}$, there are 27 such ensembles. Since the number of NS-FAIDs in these ensembles can be very large, we only evaluate part of them, by further imposing the following two constraints.

*a) Decoding performance constraint:* We only consider irregular NS-FAIDs defined by triplets of framing functions

TABLE III
HARDWARE COMPLEXITY VERSUS DECODING PERFORMANCE TRADEOFF FOR OPTIMIZED IRREGULAR NS-FAIDS

| NS-FAIDs Ensemble | Framing functions applied to | | | SNR-thres (dB) (& gain factor $\mu$) @BER $= 10^{-6}$ | SNR gain/loss (+/− dB) | Memory size reduction (%) | | |
|---|---|---|---|---|---|---|---|---|
| | $d_v = 2$ | $d_v = 3$ | $d_v = 6$ | | | VN-mess. | CN-messages | |
| | | | | | | | uncomp. | comp. |
| NS-FAID-444 (MS) | LUT0 | LUT0 | LUT0 | 1.374 ($\mu$=3.2) | 0.000 | 0.00 | 0.00 | 0.00 |
| NS-FAID-432 | LUT0 | LUT1 | LUT6 | 1.188 ($\mu$=3.0) | +0.186 | −27.63 | 0.00 | 0.00 |
| NS-FAID-433 | LUT0 | LUT3 | LUT2 | 1.015 ($\mu$=2.8) | +0.359 | −17.76 | 0.00 | 0.00 |
| NS-FAID-332 | LUT4 | LUT3 | LUT6 | 1.273 ($\mu$=2.6) | +0.101 | −34.87 | −25.00 | −13.04 |
| NS-FAID-333 | LUT4 | LUT4 | LUT3 | 1.110 ($\mu$=2.4) | +0.264 | −25.00 | −25.00 | −13.04 |
| NS-FAID-222 | LUT7 | LUT5 | LUT5 | 2.299 ($\mu$=2.3) | −0.925 | −50.00 | −50.00 | −26.09 |

TABLE IV
LUTs USED BY NS-FAIDs IN TABLE III

| m | LUT0 | LUT1 | LUT2 | LUT3 | LUT4 | LUT5 | LUT6 | LUT7 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | ±1 | ±1 | ±1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 3 | 2 | 2 | 3 | 3 | 1 | 1 | 5 |
| 4 | 4 | 3 | 2 | 3 | 3 | 5 | 7 | 5 |
| 5 | 5 | 3 | 7 | 3 | 7 | 5 | 7 | 5 |
| 6 | 6 | 7 | 7 | 7 | 7 | 5 | 7 | 5 |
| 7 | 7 | 7 | 7 | 7 | 7 | 5 | 7 | 5 |
| $w$ | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |

($F_2$, $F_3$, and $F_6$), such that $F_{d_v} \in \mathcal{U}^{(\text{best})}$-NS-FAID-$w_{d_v}$, for any $d_v \in \{2, 3, 6\}$.

*b) Memory size reduction constraint:* We further impose the following inclusion constraint between the image sets of framing functions used for different VN-degrees. Let $w^{(\max)} = \max_{d_v} w_{d_v}$ and $d_v^{(\max)} = \text{argmax}_{d_v} w_{d_v}$. We impose that $\text{Im}(F_{d_v}) \subseteq \text{Im}(F_{d_v^{(\max)}})$, $\forall d_v \in \{2, 3, 6\}$. According to (9), this constraint ensures that CN-messages can be represented by using only $w^{(\max)}$ bits, which is particularly suitable for layered architectures.

The number of irregular NS-FAIDs that satisfy the above two constraints is equal to $N_{\text{NS-FAIDs}}^{\text{irregular}} = 7\,017\,762$.

*2) Density Evolution Evaluation:* For each of the $N_{\text{NS-FAIDs}}^{\text{irregular}}$ irregular NS-FAIDs, we compute its decoding threshold for a target BER $\eta = 10^{-6}$, using the DE technique from Section II-E. The threshold computation also encompasses the optimization of the channel gain factor $\mu$. Hence, for each NS-FAID, we first determine the gain factor $\mu$ that maximizes the $\eta$-threshold defined in (12). The corresponding $\eta$-threshold value is then reported as the $\eta$-threshold of the NS-FAID.

DE results for the MS decoder (indicated as NS-FAID-444), as well as for the NS-FAIDs with the best $\eta$-thresholds from five NS-FAID-$w_2w_3w_6$ ensembles, are shown in Table III: the framing functions used for VN-degrees $d_v = 2, 3, 6$ are shown in columns 2–4, while the $\eta$-threshold value (in dB) and the corresponding gain factor $\mu$ are shown in column 5. The SNR gain (+) or loss (−) reported in column 6 corresponds to the differences between the SNR threshold of the MS decoder (NS-FAID-444) and the SNR threshold of the best NS-FAID-$w_2w_3w_6$. The memory size reduction of the NS-FAID-$w_2w_3w_6$ decoders compared with the MS decoder is reported in columns 7–9, for both VN and CN messages. For CN messages, two possibilities are considered, according to whether they are stored in an uncompressed or compressed format, where compressed format means that only the signs, first minimum, second minimum, and index of the first minimum are stored [22]. Finally, the framing functions' LUTs of the best NS-FAID-$w_2w_3w_6$ decoders are reported in Table IV.

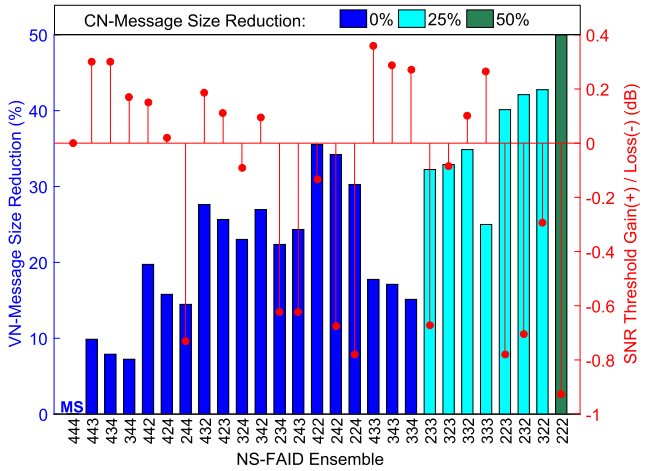



Fig. 1. Memory size reduction versus decoding performance.

Fig. 1 captures the tradeoff between decoding performance and memory size reduction, for each of the 27 NS-FAID-$w_2w_3w_6$ ensembles. For each ensemble, we select the NS-FAID with the best threshold, and indicate the corresponding memory gains and decoding performance. The height of vertical bars indicates the VN memory size reduction (values on the left vertical axis), while their color indicates the CN memory size reduction (uncompressed CN message storage is assumed in the legend). The red stems indicate the SNR threshold gain or loss compared with MS decoder (values on the right vertical axis). It can be seen that the NS-FAID-332 decoder allows a significant memory size reduction for both VN and CN messages, while still performing 0.1 dB ahead the MS decoder. The NS-FAID-433 decoder is also a very good candidate for applications requiring increased decoding performance: it achieves the best SNR gain (0.36 dB), while providing a VN memory size reduction by 17.76% with respect to the MS decoder.

Finally, it is worth noting that the reported memory size reductions do not necessarily translate as such in hardware

implementations, for several reasons. First, depending on the hardware architecture, VN messages may or may not be stored in a dedicated memory. For instance, layered architectures only require the storage of CN messages, which can be further stored in either a compressed or uncompressed format. Moreover, VN processing units (VNUs) need to be equipped with a framing and a deframing module, which may offset part of the promised gains. This is even more true in case of irregular codes, for which some VNUs may need to implement more than one single framing function, since the same VNU may be reused to process VN of different degrees (except for fully parallel architectures). To assess the gains of NS-FAIDs in practical implementations, the integration of the NS-FAID mechanism (framing/deframing modules) into two different MS decoder architectures is discussed in Section IV, and ASIC synthesis results for the CMOS 65-nm process technology are provided in Section V.

## IV. Hardware Architectures for NS-FAID Decoders

In this section, we propose two layered decoder architectures for QC-LDPC codes, with both MS and NS-FAIDs decoding kernels. Proposed architectures target high throughput, while ensuring an efficient use of the hardware resources. Two possible approaches to achieve high throughput are explored, consisting in either pipelining the data path or increasing the hardware parallelism.

We consider a QC-LDPC code defined by a base matrix $B$ of size $R \times C$, and expansion factor $z$, corresponding to a parity check matrix $H$ of size $M \times N$, with $M = zR$ and $N = zC$. With the notation from Section II-E, we denote by $\underline{\boldsymbol{x}} = (x_1, \ldots, x_N) \in \mathcal{X}^N$ the transmitted codeword, by $\underline{\boldsymbol{y}} = (y_1, \ldots, y_N) \in \mathcal{Y}^N$ the received word, and by $(\gamma_1, \ldots, \gamma_N)$ the input LLRs of the decoder, where $\gamma_n = \varphi(y_n)$ and $\varphi$ is the quantization map. VN and CN messages are denoted by $\alpha_{m,n}$ and $\beta_{m,n}$, respectively, and the *a posteriori* (AP)-LLR of a VN $n$ is denoted by $\tilde{\gamma}_n$. Hence, $\tilde{\gamma}_n = \gamma_n + \sum_{m \in H(n)} \beta_{m,n}$, where the sum is taken over all the CNs $m$ connected to $n$, denoted by $m \in H(n)$. Input LLRs and exchanged messages are quantized on $q$-bits, while AP-LLRs are assumed to be quantized on $\tilde{q}$ bits, with $\tilde{q} > q$.

A decoding layer (or simply referred to as a layer) consists of one or several consecutive rows of $B$, assuming that they do not overlap, i.e., each column of $B$ has at most one nonnegative entry within each layer. It is assumed that the same number of rows of $B$ participates in each decoding layer, which is denoted by RPL (rows per layer). Hence, the number of decoding layers is given by $L = R/\text{RPL}$. We further define $Z = z \times \text{RPL}$, corresponding to the number of rows of $H$ (parity checks) within one decoding layer, and referred to as the parallelism degree of the hardware architecture. To ease the description of the hardware architectures proposed in this section, we shall assume that all CNs have the same degree, denoted by $d_c$. However, no assumptions are made concerning VN degrees. We present each architecture assuming the MS decoding kernel is being implemented, and then we discuss the required changes in order to integrate the NS-FAID decoding kernel.

### A. Pipelined Architecture

The proposed architecture with MS decoding kernel is detailed in Fig. 2. A high-level representation is also shown in Fig. 5(a), for both MS and NS-FAID decoding kernels. The architecture is optimized so as to reduce the critical path. In particular, we completely reorganize the interconnect network (barrel shifters BS_INIT and BS_R, see in the following), thus removing the need for a barrel shifter on the writing data back side. The main blocks of the architecture are discussed as follows.

*1) Input/Output Buffers:* The input buffer, implemented as a number of serial input parallel output shift registers, is used to store the input LLR values ($\gamma_n$) received by the decoder. The output buffer is used to store the hard bit estimates of the decoded word. Input–output buffers allow data load/offload operations to take place concomitantly with the decoding of the current code word.

*2) Memory Blocks:* Two memory blocks are used, one for AP-LLR values ($\tilde{\gamma}$_memory) and one for CN-messages ($\beta$_memory). $\tilde{\gamma}$_memory is implemented by registers, in order to allow massively parallel read or write operations. It is organized in $C$ blocks, denoted by $AP_j$ $(j = 1, \ldots, C)$, corresponding to the columns of the base matrix, each one consisting of $z \times \tilde{q}$ bits. Data are read from/written to blocks corresponding to nonnegative entries in the decoding layer being processed. $\beta$_memory is implemented as a dual port random access memory (RAM), in order to support pipelining, as explained in the following. Each memory word consists of $Z \times \beta$_messages, corresponding to one decoding layer. Depending on the CN unit (CNU) implementation, $\beta$_messages can be either "uncompressed" (i.e., for a CN $m$, the corresponding $\beta$_message is given by the $d_c$ values $[\beta_{m,n_1}, \ldots, \beta_{m,n_{d_c}}]$, where $n_1, \ldots, n_{d_c}$ denote the VN connected to $m$) or "compressed" (i.e., for a CN $m$, the corresponding $\beta$_message is given by the signs of the above $\beta_{m,n_i}$ messages, their first and second minimum, denoted by min1 and min2, and the index of the first minimum, denoted by indx_min1) [22].

*3) Read and Write Permutations (PER_R and 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.

*4) Barrel Shifters (BS_INIT and BS_R):* Barrel shifters are used to implement the cyclic (shift) permutations, according to the nonnegative entries of the base matrix. The $\tilde{\gamma}$_memory is initialized from the input LLR values stored in the input buffer. However, input LLR values are shifted by BS_INIT block before being written to the $\tilde{\gamma}$_memory, according to the *last* nonnegative shift factor on the corresponding base matrix column. BS_R blocks are then used to shift the LLR values read from the $\tilde{\gamma}$_memory, such that to properly align them with the appropriate VNU. Note that there are $d_c$ BS_R_$\{1, \ldots, d_c\}$ blocks. In case $\text{RPL} = 1$, a decoding layer corresponds to a row of $B$, and each BS_R block is used to shift the LLR values within one of the $d_c$ columns with nonnegative entries in the current row. Let $i$ be the index of the current row. The

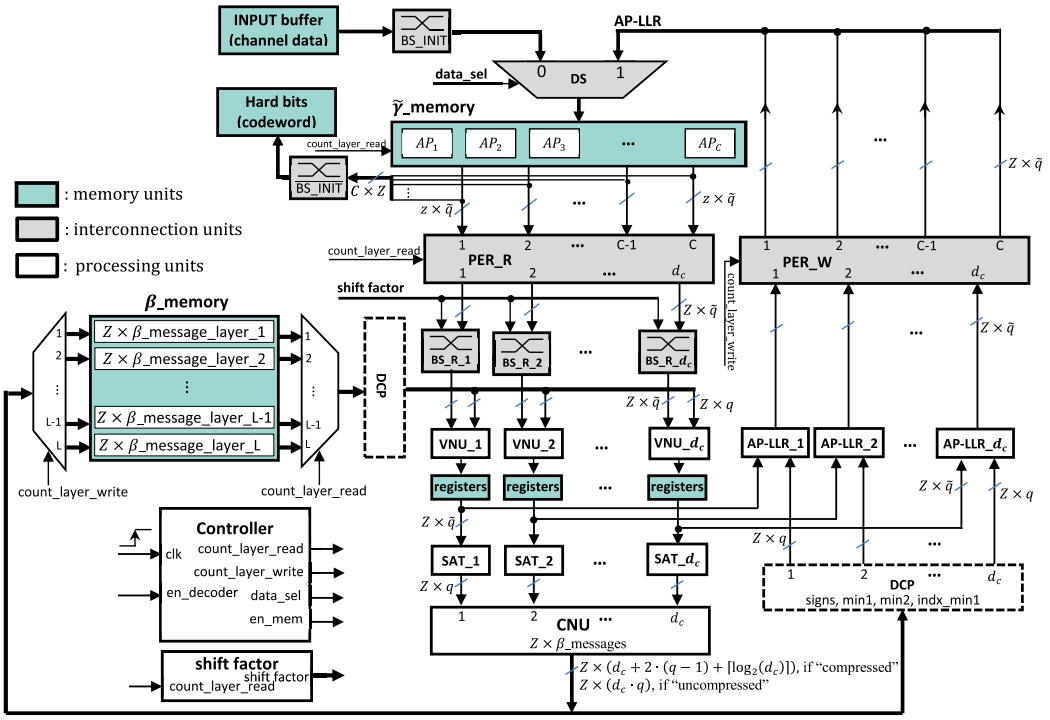



Fig. 2. Block diagram of the proposed pipelined architecture with MS-kernel.

cyclic shift implemented by a BS_R block, corresponding to a column $j$ with $b_{i,j} \geq 0$, is given by $-b_{i',j} + b_{i,j}$, where $b_{i',j}$ is the previous nonnegative entry in column $j$ (i.e., the previous row $i'$ with $b_{i',j} \geq 0$). In case RPL $> 1$, each BS_R block actually consists of RPL subblocks as above, with one subblock for each row in the layer. The values of the cyclic shifts are computed off-line for each layer $\ell$. This eliminates the need for data write-back barrel shifters, thus reducing the critical path of the design. Finally, the $\overline{\text{BS\_INIT}}$ block operates oppositely to BS_INIT, and is used to shift back the hard decision bits into appropriate positions.

*5) Variable Node Units and AP-LLR Units:* These units compute VN-messages ($\alpha_{m,n}$) and AP-LLR values ($\tilde{\gamma}_n$). Each VN message is computed by subtracting the corresponding CN message from the AP-LLR value, that is $\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}$. This operation is implemented by a $\tilde{q}$-bit subtractor, and hence the $\alpha_{m,n}$ value outputted by the VNU is quantized on $\tilde{q}$ bits. The AP-LLR value is updated by the AP-LLR unit, by $\tilde{\gamma}_n = \alpha_{m,n} + \beta_{m,n}^{\text{new}}$, where $\beta_{m,n}^{\text{new}}$ is the corresponding CN message computed at the current iteration (see in the following).

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

*7) Check Node Units:* These processing units compute the CN-messages ($\beta_{m,n}$). For simplicity, Fig. 2 shows one CNU block with $d_c$ inputs, each one of size $Z \times q$ bits. Thus, this block actually includes $Z$ computing units used to process in parallel the $Z$ CNs within one layer. The CNU is implemented by using either: 1) the high-speed low-cost tree-structure approach proposed in [23] for "compressed" CN-messages or 2) comparator trees for "uncompressed" CN-messages.

*8) Decompress (DCP):* This block is only used in case that the CN messages are in compressed format (signs, min1, min2, and indx_min1). It converts the $\beta$_messages from compressed to the uncompressed format.

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| 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 |
| -1 | 24 | -1 | -1 | -1 | -1 | -1 | 51 | -1 | 38 | -1 | -1 | -1 | -1 | 6 | -1 | -1 | -1 | -1 | 23 | -1 | 16 | -1 | -1 |

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

*9) Controller:* This block generates control signals, such as *count_layer_read* and *count_layer_write* to indicate which layers are being processed, *write_en* to enable data writing, and so on. It also controls the synchronous execution of the other blocks.

*10) Pipelining:* To increase the operating frequency, the data path is pipelined by adding a set of registers after the VNU-blocks. The timing schedule is shown in Fig. 5(a), where the two pipeline stages (P1 and P2) are indicated by purple and brown arrows. Hence, processing one layer takes two clock cycles, but at each clock cycle the two pipeline stages work on two consecutive layers of the base matrix. This imposes specific constraints on the base matrix, as consecutive layers must not overlap, in order to avoid $\tilde{\gamma}$_memory conflicts (note that memory stall cycles would cancel the pipelining effect). An example of $d_c = 6$ regular base matrix without overlap between consecutive layers is given in Fig. 3, assuming that each layer corresponds to one row of the base matrix.

*11) Regular NS-FAID Decoding Kernel:* The changes required to integrate a regular NS-FAID decoding kernel, with framing function $F$, are shown in Fig. 5(a). First, the

Saturation (SAT) block used within the MS-decoding kernel is replaced by a Framing (FRA) block. Note that the output of the VNU consists of $\tilde{q}$-bit (unsaturated) VN-messages. Hence, the FRA block actually implements the concatenation of the following operations, corresponding to $F \circ s_{\mathcal{M}}$ in (4):

$$[-\tilde{Q}, \ldots, +\tilde{Q}] \xrightarrow{s_{\mathcal{M}}} [-Q, \ldots, +Q] \xrightarrow{F} \mathrm{Im}(F) \xrightarrow{\sim} [-W, \ldots, +W] \quad (13)$$

where $[-\tilde{Q}, \ldots, +\tilde{Q}]$ is the alphabet of unsaturated messages ($\tilde{Q} = 2^{\tilde{q}-1} - 1$), $F$ is the framing function being used, $\mathrm{Im}(F)$ is the image of $F$ (which is a subset of $[-Q, \ldots, +Q]$ according to the framing function definition), and the last operation consists of a requantization of the $\mathrm{Im}(F)$ values on a number of $w$-bits, where $w = \lceil \log_2(W) \rceil + 1$ is the framing bit-length. The Deframing (DE-FRA) block simply converts back from $w$-bit to $q$-bit values ($[-W, \ldots, +W] \xrightarrow{\sim} \mathrm{Im}(F) \subset [-Q, \ldots, +Q]$), i.e., it inverts the requantization operation above. Although we have to add the deframing blocks, the reduction of the CN-messages size may still save significant hardware resources, as compared with MS decoding. This will be discussed in more detail in Section V.

*12) Irregular NS-FAID Decoding Kernel:* First, we note that the pipeline architecture proposed in this section can be applied to the WiMAX QC-LDPC code with rate 1/2, considered in Section III-B, by assuming that each decoding layer consists of one row of the base matrix. Indeed, it is known that for this code, the rows of the base matrix can be reordered, such that any two consecutive rows do not overlap [24].

Regarding the integration of an irregular NS-FAID decoding kernel, the same framing (FRA) or deframing (DE-FRA) block is reused for several VNs, which may be of different degrees. This may require several framing functions to be implemented within the FRA/DE-FRA blocks, thus increasing the hardware complexity. To overcome this problem, one may change the way the VNs are mapped to the processing units, by reordering the columns of the base matrix processed within each decoding layer. We determine off-line such a reordering for each decoding layer, so as to minimize the number of FRA/DE-FRA blocks implementing more than one single framing function. Hence, the PER_R and PER_W blocks, which ensure the proper alignment between data and processing units, are redefined accordingly.

The optimal mapping between VNs and VNUs is shown in Fig. 4, for the base matrix with reordered rows from [24]. For each VNU, we indicate the index of the VN (or equivalently, base matrix column) processed by the VNU, within each decoding layer. The last row of the table indicates the number of framing functions that have to be implemented within the FRA/DE-FRA blocks corresponding to each VNU. One may see that three of the FRA/DE-FRA blocks must implement two framing functions, while the other four FRA/DE-FRA blocks implement only one framing function.

| | VNU1 | VNU2 | VNU3 | VNU4 | VNU5 | VNU6 | VNU7 |
|---|---|---|---|---|---|---|---|
| Layer 1 | $v2$ | $v13$ | $v9$ | $v10$ | $v3$ | $v14$ | – |
| Layer 2 | $v4$ | $v6$ | $v5$ | $v8$ | $v12$ | $v15$ | $v16$ |
| Layer 3 | $v7$ | $v3$ | $v11$ | $v10$ | $v17$ | $v18$ | – |
| Layer 4 | $v1$ | $v6$ | $v13$ | $v12$ | $v8$ | $v24$ | – |
| Layer 5 | $v4$ | $v3$ | $v11$ | $v10$ | $v19$ | $v20$ | – |
| Layer 6 | $v1$ | $v6$ | $v5$ | $v8$ | $v12$ | $v21$ | $v22$ |
| Layer 7 | $v4$ | $v3$ | $v9$ | $v10$ | $v23$ | $v24$ | – |
| Layer 8 | $v2$ | $v6$ | $v7$ | $v8$ | $v12$ | $v14$ | $v15$ |
| Layer 9 | $v1$ | $v3$ | $v9$ | $v19$ | $v16$ | $v17$ | – |
| Layer 10 | $v5$ | $v6$ | $v13$ | $v12$ | $v8$ | $v18$ | $v19$ |
| Layer 11 | $v2$ | $v3$ | $v7$ | $v10$ | $v20$ | $v21$ | – |
| Layer 12 | $v6$ | $v8$ | $v11$ | $v12$ | $v22$ | $v23$ | – |
| No. FRAs | 2 | 2 | 1 | 1 | 2 | 1 | 1 |

Fig. 4. Mapping between VNs and VNUs. Black: VNs of degree 2. Red: VNs of degree 3. Blue: VNs of degree 6.

## B. Full-Layer Architecture

A different possibility to increase throughput is to increase the hardware parallelism, by including several nonoverlapping rows of the base matrix in one decoding layer. For instance, for the base matrix in Fig. 3, we may consider RPL $= 4$ consecutive rows per decoding layer, and thus the number of decoding layers is $L = 3$. In this case, each column of the base matrix has one (and only one) nonzero entry in each decoding layer; such a decoding layer is referred to as being full. Full layers correspond to the maximum hardware parallelism that can be exploited by layered architectures, but they also prevent the pipelining of the data path. One possibility to implement a full-layer decoder is to use a similar architecture to the pipelined one, by removing the registers inserted after the VNU (since pipelining is incompatible with the use of full layers), and updating the control unit. However, in such an architecture, read/write operations from/to the $\beta$_memory would occur at the same memory location, corresponding to the current layer being processed $\ell$. This would require the use of asynchronous dual-port RAM to implement the $\beta$_memory, which in general is known to be slower than synchronous dual-port RAM. The architecture proposed in this section, shown in Fig. 5(b), is aimed at avoiding the use of asynchronous RAM, while providing an effective way to benefit from the increased hardware parallelism enabled by the use of full layers. We discuss below the main changes with respect to the pipelined architecture from the Section IV-A, consisting of the $\alpha$_memory and the barrel shifters blocks (the other blocks are the same as for the pipelined architecture), as well as a complete reorganization of the data path. However, it can be easily verified that both architectures are logically equivalent, i.e., they both implement the same decoding algorithm.

*1) $\alpha$_Memory:* This memory is used to store the VN-messages for the current decoding layer (unlike the previous architecture, the AP-LLR values are not stored in memory). Since only one $\tilde{q}$-bit (unsaturated) VN-message is stored for each VN, this memory has exactly the same size as the $\tilde{\gamma}$_memory used within the previous pipelined architecture. VN-messages for the current layer $\ell$ are read from the $\alpha$_memory, then saturated or framed depending on the decoding kernel, and supplied to the corresponding CNUs. CN-messages computed by the CNUs are stored in the $\beta$_memory (location corresponding to layer $\ell$), and also forwarded to the AP-LLR unit, through the DCP (decompress) and DE-FRA (deframing) blocks, according to the CNU implementation (compressed or uncompressed) and the decoding

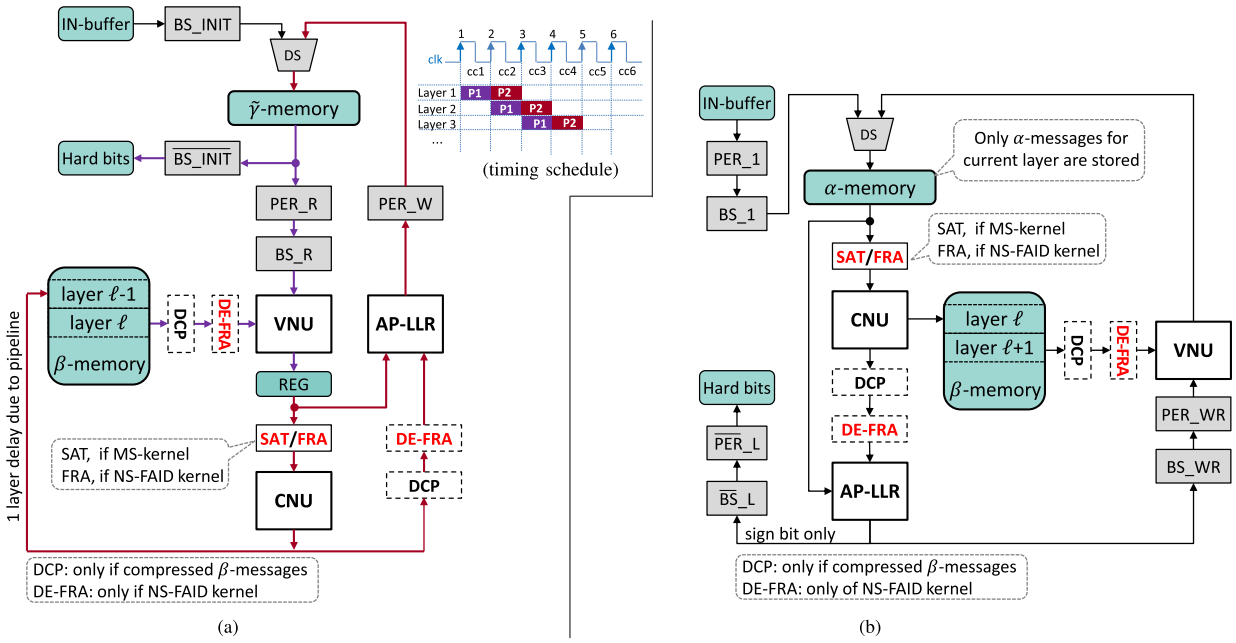



Fig. 5. High-level description of the proposed HW architectures, with both MS and NS-FAID kernels. (a) Pipelined architecture. (b) Full-layer architecture.

kernel (MS of NS-FAID). The AP-LLR unit computes the sum of the incoming VN- and CN-messages, which corresponds to the AP-LLR value to be used at layer $\ell+1$ (since already updated by layer $\ell$). The AP-LLR value is forwarded to the VNU, through corresponding BS and PER blocks. Eventually, the VN-message for the layer $\ell+1$ is computed as the difference between the incoming AP-LLR and the corresponding layer-$(\ell+1)$ CN-message computed at the previous iteration, the latter being read from the $\beta$_memory.

*2) PER/BS Blocks:* PER_1/BS_1 blocks permute/shift the data read from the input buffer, according to the positions/values of the nonnegative entries in the first decoding layer. Similar to the BS_R blocks in the pipelined architecture, the PER_WR/BS_WR blocks permute/shift the AP-LLR values, according to the difference between the positions/values of the current layer's $(\ell)$ nonnegative entries and those of the next layer $(\ell+1)$. This way, VN-messages stored in the $\alpha$_memory are already permuted and shifted for the subsequent decoding layer. Finally, PER_$L$/BS_$L$ blocks permute/shift the hard decision bits (sign of AP-LLR values), according to the positions/values of the nonnegative entries in the last decoding layer.

## V. Implementation Results

This section reports implementation results, as well as the error correction performance of the implemented codes, in order to corroborate the analytic results obtained in Section III. As mentioned in Section IV, both architectures are logically equivalent, and thus they both yield the same decoding performance (assuming that they implement the same MS/NS-FAID decoding kernel), but they may have a different performance in terms of area and throughput.

### A. Regular LDPC Codes

We consider the $(3,6)$-regular QC-LDPC code with base matrix $B$ of size $R \times C = 12 \times 24$, as shown in Fig. 3. The expansion factor $z = 54$, and thus the code-word length is $N = zC = 1296$ bits. The base matrix can be divided in either $L = 12$ decoding layers (RPL $= 1$), for the pipelined architecture, or $L = 3$ horizontal decoding layers (RPL $= 4$), for the full-layer architecture.

Fig. 6(a) shows the BER performance of the MS decoder with quantization parameters $(q, \tilde{q}) = (4,6)$, as well as $q = 4$-bit NS-FAIDs with $\omega = 2$ and $\omega = 3$ [framing functions $F$ corresponding to $\omega$ and $F(0)$ values in the legend are those from Table II]. Binary input AWGN channel model is considered, with 20 decoding iterations. It can be seen that the simulation results corroborate the analytic results from Section III-A, in terms of SNR gain/loss provided by NS-FAIDs, as compared with MS. For comparison purposes, we have further included simulations results for the floating-point belief propagation decoder [12], as well as the MS decoder with $(3,5)$ and $(2,4)$-quantization.

ASIC postsynthesis implementation results on 65-nm CMOS technology are shown in Table V, for the MS$(4,6)$ decoder and the NS-FAIDs with [$\omega = 3$ and $F(0) = 0$] and [$\omega = 2$ and $F(0) = \pm 1$], indicated in the table as NS-FAID-3 and NS-FAID-2, respectively. The first (Variant) row in Table V indicates the architecture (pipelined or full layers) and the CNU type (compressed or uncompressed). We also note that for the NS-FAID-2, the assumption that 0 is mapped to either $-1$ or $+1$, with equal probability, is only needed for theoretical analysis (the symmetry of the decoder allows reducing the analysis to the all-zero code word). However, in practical situations, one may always map 0 to $+1$, since random code words are transmitted (for instance, in telecommunications systems, pseudorandomness

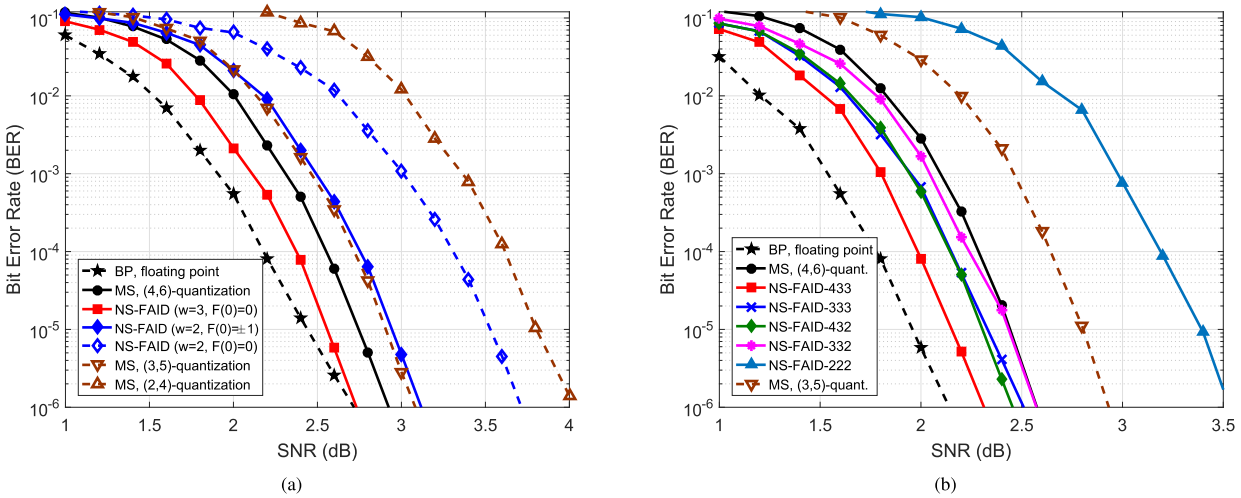



Fig. 6. BER performance of optimized regular and irregular NS-FAIDs. (a) (3, 6)-regular LDPC code. (b) WiMAX irregular LDPC code.

TABLE V
ASIC POSTSYNTHESIS IMPLEMENTATION RESULTS ON 65-nm CMOS TECHNOLOGY FOR (3, 6) REGULAR LDPC

| **Variant** | **pipelined.uncompressed** | | | **pipelined.compressed** | | | **full_layers.uncompressed** | | | **full_layers.compressed** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Decoder** | MS(4,6) | NS-FAID-3 | NS-FAID-2 | MS(4,6) | NS-FAID-3 | NS-FAID-2 | MS(4,6) | NS-FAID-3 | NS-FAID-2 | MS(4,6) | NS-FAID-3 | NS-FAID-2 |
| **Max. Freq. (MHz)** | 200 | 222 | 227 | 175 | 200 | 208 | 151 | 172 | 192 | 125 | 147 | 172 |
| **Throughput (Mbps)** | 1075 | 1193 | 1220 | 941 | 1075 | 1118 | 3261 | 3715 | 4147 | 2700 | 3175 | 3715 |
| **Area** ($mm^2$) | 0.45 | 0.42 | 0.38 | 0.41 | 0.38 | 0.36 | 0.80 | 0.72 | 0.68 | 0.75 | 0.67 | 0.65 |
| **TAR** ($Mbps/mm^2$) | 2389 | 2840 | 3210 | 2295 | 2828 | 3105 | 4076 | 5159 | 6098 | 3600 | 4738 | 5715 |
| **±% w.r.t. MS(4,6)** | 0 | +18.88 | +34.37 | 0 | +23.22 | +35.29 | 0 | +26.57 | +49.61 | 0 | +31.61 | +58.75 |

of the transmitted data is ensured by a scrambling mechanism).

Throughput reported in Table V is given by the formula

$$\text{Throughput} = \frac{N \times f_{\max}}{\delta + L \times n_{\text{iter}}} \tag{14}$$

where $f_{\max}$ is the maximum operating frequency (postsynthesis), $n_{\text{iter}}$ is the number of decoding iterations (set to 20), and $\delta = 1$ for the pipelined architecture or $\delta = 0$ for the full-layer architecture. To keep the throughput comparison on an equal basis, we further define the throughput-to-area ratio metric TAR = throughput/area ($Mb/s/mm^2$).

While the NS-FAID-3 decoder outperforms the baseline MS(4, 6) decoder by 0.19 dB at BER $= 10^{-5}$ [see Fig. 6(a)], it can be seen from Table V that it also exhibits a TAR improvement between 18.88% and 31.61%, depending on the hardware architecture and a CNU type. As predicted, the NS-FAID-2 decoder exhibits a performance loss of 0.21 dB compared with MS(4, 6), but yields a significant TAR improvement, by 34.37% to 58.75%.

### *B. Component-Level Area Analysis*

This section presents a component-level analysis of area savings and overhead of the NS-FAID kernel implementation with respect to the conventional MS decoder, for the full-layer architecture with uncompressed CN messages. Note that this architecture achieves the highest TAR value, for both NS-FAID and MS kernels (see Table V). Fig. 7 provides area results for the three blocks of the architecture that are impacted by the use of the NS-FAID kernel: the $\beta$-memory block, the SAT/FRA and DE-FRA blocks, and the CNUs (which operate in the lower quantization domain). All the other blocks are the same for both MS and NS-FAIDs. The total area is also shown on the right-hand side of Fig. 7, with corresponding area values shown on the right $y$-axis. The height of the vertical bars indicates the area values obtained when the decoders are synthesized to maximize their operating clock frequency (as reported in Table V). As expected, it can be observed that NS-FAIDs allow significant area reductions for the $\beta$-memory and CNU blocks. Indeed, for NS-FAID-2 and NS-FAID-3 decoders, the $\beta$-memory block occupies 0.075 and 0.111 $mm^2$, respectively, representing only 49.7% and 73.5% as compared to MS (0.151 $mm^2$). Gains even more significant are obtained for the CNU block, which occupies 0.033 and 0.045 $mm^2$, representing only 24.4% and 33.3% as compared to MS (0.135 $mm^2$). Since NS-FAIDs require an additional DE-FRA block, this leads to an increased area of the SAT/FRA & DE-FRA blocks. However, these blocks make only a modest contribution to the total area values. These results confirm the benefits of the NS-FAID approach, in terms of both area and maximum operating frequency.

Finally, we note that the occupied area may considerably increase when timing constraints are pushed to their limits [27] (i.e., when the design is synthesized at the maximum operating frequency). Indeed, the occupied area is known to decrease with loosening timing constraint, and it eventually

TABLE VI
ASIC POSTSYNTHESIS IMPLEMENTATION RESULTS ON 65-nm CMOS TECHNOLOGY FOR WiMAX LDPC

| Variant | pipelined.uncompressed | | | | | | pipelined.compressed | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decoder | MS(4,6) | NS-FAID 433 | NS-FAID 432 | NS-FAID 333 | NS-FAID 332 | NS-FAID 222 | MS(4,6) | NS-FAID 433 | NS-FAID 432 | NS-FAID 333 | NS-FAID 332 | NS-FAID 222 |
| **Max. Freq. (MHz)** | 175 | 172 | 178 | 192 | 192 | 200 | 161 | 156 | 161 | 178 | 178 | 200 |
| **Throughput (Mbps)** | 1673 | 1644 | 1701 | 1835 | 1835 | 1912 | 1539 | 1491 | 1539 | 1701 | 1701 | 1912 |
| **Area ($mm^2$)** | 0.87 | 0.88 | 0.90 | 0.82 | 0.80 | 0.70 | 0.77 | 0.79 | 0.79 | 0.75 | 0.75 | 0.72 |
| **TAR ($Mbps/mm^2$)** | 1922 | 1868 | 1890 | 2237 | 2293 | 2731 | 1998 | 1887 | 1948 | 2268 | 2268 | 2655 |
| **$\pm\%$ w.r.t. MS(4,6)** | 0 | -2.81 | -1.66 | +16.39 | +19.30 | +42.09 | 0 | -5.56 | -2.50 | +13.51 | +13.51 | +32.88 |

TABLE VII
COMPARISON BETWEEN THE PROPOSED NS-FAID AND THE STATE-OF-THE-ART IMPLEMENTATIONS FOR THE WiMAX QC-LDPC CODE

| Decoders | K. Zhang'09 [4] | B. Xiang'11 [6] | T. Heidari'13 [25] | W. Zhang'15 [24] | K. Kanchetla'16 [26] | This work NS-FAID-332 |
|---|---|---|---|---|---|---|
| **Code length** | 2304 | 576-2304 | 2304 | 576-2304$^{(\dagger)}$ | 576-2304$^{(\dagger)}$ | 2304 |
| **Technology (nm)** | 90 | 130 | 130 | 40 | 90 | 65 |
| **Frequency (MHz)** | 950 | 214 | 100 | 290 | 149 | 192 |
| **Iterations** | 10 | 10 | 10 | 10 | 5 | 20 |
| **Throughput (Mbps)** | 2200 | 955 | 183 | 2227 | 955 | 1835 |
| **Tput scaled to 65nm (Mbps)** | 3036 | 1910 | 366 | 1370 | 1318 | 1835 |
| **Area ($mm^2$)** | 2.90$^{(*)}$ | 3.03$^{(*)}$ | 6.90$^{(**)}$ | 2.26$^{(*)}$ | 11.42$^{(*)}$ | 0.80$^{(*)}$ |
| **Area scaled to 65nm ($mm^2$)** | 1.51$^{(*)}$ | 0.76$^{(*)}$ | 1.73$^{(**)}$ | 5.97$^{(*)}$ | 5.94$^{(*)}$ | 0.80$^{(*)}$ |
| **TAR ($Mbps/mm^2$)** | 2011 | 2513 | 212 | 229 | 222 | 2293 |
| **NTAR ($Mbps/mm^2/iter$)** | 20110 | 25130 | 2120 | 2290 | 1110 | 45860 |

$^{(\dagger)}$ support both WiMAX and Wi-Fi standards
$^{(*)}$ only core area is reported
$^{(**)}$ total chip area is reported
TAR = (Throughput scaled to 65nm) / (Area scaled to 65nm)
NTAR = TAR $\times$ Iterations

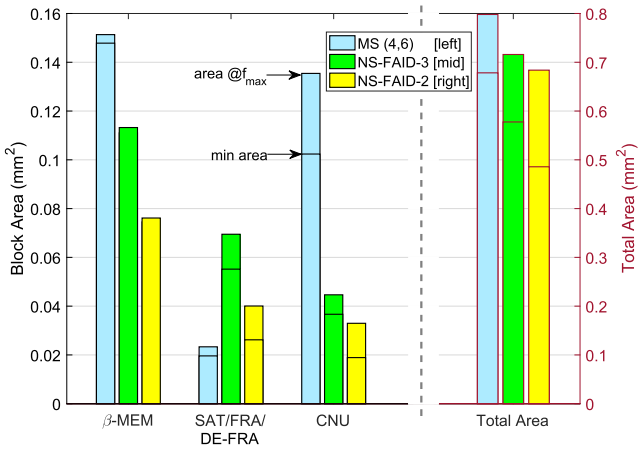



Fig. 7. Component-level area comparison (full-layer architecture, with uncompressed CN messages).

reaches a minimum value when the timing constraint is loose enough. For completeness, we have also depicted the case when the design is optimized with respect to the occupied area. Minimum area values are also shown in Fig. 7, by a horizontal line across each of the area bars (note that no decrease in the area of the $\beta$-memory block has been observed for the NS-FAID-2 and NS-FAID-3 decoders). It can be observed that the minimum total area values of NS-FAID-2 and NS-FAID-3 decoders are 0.486 and 0.578 $mm^2$, respectively, representing 71.7% and 85.3% of the minimum total area of the MS decoder (0.678 $mm^2$). These results show the same resource consumption trend as for the maximum frequency case.

### C. Irregular LDPC Codes

We consider the irregular WiMAX QC-LDPC code of rate 1/2, with base matrix of size $R \times C = 12 \times 24$ [21]. The expansion factor $z = 96$, thus resulting in a codeword length $N = zC = 2304$ bits. The pipelined architecture from Section IV-A is implemented, with RPL $= 1$ row per decoding layer, after reordering the rows of the base matrix, such that any two consecutive rows do not overlap [24]. Note that the full-layer architecture does not apply to irregular WiMAX LDPC codes, since it is not possible to group the rows of the base matrix in full decoding layers.

BER results for the MS$(4, 6)$ decoder and the NS-FAID-$w_1w_2w_3$ decoders from Table III are shown in Fig. 6(b), while ASIC postsynthesis implementation results on 65-nm CMOS technology are shown in Table VI. The throughput reported is computed using (14). TAR results and corresponding gain/loss $(+/-)$ with respect to the MS$(4, 6)$ decoder are reported on the last row. The NS-FAID-433 and NS-FAID-432 decoders outperform the MS decoder by 0.3 and 0.15 dB (at BER$=10^{-5}$), respectively, at the price of a small degradation of the TAR. NS-FAIDs-333 improves the BER performance by 0.12 dB, with TAR improvement by 13.51%–16.39%,

depending on the CNU type (compressed or uncompressed). NS-FAIDs-332 exhibits similar BER performance, with TAR improvement by 13.51%–19.30%. The NS-FAID-222 decoder yields the most significant TAR improvement (up to 42.09%), but this comes at the price of a significant BER degradation by $\approx 1$ dB as estimated in Section III-B.

To further emphasize the high-throughput characteristic of the proposed architecture, the irregular NS-FAID-332 decoder is further compared with other state-of-the-art implementations of WiMAX decoders in Table VII. We also report the TAR and normalized TAR (NTAR) metrics, so as to keep the throughput comparison on an equal basis with respect to technology, area, and a number of iterations. To scale throughput and area to 65 nm, we use scale factors (technology_size/65) and (65/technology_size)$^2$, respectively, as suggested in [28]. The computation of the TAR and NTAR metrics is detailed in the footnote of Table VII. Note that for all the reported implementations, the achieved throughput is inversely proportional to the number of iterations, and hence the NTAR metric corresponds to the TAR value assuming that only one decoding iteration is performed. We mention that the decoders proposed in [24] and [26] are reconfigurable decoders that support the IEEE 802.16e (WiMAX) and the IEEE 802.11n (WiFi) wireless standards. The reported throughput is the maximum achievable coded throughput for the $(1152, 2304)$ WiMAX code, with either ten or five decoding iterations. From Table VII, it can be seen that the proposed irregular NS-FAID compares favorably with the state-of-the-art implementations, yielding an NTAR value of 45.86 Gb/s/ mm$^2$/iteration.

## VI. Conclusion

In this paper, we first introduced the new framework of NS-FAIDs, which allows trading off decoding performance for hardware complexity reductions. NS-FAIDs have been optimized by DE and shown to provide significant memory size reductions, with similar or even better decoding performance, as compared to the MS decoder. Then, two hardware architectures have been presented, making use of either pipelining or increased hardware parallelism in order to increase throughput. Both MS and NS-FAID decoding kernels have been integrated into each of the two proposed architectures, and compared in terms of area and throughput. ASIC postsynthesis implementation results demonstrated the effectiveness of the NS-FAID approach in yielding significant improvements in terms of area and throughput, as compared to the MS decoder, with even better or only slightly degraded decoding performance.

## Acknowledgment

Preliminary version of part of this paper has been previously published in [17] and [18]. In this paper, the previous definition and density-evolution analysis of nonsurjective finite alphabet iterative decoders (NS-FAIDs) [17] is extended to framing functions with $F(0) = \pm\lambda$, such as to cover a larger class of decoders, which is shown to significantly improve the decoding performance in case that the exchanged messages are quantized on a small number of bits (e.g., 2 bits per exchanged message). Optimization results presented in Section III are new, and they report on the optimization of regular and irregular NS-FAIDs, by considering the proposed extension. The hardware architectures proposed in [18] have been extended to cover the case of irregular NS-FAIDs. In addition, implementation results reported in this paper target an ASIC technology, which is more likely to reflect the benefits of the proposed NS-FAID approach in terms of throughput/area tradeoff. All the implementation results reported in Section V (for both regular and irregular codes) are new.

## References

[1] M. Karkooti and J. R. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," in *Proc. Int. Conf. Inf. Technol., Coding Comput. (ITCC)*, vol. 1. Apr. 2004, pp. 579–585.

[2] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 1, pp. 98–111, Jan. 2011.

[3] V. A. Chandrasetty and S. M. Aziz, "Resource efficient LDPC decoders for multimedia communication," *Integr., VLSI J.*, vol. 48, pp. 213–220, Jan. 2015.

[4] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, Aug. 2009.

[5] X. Peng, Z. Chen, X. Zhao, D. Zhou, and S. Goto, "A 115 mW 1 Gbps QC-LDPC decoder ASIC for WiMAX in 65 nm CMOS," in *Proc. IEEE Asian Solid State Circuits Conf. (A-SSCC)*, Nov. 2011, pp. 317–320.

[6] B. Xiang, D. Bao, S. Huang, and X. Zeng, "An 847–955 Mb/s 342–397 mW dual-path fully-overlapped QC-LDPC decoder for WiMAX system in 0.13 $\mu$m CMOS," *IEEE J. Solid-State Circuits*, vol. 46, no. 6, pp. 1416–1432, Jun. 2011.

[7] E. Boutillon and G. Masera, "Hardware design and realization for iteratively decodable codes," in *Channel Coding: Theory, Algorithms, and Applications*. Amsterdam, The Netherlands: Elsevier, 2014, pp. 583–642.

[8] S. K. Planjery, S. K. Chilappagari, B. Vasić, D. Declercq, and L. Danjean, "Iterative decoding beyond belief propagation," in *Proc. IEEE Inf. Theory Appl. Workshop (ITA)*, Jan. 2010, pp. 1–10.

[9] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasić, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *Electron. Lett.*, vol. 47, no. 16, pp. 919–921, 2011.

[10] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasić, "Finite alphabet iterative decoders—Part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

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

[12] V. Savin, "LDPC decoders," in *Channel Coding: Theory, Algorithms, and Applications*. Amsterdam, The Netherlands: Elsevier, 2014, pp. 211–260.

[13] T. T. Nguyen-Ly *et al.*, "FPGA design of high throughput LDPC decoder based on imprecise offset min-sum decoding," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2015, pp. 1–4.

[14] D. Oh and K. K. Parhi, "Min-sum decoder architectures with reduced word length for LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 1, pp. 105–115, Jan. 2010.

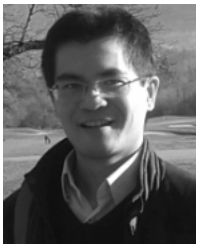
[15] V. A. Chandrasetty and S. M. Aziz, "An area efficient LDPC decoder using a reduced complexity min-sum algorithm," *Integr., VLSI J.*, vol. 45, no. 2, pp. 141–148, 2012.

[16] S. Abu-Surra, E. Pisek, T. Henige, and S. Rajagopal, "Low-power dual quantization-domain decoding for LDPC codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 3151–3156.

[17] T. T. Nguyen-Ly, K. Le, V. Savin, D. Declercq, F. Ghaffari, and O. Boncalo, "Non-surjective finite alphabet iterative decoders," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[18] T. T. Nguyen-Ly, V. Savin, X. Popon, and D. Declercq, "High throughput FPGA implementation for regular non-surjective finite alphabet iterative decoders," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[19] Z. Mheich, T.-T. Nguyen-Ly, V. Savin, and D. Declercq, "Code-aware quantizer design for finite-precision min-sum decoders," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, Varna, Bulgaria, Jun. 2016, pp. 1–5.
[20] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.
[21] *Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, Amendment to Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Standard IEEE-802.16e, 2005.
[22] Z. Wang and Z. Cui, "A memory efficient partially parallel decoder architecture for quasi-cyclic LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 4, pp. 483–488, Apr. 2007.
[23] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin, "Algorithms of finding the first two minimum values and their hardware implementation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 11, pp. 3430–3437, Dec. 2008.
[24] W. Zhang, S. Chen, X. Bai, and D. Zhou, "A full layer parallel QC-LDPC decoder for WiMAX and Wi-Fi," in *Proc. IEEE 11th Int. Conf. ASIC (ASICON)*, Nov. 2015, pp. 1–4.
[25] T. Heidari and A. Jannesari, "Design of high-throughput QC-LDPC decoder for WiMAX standard," in *Proc. 21st Iranian Conf. Electr. Eng. (ICEE)*, May 2013, pp. 1–4.
[26] 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 Syst.*, vol. 10, no. 2, pp. 111–120, 2016.
[27] H. Bhatnagar, *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler Physical Compiler and PrimeTime*. New York, NY, USA: Springer-Verlag, 2007.
[28] J. R. Hauser, "MOSFET device scaling," in *Handbook of Semiconductor Manufacturing Technology*. Boca Raton, FL, USA: CRC Press, 2008.



**Thien Truong Nguyen-Ly** received the B.S. and M.S. degrees in electronic and telecommunication engineering from the Ho Chi Minh City University of Technology (HCMUT), Ho Chi Minh City, Vietnam, in 2010 and 2012, respectively. He is currently working toward the Ph.D. degree in telecommunications engineering with the Broadband Wireless Systems Laboratory, CEA-LETI, MINATEC Campus, Grenoble, France, and ETIS ENSEA/UCP/CNRS UMR-8051, Cergy-Pontoise, France.

From 2010 to 2014, he was a Lecturer with the Faculty of Electrical and Electronic Engineering, HCMUT. His current research interests include error-correction coding, the analysis and implementation of low-density parity-check decoder architectures on FPGA/ASIC platform, and speech processing.

**Valentin Savin** received the master's degree in mathematics from the Ecole Normale Suprieure of Lyon, Lyon, France, in 1997, the Ph.D. degree in mathematics from the J. Fourier Institute, Grenoble, France, in 2001, and the M.S. degree in cryptography, security, and coding theory from the University of Grenoble 1, Grenoble.

Since 2005, he has been with the Digital Communications Laboratory, CEA-LETI, Grenoble, France, first as a two-year Postdoctoral Fellow, and then as a Research Engineer. Since 2016, he has been appointed as a CEA Senior Expert on information and coding theory. Over the last years, he has been involved in the design of low-complexity decoding algorithms for low-density parity-check (LDPC) and polar codes, and in the analysis and the optimization of LDPC codes for physical and upper layers applications. He has published over 80 papers in international journals and conference proceedings, and he holds ten patents. He is currently participating in or coordinating several French and European research projects in information and communications technology.

**Khoa Le** received the B.S. and M.S. degrees in electronics and telecommunication engineering from the Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam, in 2010 and 2012, respectively. He is currently working toward the Ph.D. degree with the ETIS Laboratory, École Nationale Supérieure de l'électronique et de ses Applications, University of Cergy-Pontoise, CNRS UMR-8051, Cergy-Pontoise, France.

His current research interests include error correcting code algorithms, analysis, and their implementations in FPGA/ASIC.

**David Declercq** (SM'11) was born in 1971. He received the Ph.D. degree in statistical signal processing from the University of Cergy-Pontoise, Cergy-Pontoise, France.

He is currently a Full Professor with the École Nationale Supérieure de l'électronique et de ses Applications, Cergy-Pontoise. He has held a junior position at the Institut Universitaire de France from 2009 to 2014. His current research interests include digital communications and error-correction coding theory. For several years, he was involved in the particular family of low-density parity-check (LDPC) codes, both from the code and decoder design aspects. Since 2003, he has been developing a strong expertise on nonbinary LDPC codes and decoders in high-order Galois fields GF($q$). A large part of his research projects are related to nonbinary LDPC codes. He mainly investigated two aspects: the design of GF($q$) LDPC codes for short and moderate lengths and the simplification of the iterative decoders for GF($q$) LDPC codes with complexity/performance tradeoff constraints. He has published over 40 papers in major journals, including the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON INFORMATION THEORY, the IEEE COMMUNICATIONS LETTERS, and *EURASIP Journal on Wireless Communications and Networking*, and over 120 papers in major conferences in information theory and signal processing.

Dr. Declercq is the General Secretary of the National GRETSI Association.

**Fakhreddine Ghaffari** received the B.Sc. degree in electrical engineering and the M.S. degree from the National School of Electrical Engineering, Safax, Tunisia, in 2001 and 2002, respectively, and the Ph.D. degree in electronics and electrical engineering from the University of Nice Sophia Antipolis, Nice, France, in 2006.

He is currently an Associate Professor with ETIS ENSEA, University of Cergy Pontoise, Cergy Pontoise, France. His current research interests include VLSI design and implementation of reliable digital architectures for wireless communication applications in ASIC/FPGA platforms and the study of mitigating transient faults from algorithmic and implementation perspectives for high-throughput applications.

**Oana Boncalo** received the B.Sc. and Ph.D. degrees in computer engineering from University Politehnica Timisoara, Timisoara, Romania, in 2006 and 2009, respectively.

She is currently an Associate Professor with University Politehnica Timisoara. She has published over 50 research papers in topics related to digital design. Her current research interests include computer arithmetic, low-density parity-check decoder architectures, digital design, and reliability estimation and evaluation.