A survey of FPGA-based LDPC decoders

Peter Hailes, Lei Xu, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo
School of ECS, University of Southampton, SO17 1BJ, UK
Corresponding author: lh@ecs.soton.ac.uk

***Abstract*—Low-Density Parity Check (LDPC) error correction decoders have become popular in communications systems, as a benefit of their strong error correction performance and their suitability to parallel hardware implementation. A great deal of research effort has been invested into LDPC decoder designs that exploit the flexibility, the high processing speed and the parallelism of Field-Programmable Gate Array (FPGA) devices. FPGAs are ideal for design prototyping and for the manufacturing of small-production-run devices, where their in-system programmability makes them far more cost-effective than Application-Specific Integrated Circuits (ASICs). However, the FPGA-based LDPC decoder designs published in the open literature vary greatly in terms of design choices and performance criteria, making them a challenge to compare. This paper explores the key factors involved in FPGA-based LDPC decoder design and presents an extensive review of the current literature. In-depth comparisons are drawn amongst 140 published designs (both academic and industrial) and the associated performance trade-offs are characterised, discussed and illustrated. Seven key performance characteristics are described, namely their processing throughput, processing latency, hardware resource requirements, error correction capability, processing energy efficiency, bandwidth efficiency and flexibility. We offer recommendations that will facilitate fairer comparisons of future designs, as well as opportunities for improving the design of FPGA-based LDPC decoders.**

***Index Terms*—Digital communication, error correction codes, low-density parity check (LDPC) codes, field programmable gate array, iterative decoding**

I. Introduction

LOW-Density Parity Check (LDPC) codes may be employed for correcting transmission errors in communication systems. They represent a class of Forward Error Correction (FEC) codes that are currently the focus of much research within the communications community. They were first proposed by Gallager in 1962 [1], but they were considered to be too complex for practical simulation and implementation at the time of their conception, hence they were left largely untouched for decades. Apart from their excellent performance, perhaps partially motivated by the fact that the turbo codes patented during the early 1990s attracted a license-fee, in 1996 LDPC codes were rediscovered by Mackay and Neal [2], and ever since have enjoyed a renaissance. Given the increased computing power available today they have become a key component of many commercialised communication systems, including WiFi [3], WiMAX [4], DVB-S2 [5], CCSDS [6] and ITU G.hn [7].

LDPC codes benefit from a number of appealing features that make them very attractive for implementation. The LDPC decoding algorithm can be implemented using low-complexity calculations, resulting in a relatively low design and implementation cost for the processing hardware. Like turbo codes, LDPC codes are decoded iteratively, achieving an error correction performance that is close to the theoretical limit when decoding messages that have large block lengths [8]. However, in contrast to turbo codes, there is a wide variety of possible algorithms and levels of parallelisation that may be considered for the design of LDPC decoders, presenting designers with a range of options that may be relied upon to achieve the desired characteristics.

However, while the design of the individual processing components is relatively simple, the design of a complete LDPC decoder is subject to a complex interplay between a number of system characteristics, namely the processing throughput, processing latency, hardware resource requirements, error correction capability, processing energy efficiency, bandwidth efficiency and flexibility. These characteristics depend on a number of system parameters, namely the architecture, the LDPC code employed, the algorithm used and the number of decoding iterations. This relationship is shown in Fig. 1. Note that the bandwidth efficiency also depends on the modulation scheme chosen, as does the transmission energy efficiency, which furthermore depends on the coding gain and the error correction capability of the chosen LDPC code. To elaborate a little further in the context of Fig. 1, we can improve the error correction capability in many different ways, for example by using a stronger LDPC code or more decoding iterations. Naturally, increasing the number of iterations increases the complexity and hence reduces the processing energy efficiency, but increases the transmit energy efficiency. Hence the total energy dissipation should be considered holistically, when designing an LDPC decoder. Further similar trade-offs will emerge throughout our forthcoming discussions.

In order to fully characterise an LDPC decoder design, it is necessary to physically implement it. Perhaps the simplest way of doing so is to use a Field-Programmable Gate Array (FPGA) device, which facilitates rapid prototyping and fast parallel logic processing. This approach is especially useful for measuring the Bit Error Rate (BER) performance, since simulations that would take days on a computer can be completed in only hours when using a custom FPGA implementation [9]. These advantages are evident from the sheer number of published FPGA-based LDPC decoder designs that exist in the open literature, which will be compared

The financial support of the PhD studentship provided by Altera, California USA, the grants EP/J015520/1 and EP/L010550/1 provided by EPSRC, Swindon UK, the grant TS/L009390/1 provided by Innovate UK, Swindon UK, as well as the Advanced Fellow grant provided by the European Research Council is gratefully acknowledged. The research data for this paper is available at http://dx.doi.org/10.5258/SOTON/384946.

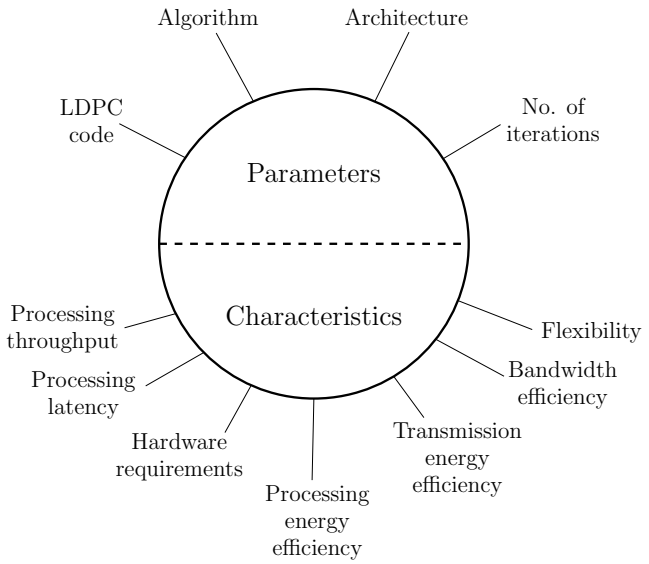


Fig. 1. FPGA-based LDPC decoder system parameters and characteristics

later in this paper. Furthermore, the decoding techniques and implementation-oriented research presented alongside these designs has been of significant benefit to the wider communications research community [10]–[15]. In particular, the implementational characteristics of these FPGA-based LDPC decoders are increasingly informing the holistic design of communication systems.

In addition to their suitability for prototyping, FPGAs constitute a viable alternative to Application-Specific Integrated Circuits (ASICs) for the LDPC decoders of small-production-run communication devices, while their programmability has made them attractive for software-defined radios. This paper focuses exclusively on FPGA implementations of LDPC decoders, since they cannot be fairly compared to ASIC implementations, which are designed at a significantly higher development cost to have particularly high performance for high-production-run applications. Indeed, ASIC implementations are even difficult to compare with each other, because some papers provide post-synthesis results, while others offer post-layout results. Meanwhile, some papers consider only the ASIC core, while others include both the memory and Input/Output (I/O) resources.

This paper has been conceived for achieving the following aims:

- Provide a tutorial on LDPC decoding, discussing both the parameters and characteristics that affect the performance of FPGA implementations.
- Accurately compare all implementations of FPGA-based LDPC decoders that we are aware of.
- Characterise the observed trade-offs and relationships between the system parameters and characteristics.
- Recommend good practice to aid future designs of FPGA-based LDPC decoders, and to make published designs more comparable with each other.
- Identify opportunities for the further enhancement of FPGA-based LDPC decoders.

The structure of the paper is as follows. Section II presents a brief tutorial on the LDPC code structure and encoding, as well as describing variations on the decoding algorithms, decoder architectures and FPGA devices. Section III provides our comparison of all FPGA-based LDPC decoders that we are aware of, whilst discussing the parameters and characteristics of an LDPC decoder in more detail. Section IV illustrates and characterises the observed trade-offs and relationships between the various parameters and characteristics of FPGA-based LDPC decoders. Recommendations for readers interested in developing their own FPGA-based LDPC decoders are offered in Section V, along with suggestions for further work in the area. Finally, we offer our conclusions in Section VI. This structure is depicted in Fig. 2.

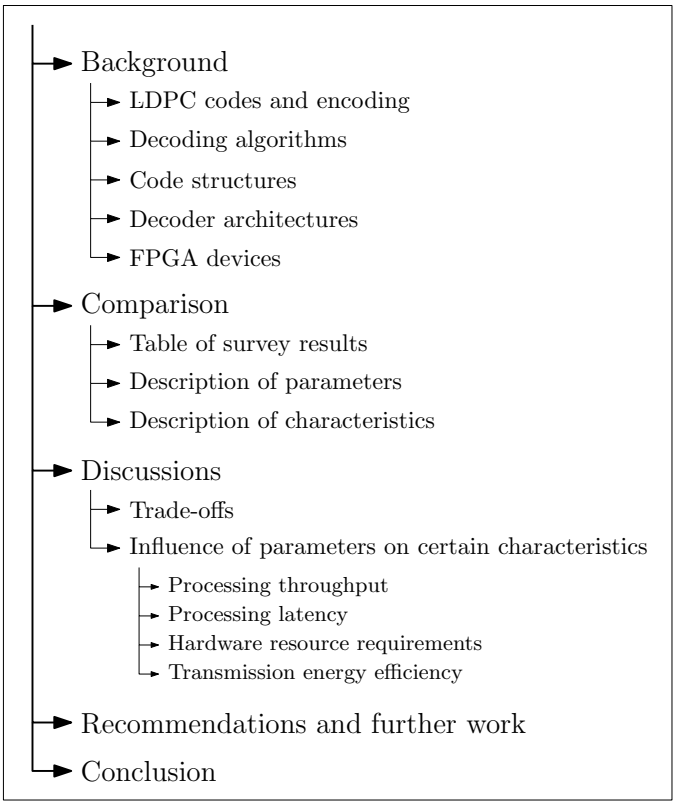


Fig. 2. Structure of this paper

II. Background

This section presents a tutorial on FPGA-based LDPC decoders. Section II-A commences by discussing FEC, before LDPC codes are introduced in Section II-B. This is followed by a discussion of how LDPC codes are decoded and designed in Sections II-C and II-D, respectively. The practicalities of LDPC decoder implementations are then discussed in Section II-E, which is followed by a brief introduction to FPGAs in Section II-F.

A. Forward error correction

Fig. 3 shows a schematic of a simplified communications system, where the information message word $\mathbf{m} = \{m_i\}_{i=1}^{K}$ is a vector of K bits, which is FEC encoded in order to obtain

the codeword $\mathbf{c} = \{c_j\}_{j=1}^{N}$, which is a vector of $N > K$ bits. The FEC encoder converts the K-bit message word $\mathbf{m}$ into the N-bit codeword $\mathbf{c}$ by adding $M = N - K$ parity bits to the message. The ratio of the message length K to the total codeword length N is referred to as the coding rate R,

$$R = \frac{K}{N} = \frac{N - M}{N}. \tag{1}$$

The M additional parity bits are derived from the K message bits and hence they do not carry any information of their own. However, they are used during the FEC decoding process to allow transmission errors to be detected and even corrected, depending on the specific scheme used and on the severity of the corruption, as will be discussed below.

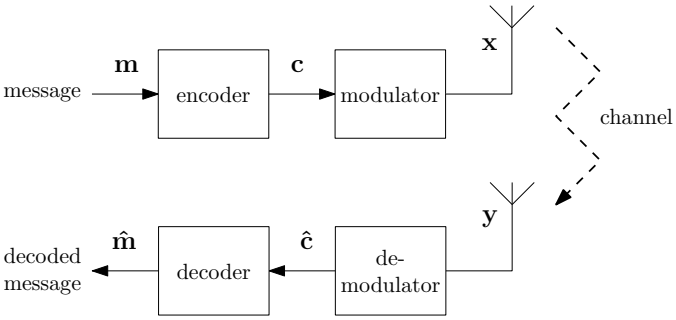


Fig. 3. A communications system

Various modulation schemes can be used for modulating the codeword $\mathbf{c}$ onto the channel. As we shall show in Section IV-B4, Binary Phase-Shift Keying (BPSK) modulation is assumed for nearly all FPGA-based LDPC decoder research. For this reason, we also assume the employment of BPSK modulation throughout this tutorial discussion. It is important to note however, that BPSK is a very simple modulation scheme, which is rarely employed alone in practical communication schemes. Therefore, considering BPSK modulation exclusively during the design phase could result in an LDPC decoder which does not necessarily work satisfactorily in practical systems, where higher-order modulation schemes are employed. Note that our analysis in Section IV will take the specific modulation scheme that was used into consideration, when comparing the error correction performance of various FPGA-based LDPC decoders.

BPSK generates the modulated symbol vector $\mathbf{x} = \{x_j\}_{j=1}^{N}$ according to $x_j = +\sqrt{E_s}$ when $c_j = 0$ and $x_j = -\sqrt{E_s}$ when $c_j = 1$, where E_s is the transmission energy per symbol. Similarly, there are several different ways of modelling the random corruption that is imposed by the channel upon the signal $\mathbf{x}$ as it is transformed into the received signal $\mathbf{y} = \{y_j\}_{j=1}^{N}$. In common with most FPGA-based LDPC research, we assume the Additive White Gaussian Noise (AWGN) channel model, in which a random noise signal is added to the transmitted signal,

$$y_j = x_j + \mathcal{CN}(0, N_0), \tag{2}$$

where $\mathcal{CN}(\cdot)$ is the complex normal distribution and N_0 is the noise power spectral density. The Signal to Noise Ratio (SNR) is given by E_s/N_0, and may also be expressed as the SNR per bit according to

$$\frac{E_b}{N_0} = \frac{1}{R} \times \frac{E_s}{N_0}. \tag{3}$$

The corruption imposed by the channel causes $\mathbf{y}$ to differ from $\mathbf{x}$ in an unpredictable manner, potentially resulting in the demodulation of a perturbed received codeword $\hat{\mathbf{c}}$, potentially including some transmission errors. The decoder of Figure 3 is employed to recover the message word $\hat{\mathbf{m}}$, and without this there would be no way of correcting (or even detecting the presence of) these errors.

The error correction capability of a FEC decoder is affected by the form of the information provided by the demodulator. Rather than using hard decisions to convert received symbols into demodulated bits, superior error correction capability can be obtained if the demodulator provides soft decisions, which are commonly expressed using the *Logarithmic-Likelihood Ratio* (LLR) [16]. The sign of an LLR (positive or negative) expresses *what* the most likely value for the corresponding bit is (0 or 1, respectively). Meanwhile, the magnitude of an LLR expresses *how* likely this value is, where 0 represents complete uncertainty and ∞ represents absolute certainty. The value of an LLR is calculated as

$$\tilde{c}_i = \log \frac{P(c_i = 0 \mid y_i)}{P(c_i = 1 \mid y_i)}, \tag{4}$$

where $\tilde{c}_i$ is the output LLR, c_i is the transmitted bit and y_i is the received symbol.

Here, the logarithm is used because it reduces the dynamic range of the likelihood ratio, tending to produce values in the range of -10 to $+10$, rather than 0.0001 to $10,000$. This also allows probability intersections to be calculated using additions, rather than hardware-intensive multiplication operations. LLRs are extensively used throughout the LDPC decoding process, as will be detailed below.

When using BPSK modulation over an AWGN channel, the demodulator can convert the received signals into LLRs according to

$$\tilde{c}_i = 4 \times R \times \frac{E_b}{N_0} \times \mathrm{Re}(y_i). \tag{5}$$

B. LDPC codes

This section provides an introduction to LDPC codes, commencing with their structure and the encoding process in Section II-B1. Following this, the decoder's Parity-Check Matrix (PCM) is introduced in Section II-B2 together with its graphical representation using factor graphs in Section II-B3.

1) Encoding: Decoding an LDPC codeword is associated with a significantly higher complexity than the encoding process, because the decoder must consider every possible message word simultaneously, while operating on the basis of soft decision LLRs rather than hard decision bits. For this reason, we focus our attention on LDPC decoders in this paper, but the encoding process is explained briefly here for the sake of completeness.

As described previously, LDPC codes permit the correction of transmission errors by supplementing each K-bit message word with M parity bits in order to produce an N-bit

codeword, where $N = K + M$ [17]. Codes which include the K bits of the message word within the N bits of the codeword are referred to as systematic, while non-systematic codes have codewords which do not directly contain the original message bits. There are 2^K possible permutations of the K-bit message word, each of which is mapped by the LDPC encoder to a corresponding one of 2^K legitimate codeword permutations. The error correction capability of the LDPC code depends on the minimum Hamming distance between any pair of these 2^K legitimate codeword permutations. Naturally high minimum distances are preferred, since these make it unlikely for a legitimate codeword to be transformed into another by the distortion introduced during transmission.

For example, a code with a message word length of $K = 6$ and a codeword length of $N = 10$ employs $M = N - K = 4$ parity bits and has a coding rate of $R = {}^{K}/_{N} = {}^{3}/_{5}$. In the case where the code is systematic, each codeword $\mathbf{c}$ may be of the form

$$\mathbf{c} = [c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}], \tag{6}$$

where $c_1 \dots c_6$ are the $K = 6$ bits of the message word $\mathbf{m}$ and $c_7 \dots c_{10}$ are the $M = 4$ parity bits. Each of the parity bits represents a parity check covering a specific subset of the message bits. As an example, the parity check bits may be obtained according to the following modulo-2 summations of message bits:

$$c_7 = c_4 \oplus c_6 \tag{7a}$$
$$c_8 = c_1 \oplus c_3 \oplus c_5 \oplus c_6 \tag{7b}$$
$$c_9 = c_2 \oplus c_5 \tag{7c}$$
$$c_{10} = c_1 \oplus c_2 \oplus c_6. \tag{7d}$$

The design of an LDPC code's parity check equations is subject to many complex factors, as will be briefly described in Section II-D. Using these equations, a $(K \times N)$-element generator matrix $\mathbf{G}$ can be constructed to efficiently describe the encoding process. In a systematic code, $\mathbf{G}$ may adopt the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_K & \mathbf{A} \end{bmatrix}, \tag{8}$$

where $\mathbf{I}_K$ is the $(K \times K)$-element identity matrix and the columns of $\mathbf{A}$ represent each of the parity checks. The generator matrix of the systematic code described above would therefore be

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{9}$$

Codewords can be calculated using this matrix by finding the modulo-2 matrix product of the message $\mathbf{m}$ and the generator matrix $\mathbf{G}$, according to $\mathbf{c} = \mathbf{m} \times \mathbf{G}$. For example, it may be readily verified that the message $\mathbf{m} = [0\ 1\ 1\ 1\ 0\ 1]$ has the corresponding codeword $\mathbf{c} = \mathbf{m} \times \mathbf{G} = [0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0]$.

2) Parity-check matrix: In the decoder, the parity checks are used to detect the presence of transmission errors in the received codeword $\hat{\mathbf{c}}$. Since all of the codeword bits involved in a parity check (including the parity bit itself) should have a modulo-2 summation of 0, Equations (7a)–(7d) can be re-written as follows:

$$c_4 \oplus c_6 \oplus c_7 = 0 \tag{10a}$$
$$c_1 \oplus c_3 \oplus c_4 \oplus c_6 \oplus c_8 = 0 \tag{10b}$$
$$c_2 \oplus c_5 \oplus c_9 = 0 \tag{10c}$$
$$c_1 \oplus c_2 \oplus c_6 \oplus c_{10} = 0. \tag{10d}$$

These equations are more commonly viewed as a PCM $\mathbf{H}$, which has N columns corresponding to the bits of the codeword and M rows corresponding to the parity checks. A non-zero entry in any position H_{ji} indicates that the i-th bit c_i takes part in the j-th parity check. In the case of systematic codes $\mathbf{H}$ is related to $\mathbf{G}$ according to

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}^T & \mathbf{I}_M \end{bmatrix}. \tag{11}$$

Continuing our example from above, we have

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{12}$$

Upon obtaining a received codeword $\hat{\mathbf{c}}$, the syndrome $\mathbf{s}$ can be calculated according to $\mathbf{s} = \hat{\mathbf{c}} \times \mathbf{H}^T$. In the case where $\hat{\mathbf{c}}$ is a legitimate codeword permutation, the syndrome will equate to a vector of zeros. This may be demonstrated by re-using the codeword calculated in the previous subsection, which equates to a $(1 \times M)$-element vector of 0s when multiplied by $\mathbf{H}^T$.

Note however that an LDPC $\mathbf{H}$ matrix of the form shown in (12) is very unusual in practice. As it will be explained in Section II-C1, the decoder's error correction ability is dictated by the number of non-zero entries in each row or column, which is referred to as its *weight*. More specifically, columns with a weight of 1 can result in the decoder being unable to correct some transmission errors. This can be avoided by modifying the PCM $\mathbf{H}$ using elementary row operations (modulo-2 additions and swaps). In the case of the above example, this may lead to:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{13}$$

This modified $\mathbf{H}$ avoids any weight-1 columns, while still checking the same distribution of parity bits that was added to codewords by the generator matrix $\mathbf{G}$ of (9). Note however that this toy-example PCM is still unusual for a realistic LDPC code. Specifically, the PCM used in LDPC decoding should be sparse, containing far fewer non-zero entries than 0s. Clearly, the $\mathbf{H}$ of (13) does not satisfy this constraint, owing to its codeword length of $N = 10$, which is very short compared to practical LDPC codewords, which tend to be hundreds or even thousands of bits long.

Owing to its significance in the decoding process, the PCM $\mathbf{H}$ is commonly used to define a particular LDPC code design. As discussed later in Section II-D, creating a $\mathbf{H}$ matrix that

achieves a strong error correction capability is a complex task, so this is usually the first aspect of the code to be designed. Following this, the generator matrix $\mathbf{G}$ can be derived from $\mathbf{H}$, by following the reverse of the process described above.

3) Factor graphs: The PCM $\mathbf{H}$ can also be visualised graphically using a factor graph, which is also known as a *Tanner graph* [18]. This is exemplified in Fig. 4 for the PCM of (13). A factor graph is comprised of two sets of connected nodes, namely N Variable Nodes (VNs) for representing the columns of $\mathbf{H}$ and M Check Nodes (CNs) for representing the rows.

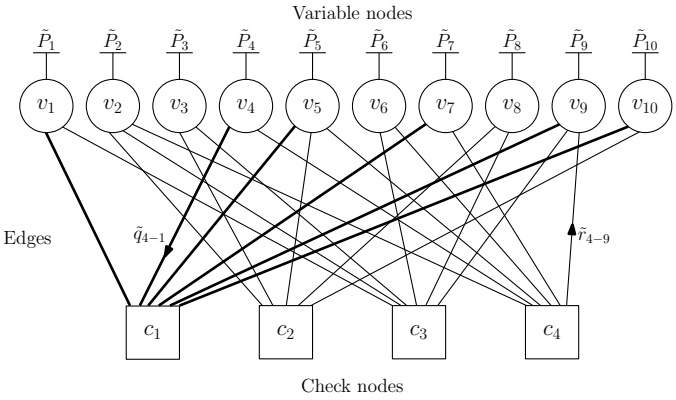


Fig. 4. A factor graph for an example LDPC code

The connections $\tilde{P}_i$ above each VN in Fig. 4 pertain to LLRs associated with the N codeword bits of $\hat{\mathbf{c}}$. An edge connects the i-th VN v_i to the j-th CN c_j if there is a non-zero element in the i-th column and j-th row of $\mathbf{H}$, $H_{ji} = 1$. To illustrate this, all of the edges that are connected to the 1st CN c_1 in Fig. 4 are shown with thicker lines. These edges are connected to the 1st, 4th, 5th, 9th and 10th VNs, in accordance to the position of the 1s in the top row of $\mathbf{H}$ in (13).

The *degree* of a node is defined as the number of other nodes that it is connected to and is equal to the corresponding row or column weight in $\mathbf{H}$. The degree of the CNs D_c and the degree of the VNs D_v are important parameters in an LDPC code. If all CNs have the same degree D_c and all VNs have the same degree D_v, the LDPC code is said to be *regular*. If either value varies from node to node, the code is said to be *irregular* and D_c and D_v can be expressed as the average degree over all nodes. For example, the factor graph of Fig. 4 is irregular with $D_c = 5.75$ and $D_v = 2.3$. In any case, the number of 1s in the PCM $\mathbf{H}$ must be the same regardless, whether it is viewed row-by-row or column-by-column, giving $D_c \times M = D_v \times N$, with $D_v = D_c \times (1 - R)$.

C. LDPC decoding

LDPC codes are typically decoded using a belief propagation (BP) algorithm in which messages – typically in the form of LLRs – are iteratively passed in both directions along the edges between connected nodes [19]. For example, Fig. 4 illustrates a message $\tilde{q}_{4-1}$ sent from the 4th VN v_4 to the 1st CN c_1, while the message $\tilde{r}_{4-9}$ is sent from the 4th CN c_4 to the 9th VN v_9. The messages provided as inputs to a node are processed by activating that node, causing it to create new output messages that are sent back to the nodes it is connected to. Thus the processing of the LDPC decoder is delegated to the many individual calculations performed by the individual nodes, rather than being a single monolithic global equation. An important facet of the belief propagation algorithm is that any message sent to a particular node does not depend on the message received from that node. For example, CN c_2 is connected to VNs v_2, v_3, v_5, v_8 and v_{10}; however, the message $\tilde{r}_{2-5}$ it sends to v_5 will be calculated based only on the messages it has received from v_2, v_3, v_8 and v_{10}.

Nodes are activated in an order determined by the LDPC decoder's *schedule*. This has a significant effect upon the LDPC decoder's error correction capability, as well as on its other characteristics. Many different schedules exist and the most common options will be outlined in Section II-C1. Following this, variations of the specific calculations performed within CNs and VNs will be presented in Sections II-C2 and II-C3 respectively.

1) Scheduling: The schedule of the LDPC decoding process determines the order in which VNs and CNs are processed, as well as whether multiple nodes are processed in parallel. Many scheduling variations exist, but the three most common schedules are described here, namely flooding [20], Layered Belief Propagation (LBP) [21] and Informed Dynamic Scheduling (IDS) [22].

Flooding is perhaps the most conceptually simple LDPC decoding schedule. Here, the factor graph is processed in an iterative manner, where each iteration comprises the simultaneous activation of all CNs, followed by the simultaneous activation of all VNs [19]. An example of this schedule is depicted in Fig. 5. It can be seen that at first the CNs c_1–c_4 shown in dark grey calculate their messages, which are then

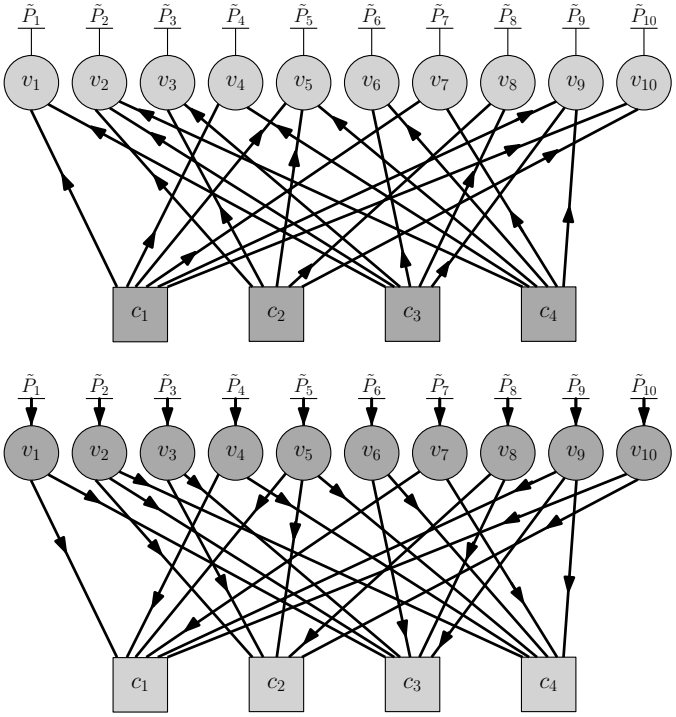


Fig. 5. An example of the flooding schedule

sent along every edge (in bold) to every receiving VN, shown in light grey. In the second half-iteration, the VNs are shown in dark grey to indicate that they are performing calculations, while the CNs are only receiving messages, so they are shown in light grey.

While **Layered Belief Propagation** also operates in an iterative manner, it processes the nodes more sequentially within each iteration, activating only one or a specific subset of nodes at a time [21]. LBP is commonly operated in a CN-centric manner, processing each CN in turn. Once a CN has been activated, all of its connected VNs are activated before moving on to the next CN. Once every CN has been processed, the iteration is complete. Using Fig. 6 as an example, LBP may commence each decoding iteration by activating CN c_1 first, sending messages to each of its connected VNs: v_1, v_4, v_5, v_7, v_9 and v_{10}. Each of these VNs may then be activated, sending new messages to each of their connected CNs, except c_1. Following this, c_2 may be activated, allowing it to make use of the new information received from v_5 and v_{10} alongside the information previously received from its other connected VNs. This process continues until every CN has been activated, which then marks the end of one decoding iteration.

LBP has the advantage that the information obtained during an iteration is available to aid the remainder of the iteration. Owing to this however, it does not have the same high level of parallelism as the flooding schedule, possibly resulting in a lower processing throughput and a higher processing latency. It can also be seen that M CN activations and $D_c \times M$ VN activations occur per iteration, resulting in a higher computational complexity per iteration, when compared to the flooding schedule. However, it will also be shown in Section II-C2 that CN activations can be significantly more computationally expensive than the VN activations, hence the increased cost is manageable. Additionally, LBP tends to converge to the correct codeword using fewer iterations and therefore with lower computational complexity than flooding [17], resulting in lower complexity overall.

Informed Dynamic Scheduling inspects the messages that are passed between the various nodes, selecting to activate whichever node is expected to offer the greatest improvement in belief [22]. This requires IDS to perform additional calculations in order to determine which node to activate at each stage of the decoding process. However, IDS facilitates convergence using fewer node activations than in either flooding or LBP, which may lead to a lower complexity overall.

During IDS, the difference between the previous message sent over an edge and the message that is obtained using recently-updated information [23] is calculated. This difference is termed the *residual*, and represents the improvement in belief that is achieved by the new message. Like the LBP schedule, IDS is commonly centred on the CNs. At the start of the iterative decoding process, the residual for each output of each CN is calculated as the magnitude of the message to be sent over that edge. The message with the greatest residual is identified, and the receiving VN is then activated, sending updated messages to each of its connected CNs. These CNs then calculate new residuals for each of their edges as the difference between its new message and its previous message.

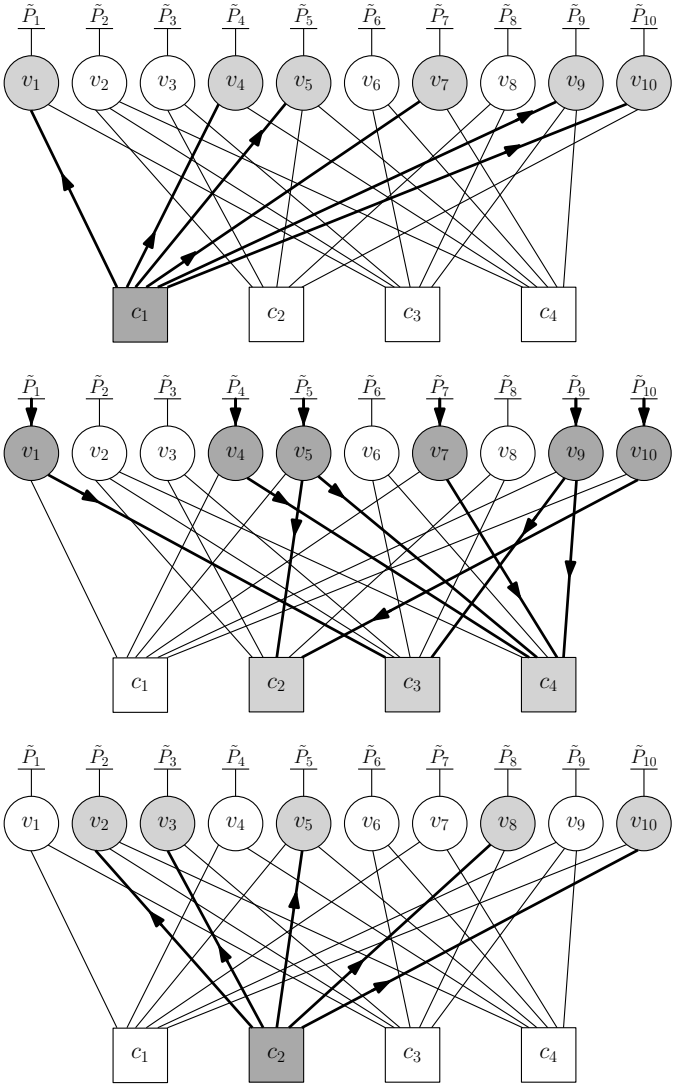


Fig. 6. An example of the layered belief propagation schedule

All of the residuals in the graph are then compared for the sake of identifying the new maximum, before the process is repeated.

Using Fig. 7 as before, suppose that at the start of the iterative decoding process, the message $\tilde{r}_{3-8}$ from CN c_3 is identified as having the highest magnitude of all the check-to-variable messages in the graph. Owing to this, $\tilde{r}_{3-8}$ is passed to the VN v_8, which is then activated, in order to obtain the message $\tilde{q}_{8-2}$ which is then passed to c_2. The CN c_2 can then be activated to calculate new residuals for its other four edges, as the difference between their previous messages and their new messages that have been obtained using the updated information from v_8. These new residuals are then compared with the others from the previous step, allowing a new global maximum to be identified, to inform the next step of the decoding process. Note that the next highest residual within the factor graph does not necessarily have to originate from the most recently updated CN c_2. In the example seen in Fig. 7, it can be seen that c_2 is activated to calculate residuals but it is $\tilde{r}_{1-4}$ from CN c_1 to VN v_4 that is sent. This implies

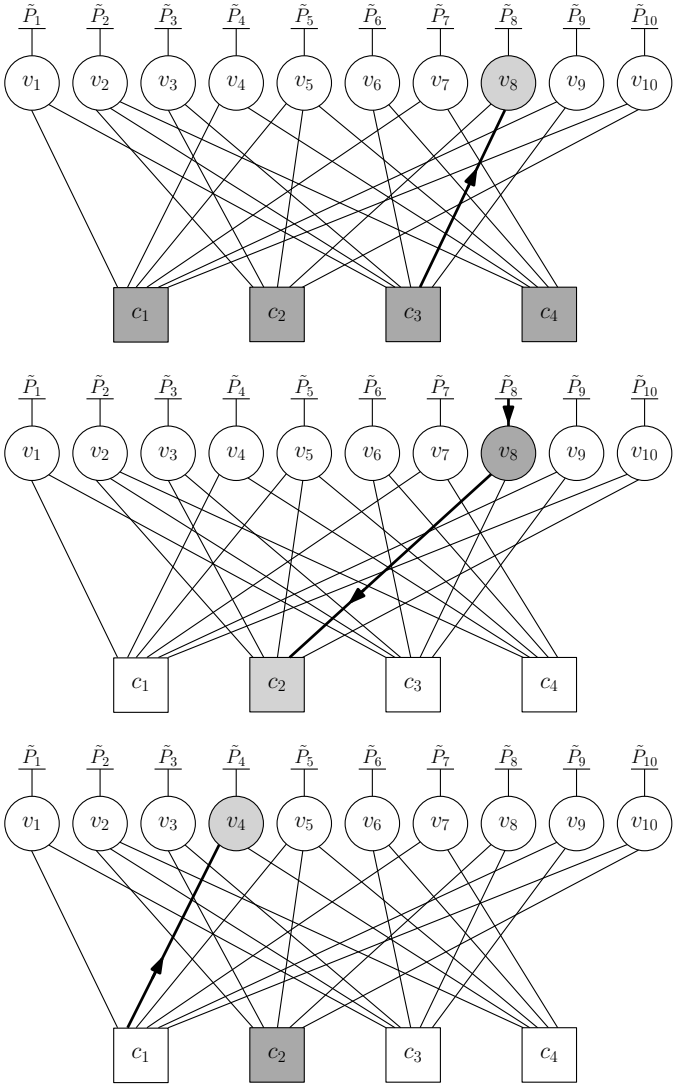


Fig. 7. An example of informed dynamic scheduling

that there is no single straightforward concept of iterations in IDS, since it is possible for a particular CN to be updated several times before another is updated once.

2) Check node calculations: The calculations performed within the CNs vary between different LDPC decoding algorithms. Of the many that exist, the two most common LDPC decoding algorithms are the Sum-Product Algorithm (SPA) [24] and the Min-Sum Algorithm (MSA) [25].

When the j-th CN is activated, the LLR $\tilde{r}_{j-i}$ that it passes to VN v_i is a function of the inputs gleaned from all other connected VNs, except for v_i [26]. In CN c_j, this message will represent the probabilities that the bit at v_i should be 0 or 1, which is determined by whether parity check j has already been fulfilled by the bits of the other connected VNs. This is achieved by calculating the probability that c_j is receiving an even number of 1s from its other edges. For two LLR operands $\tilde{a}$ and $\tilde{b}$ this equates to

$$\tilde{r}_{j-i} = \log \frac{P(a=0)P(b=0) + P(a=1)P(b=1)}{P(a=0)P(b=1) + P(a=1)P(b=0)}, \quad (14)$$

for which we use the notation $\tilde{a} \boxplus \tilde{b}$, referred to as the *boxplus* operator [27]. Inverting (4) and substituting into (14) yields:

$$\begin{aligned} \tilde{a} \boxplus \tilde{b} &= 2\tanh^{-1}\left(\tanh\frac{\tilde{a}}{2} \times \tanh\frac{\tilde{b}}{2}\right) \quad (15) \\ &= \text{sign}\left(\tilde{a}\right) \times \text{sign}\left(\tilde{b}\right) \times \min\left(|\tilde{a}|, |\tilde{b}|\right) \\ &\quad + \log\left(1 + e^{-|\tilde{a}+\tilde{b}|}\right) - \log\left(1 + e^{-|\tilde{a}-\tilde{b}|}\right). \quad (16) \end{aligned}$$

The SPA uses the full version of (15) given above, which leads to strong error correction performance but a high computational complexity. The MSA, on the other hand, is a reduced-complexity approximation of the SPA [28], using (16) without the correction factor terms, according to

$$\tilde{a} \boxplus \tilde{b} = \text{sign}\left(\tilde{a}\right) \times \text{sign}\left(\tilde{b}\right) \times \min\left(|\tilde{a}|, |\tilde{b}|\right). \quad (17)$$

Note however that the complexity reduction offered by the MSA is attained at the cost of a degraded LDPC error correction capability. This degradation may be mitigated by adding a low-complexity approximation to the correction factor terms to (17) or by multiplying (17) by a scaling factor, which may be optimised during the design of the LDPC decoder.

3) Variable node calculations: The calculations performed in the VNs do not generally vary between algorithms. As in the operation of the CNs, the message $\tilde{q}_{i-j}$ passed from VN v_i to CN c_j is obtained as the sum of the LLRs received from all other edges, including the LLR $\tilde{P}_i$ provided on the edge from the demodulator [26]. When using a schedule that requires the simultaneous update of the outputs provided to all of the VN's edges (such as the flooding schedule), the forward-backward algorithm [24] may be used to minimise the number of additions required by the VN. In other schedules, small internal memories may be used to store the results of some intermediate additions [21].

The VNs are also used for deciding the values of the reconstructed codeword bits. Each corresponding codeword LLR $\tilde{L}_i$ is calculated as the sum of the LLRs received on the edges from all connected CNs, as well as on the edge from the demodulator. The polarity of the resultant LLR is then used to make a hard decision for the value of the corresponding bit of the reconstructed codeword $\hat{\mathbf{c}}$. More specifically, if $\tilde{L}_i < 0$ then $\hat{c}_i$ is set to 1, whereas if $\tilde{L}_i \geq 0$ then $\hat{c}_i$ is set to 0. If the reconstructed codeword has a zero-valued syndrome $\mathbf{s} = \hat{\mathbf{c}} \times \mathbf{H}^T$, then the iterative decoding process may be considered to have been a success and the process may be terminated. If not, then the iterative decoding process may be continued until a zero-valued syndrome is obtained or until an affordable complexity limit is reached. Practical LDPC decoder designs may also include other stopping criteria, as discussed later in Section II-E3.

D. LDPC code construction

In addition to the size of the factor graph and the degrees of its nodes, the position of the edges within the factor graph also has a significant impact on the associated error correction performance, as well as upon the decoding complexity. Some of the main objectives when designing the PCM $\mathbf{H}$ is to avoid

creating stopping sets [29] and short cycles [30] in the corresponding factor graph, which are associated with an eroded error correction performance. A number of techniques have been proposed for placing edges within the factor graph have been proposed, as summarised in the following subsections.

1) Random codes: Unstructured randomly-designed codes potentially achieve the best LDPC error correction performance, owing to the maximised degree of freedom that is afforded, when placing edges in this manner [31]. However, this is achieved at the cost of having to implement complex unstructured routing or memory lookup tables, in order to exchange LLRs between the variable and CNs. A straightforward recursive algorithm for creating unstructured PCMs of this form involves placing a 1 at a random unfilled location in $\mathbf{H}$, then checking to see whether doing so has violated any design constraints, such as the maximum node degrees, stopping sets or cycle lengths. If the placement is valid, the algorithm will continue and repeat the process for the next randomly placed 1. This is repeated until the desired number of edges have been positioned. If a randomly placed 1 is not valid, then it will be rejected and a new location will be tried instead. This algorithm is conceptually very simple, but whether the process can successfully complete and how quickly is unpredictable.

2) Pseudorandom codes: The original LDPC code construction method proposed by Gallager [1] involves stacking D_c number of submatrices on top of each other. Each submatrix has the dimensions $^{M}/_{D_c} \times N$, with each column having a weight of 1 and each row having a weight of D_v. The top matrix is pseudo-randomly generated, and random column permutations are applied to it in order to obtain all other submatrices.

Similarly to this, Mackay [2] proposed a code construction method, which involves constructing the PCM $\mathbf{H}$ on a column-by-column basis, where the columns are generated pseudo-randomly with appropriate weight, before being concatenated horizontally. Again, this process must be performed in a recursive manner, so that the row weights can be checked after each column is added. If D_c has been exceeded for any row, then the current column is regenerated.

3) Quasi-cyclic codes: An LDPC code wherein the cyclic shift of any legitimate codeword permutation by s places to the left or right yields another legitimate codeword permutation is termed *Quasi-Cyclic* (QC), while the code is termed *cyclic* in the special case of $s = 1$. The PCMs of QC codes are semi-structured, based on an upper matrix of elements which each represent an equally-sized square submatrix [32]. If a particular element in the upper matrix has a value of -1, then the corresponding submatrix is a null matrix. Otherwise, the submatrix is an identity matrix, which has been cyclically shifted a number of times according to the corresponding value in the upper matrix [33]. Adopting this structure facilitates low complexity memory addressing and routing for the hardware implementation, since the location of every edge in each submatrix can be determined using only knowledge of the relatively small upper matrix. This advantage can be achieved without incurring a significant sacrifice in error correction performance. Owing to this benefit, QC-LDPC codes are employed by a number of communications standards, including DVB-S2 [5], IEEE 802.11 (WiFi) [3] and IEEE 802.16 (Mobile WiMAX) [4].

4) Repeat-accumulate codes: Repeat-accumulate (RA) codes constitute another type of semi-structured codes. Like QC codes, RA codes benefit from simpler encoding/decoding than random codes, without imposing an unacceptable loss in error correction performance. The PCMs of RA codes are composed of two horizontally-concatenated submatrices $\mathbf{H}_1$ and $\mathbf{H}_2$, where $\mathbf{H}_2$ is an $(M \times M)$-element dual-diagonal matrix. This structure allows each parity bit to be calculated using only the previous parity bit and a subset of the message bits, leading to the accumulation alluded to in the code's name.

5) Progressive edge growth algorithm: Whilst not a code structure itself, the Progressive Edge Growth (PEG) algorithm [34] is an important technique of constructing codes having an excellent error correction performance. The operation of the PEG algorithm is VN-centric, focusing on each VN in turn in order to place edges. The algorithm repeatedly constructs a set of CNs as candidates for the VN to connect to. From this set, the subset of nodes having the lowest degree is extracted and one of these is randomly selected. This approach results in LDPC codes that have approximately regular degree distributions.

The PEG algorithm constructs a tree structure, alternating between the connection of VNs to CNs and vice versa. At each stage only nodes that are not already in the tree are considered for inclusion. This process continues until there are no remaining options meeting this constraint. The PEG algorithm then places an edge in the location that is identified as maximising the length of the resultant cycle within the graph, before continuing the algorithm with the selection of a different VN. In this way, a factor graph having no short cycles can be created, yielding a strong error correction performance.

E. LDPC decoding architectures

The implementation of a practical LDPC decoder is subject to numerous design decisions, such as the degree of parallelism, the representation of the LLRs and the stopping criteria. These three factors are discussed in the following subsections.

1) Parallelism: The inherent parallelism of the belief propagation algorithm facilitates the design of fully-parallel LDPC decoder architectures, in which every VN and CN in the factor graph is implemented separately in hardware [35]. Fully-parallel decoders can achieve very high processing throughputs by performing all of the VN updates and all of the CN updates simultaneously, using the flooding schedule of Fig. 5. However, this is achieved at the cost of excessive hardware resource consumption. For long codes comprising thousands of bits, the inter-node routing may require a greater area than the nodes themselves [36], rendering this architecture impractical for many decoder designs. Additionally, significant further hardware resources are required for implementing flexible routing, using a Beneš network [37], for example. Otherwise, fully-parallel decoders are completely inflexible, only supporting the single code that they are designed for.

By contrast, decoders associated with a fully-serial architecture implement just a single one of each node type in hardware. This hardware is time-multiplexed between the various

nodes of the LDPC decoder, using memories to store interim results [35]. Fully-serial decoders require few hardware resources but suffer from a very low processing throughput, since each decoding iteration could require thousands of clock cycles. However, since all of the factor graph edges are represented by memory addresses, fully-serial decoders can be readily adapted at run-time to implement a different LDPC factor graph, by rearranging the memory accesses.

In order to strike a compromise between the high processing throughput of fully-parallel architectures and the more modest hardware requirement of fully-serial architectures, many LDPC decoders implement a number of time-multiplexed nodes in a so-called partially-parallel fashion. This parametrizable degree of parallelism facilitates control over the trade-off between processing throughput and hardware resource requirements. Furthermore, this approach is of particular benefit when any structure within the PCM $\mathbf{H}$ can be exploited in the configuration of the nodes implemented in hardware. For this reason, QC codes are particularly suited to partially-parallel implementations.

2) Representation of LLRs: Another architectural consideration is the digital representation of the LLRs passed between nodes. The algorithms described earlier can be modified to replace the LLRs with single-bit hard decisions, but this causes them to suffer from a significant error correction performance loss. In general, increasing the resolution and range of the two's complement fixed point LLR representation by using a greater bit width has a positive effect on the error correction performance [38], at the cost of increasing the hardware resources required.

It is therefore desirable for a designer to quantify the effect of the fixed-point bit width on the performance of a chosen decoding algorithm, in order to determine the smallest number of bits that are required in order to achieve a satisfactory error correction performance. This may be achieved using Extrinsic Information Transfer (EXIT) charts [39], which have been conceived by ten Brink for characterising the operation of iterative decoding algorithms. More specifically, EXIT charts visualize the quality of the LLRs output by the VNs and CNs as functions of the quality of the LLRs provided to the corresponding inputs. By plotting these EXIT functions for LDPC decoders employing a range of fixed-point bit widths, a designer can quantify at a glance, how each representation improves or degrades the quality of the LLRs and hence the resultant error correction performance of the LDPC decoder [40]. This eliminates the requirement to run multiple time-consuming BER simulations.

Further to this, some designs have demonstrated that the hardware requirement can be reduced by using non-uniform quantisation schemes [41], by sending the bits of the LLR in a serial fashion rather than in parallel [42], or by utilising stochastic [36] or non-binary [43] number representations. However, these methods can also have adverse effects on the node complexity and the decoding throughput, requiring yet further investigation.

3) Stopping criteria: The design of an LDPC decoder also has to consider how to terminate the decoding process. Commonly, checks are carried out following each decoding iteration to determine whether the current state of the recovered codeword is a legitimate permutation or not, signalling whether or not decoding has been successful. These checks are performed based on the output of the VNs, as mentioned previously in Section II-C3.

Occasionally however, a received frame is corrupted in such a way that it can never be corrected. In this case, the iterative decoding process would loop infinitely, unless other criteria for stopping it were implemented. Owing to this, a maximum iteration or complexity limit may be imposed. When this limit is reached, the iterative decoding process is terminated and decoding is deemed to have failed. In implementations where a low hardware resource requirement is a greater priority than high processing throughput, the iteration limit may be the only stopping criterion imposed. Here, every received message is decoded using the same number of iterations, without early stopping. In this case, the parity checks are only used at the end of the iterative decoding process, in order to determine whether the recovered codeword is valid or not. Early stopping can also be used to detect that no error correction progress is being made with successive decoding iterations, allowing the decoding process to fail and terminate before the iteration limit is reached.

F. FPGAs

FPGAs are digital logic devices that can be flexibly programmed to perform a variety of digital functions, using a Hardware Description Language (HDL). Their main advantages are their in-field-programmability, as well as their high-speed very-parallel logic processing. Owing to these benefits, FPGAs are desirable for a multitude of applications, including software-defined radio, ASIC prototyping, digital signal processing, cryptography and computer hardware emulation. This section presents a simplified view of their internal structure, followed by a discussion of the main differences and similarities between different makes and models of FPGAs, and how they may be compared to each other.

1) Structure: The internal structure of an FPGA typically comprises a variable number of three main programmable elements, namely logic blocks, RAM blocks and I/O blocks [44]. The inputs and outputs of these blocks are linked by programmable routing, as shown in the sample schematic of Fig. 8.

The most fundamental design of a logic block comprises a Lookup Table (LUT) and a Flip-Flop (FF), as shown in Fig. 8. A LUT is a digital structure that can be programmed to perform any combinatorial function of its inputs, thus mimicking any possible combination of logic gates. Typically, FPGA LUTs have 4–6 inputs, which are used to select a value for a single output bit. Increasing the number of LUT inputs typically allows the same HDL design to be implemented using fewer LUTs, therefore reducing the amount of FPGA routing required. However, the hardware resources required by a LUT increase exponentially with its number of input bits, hence very large LUTs are impractical [44]. The output of each LUT can optionally be connected to a corresponding FF, for facilitating synchronous operation. Alternatively, the LUT

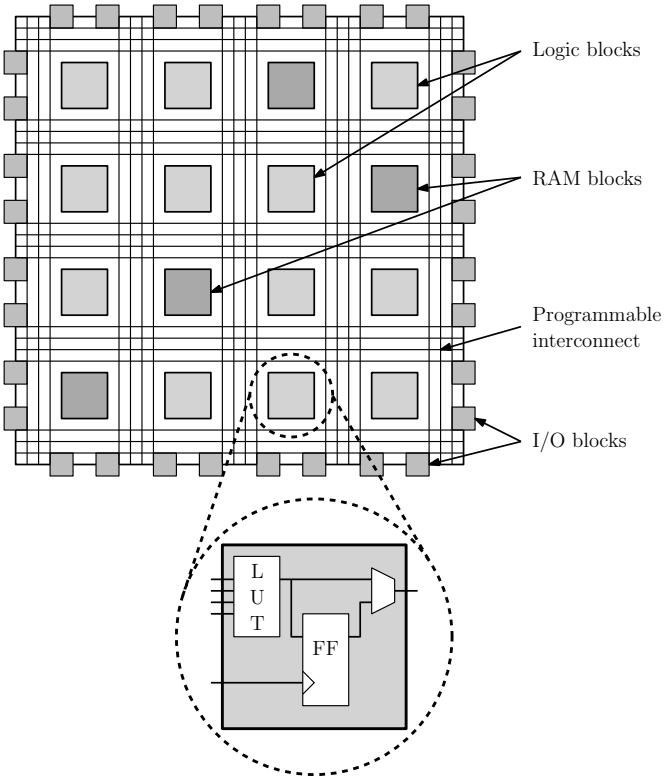


Fig. 8. FPGA structure

output can be connected directly to the inter-block routing channels. These channels can be programmed to connect any set of logic block outputs to any set of logic block inputs, subject to the FPGA size constraints.

Instead of logic blocks, some locations within an FPGA structure may contain a RAM block for storing intermediate calculation results. The size of these RAM blocks depends on the particular FPGA being used, as does their access control. More specifically, some FPGAs provide dual-port RAMs, which allow reading and writing to two different locations simultaneously. Some FPGAs may also include additional heterogeneous blocks, such as hardware multipliers and embedded processor cores, alongside non-volatile memory for storing the FPGA configuration when it is turned off [44].

2) FPGA vendor conventions: The two main vendors of FPGAs are Xilinx and Altera. Their respective FPGAs share a number of similarities, but also exhibit some differences. Some Altera FPGAs, such as the first four generations of the Cyclone family, follow the structure outlined above, operating on the basis of so-called "Logic Elements" (LEs), each of which comprises one 4-input Lookup Table (4LUT) and one FF. However, more recent Altera FPGAs are structured around "Adaptive Logic Modules" (ALMs), each of which comprises two FFs and multiple small LUTs. These ALMs also contain extra logic that optionally allows the LUTs to be combined in a variety of ways, offering the functionality of larger LUTs.

By contrast, the logic resources of Xilinx FPGAs are quantified in terms of "slices", each of which contains several LUTs and FFs. The nature and quantity of the hardware resources available within each slice varies depending on the model and generation of the FPGA. Earlier models of Xilinx FPGAs, such as the Virtex 2, employ a simple slice structure which is based on 4LUTs, while more recent models utilise 6-input Lookup Tables (6LUTs) and a more complex slice structure that allows them to be used in a larger number of configurations.

3) Comparing FPGAs: Due to the differences outlined above, comparing the hardware resources employed by the various designs implemented on different FPGAs is not straightforward. To this end, we propose an approximate metric based on the fundamental building blocks of FPGAs, namely the 4LUT and the FF [45]. We refer to this metric as *equivalent logic blocks* (ELBs), since it attempts to approximate the number of simple logic blocks comprising one 4LUT and one FF that would be required to implement each design. When calculating the number of ELBs, the LUT resources within each Altera ALM are considered to be equal to two 4LUTs, since this is one of the operating modes they offer. Similarly, to compensate for the increased size of 6LUTs compared to 4LUTs, and the additional logic that accompanies 6LUTs within Xilinx slices, each Xilinx 6LUT is considered to be approximately equal to two 4LUTs. Once this has been taken into consideration, we assume that the number of ELBs required by a design is given by the maximum of the number 4LUTs and the number of FFs that it requires.

Table I presents an overview of the main generations and models of FPGAs available from Altera and Xilinx, along with the year of their release and the maximum number of ELBs available within the largest FPGA from each family.

TABLE I: Comparison of FPGAs available from Altera and Xilinx

Manufacturer	Model	Year	Technology scale (nm)	Max. ELBs
Xilinx	Virtex	1998	220	24,576
Xilinx	Virtex E	1999	180	64,896
Xilinx	Virtex 2	2000	150	93,184
Altera	Cyclone	2002	130	20,060
Altera	Stratix	2002	130	79,040
Xilinx	Spartan 3	2003	90	66,560
Altera	Cyclone 2	2004	90	68,416
Altera	Stratix 2	2004	90	143,520
Xilinx	Virtex 4	2004	90	178,176
Altera	Stratix 3	2006	65	270,000
Xilinx	Virtex 5	2006	65	414,720
Altera	Arria	2007	90	72,172
Altera	Cyclone 3	2007	65	198,464
Altera	Stratix 4	2008	40	650,440
Altera	Cyclone 4	2009	60	149,760
Altera	Arria 2	2009	40	278,800
Xilinx	Spartan 6	2009	45	184,304
Xilinx	Virtex 6	2009	40	948,480
Altera	Stratix 5	2010	28	718,400
Altera	Cyclone 5	2011	28	227,120
Altera	Arria 5	2011	28	380,480
Xilinx	Artix 7	2011	28	269,200
Xilinx	Kintex 7	2011	28	597,200
Xilinx	Virtex 7	2011	28	1,424,000

Note that the proposed ELB metric is by no means perfect, since it does not consider the overhead associated with

routing between logic elements or the use of additional FPGA blocks, such as memory or embedded multipliers. However, it does serve as a functional approximation of the hardware requirements associated with each design considered, if they were all implemented on the same FPGA. Measuring the usage only in terms of these fundamental building blocks permits a comparison between modern FPGA models and much older designs, which would not otherwise be possible.

III. Comparison of decoders

A comprehensive review of published FPGA-based LDPC decoder designs is presented in this section. The analysis of Table II considers both the parameters that are chosen by the designers, as well as the characteristics that may be measured based on the design. Each of these is discussed and characterised in Sections III-A and III-B, together with explanations and discussions of the symbols used in Table II where applicable. The entries in Table II have been sourced from both academic publications and commercially-available soft IP cores. Unfortunately, the licensers of these commercial designs were often unwilling to divulge many of the parameters and characteristics required for this analysis, resulting in several incomplete sets of results. Furthermore, none of the licensers were willing to provide pricing information for the purposes of this survey, preventing the comparison of this interesting but non-technical characteristic of their IP.

Note that Table II presents a condensed version of our findings, showing only the most significant parameters and characteristics. In the case of references that present multiple FPGA-based LDPC decoder designs, only a representative subset has been reproduced here. A full version of our survey results may be downloaded from [46].

A. Parameters

In this section, we consider the parameters of FPGA-based LDPC decoders, which include all factors of the design that are specified by the designer. These include which LDPC PCMs to support, the decoding algorithm to employ and the number of decoding iterations used. These parameters are discussed in Sections III-A1, III-A3 and III-A4 respectively. Section III-A2 describes the architectural parameters, namely the degree of parallelism, LLR representation, clock frequency, flexibility and choice of FPGA.

1) LDPC PCMs: One of the most fundamental features of an LDPC decoder is the selection of the PCMs that it is designed to support. Decoders may support just one PCM, be tailored to a family of related PCMs or may be designed to be completely flexible. As discussed in Section II-B, each PCM $\mathbf{H}$ has a number of parameters, namely N, M, D_c and D_v. However, the total number of edges in the corresponding factor graph can be considered to encompass all of these factors, representing the overall size and complexity of the code, as listed in Table II.

2) Architecture: Architectural decisions influence the physical implementation and hardware used by the decoder. As described in Section II-E, the primary architectural parameter is the degree of parallelism, which may be classified as fully-parallel, partially-parallel or fully-serial. This parameter may be quantified by the total number of Processing Units (PUs) instantiated by the decoder, as listed in Table II. Frequently these processors perform the function of individual VNs and CNs, although some designs use a different approach.

The operand width of the LLR representation, as listed in Table II, is also a measurable parameter, which affects the LDPC decoder's error correction performance. Designs using a higher number of bits may be expected to have superior error correction performance than their counterparts employing fewer bits. However, this is typically achieved at the cost of a larger hardware resource requirement or a lower processing throughput.

The quantisation scheme used in the LLR representation may be either uniform or non-uniform, as denoted by a 'U' or an 'N' in Table II, respectively. In uniform quantisation schemes, the entire range of representable LLR values has a constant resolution, allowing the VN and CN functions to be implemented using straightforward binary arithmetic. By contrast, non-uniform quantisation schemes typically adopt a finer resolution for lower LLR magnitudes and a lower resolution for larger magnitudes. This facilitates a more beneficial trade-off between range and resolution, but makes the associated processing significantly more complex. Many authors (e.g. [42], [47], [48]) mention the number of bits used in their FPGA-based LDPC decoders, but do not detail the quantisation scheme employed. Since non-uniform schemes require significantly more details than uniform representations, these cases are assumed to employ uniform quantisation and are marked with an asterisk in Table II.

The maximum achievable clock frequency of an FPGA-based LDPC decoder depends largely on the capabilities of the FPGA employed, but also on some design decisions such as the critical path length. For example, designs that process entire VNs or CNs in a single clock cycle typically have long critical paths, while designs that only perform one arithmetic or logical operation per clock cycle typically have much shorter critical paths. Based on this observation, the clock frequency is included as a parameter in this analysis. The majority of authors have explicitly stated the clock frequency at which their decoder operates. However, in some cases (eg. [76]) we have derived the clock frequency from other data, as indicated by an asterisk in Table II.

Many decoder architectures are highly optimised to the specific characteristics of the single LDPC PCM that they are designed to support (eg. [43], [62], [74]). By contrast, some other designs instead adopt a more general architecture (eg. [57], [60], [66]), sacrificing performance for the flexibility to switch between several supported PCMs at run-time. A decoder's flexibility may be considered to be both a figure of merit and an architectural decision that is made by the designer, allowing it to be regarded as a characteristic or as a parameter. However, we show in Section IV-A that adding flexibility to a design can only be achieved as a trade-off against some other desirable characteristics. For this reason, we treat flexibility as a characteristic in this paper.

The selection of an FPGA for the implementation of an

TABLE II: Comparison of FPGA LDPC decoders

Ref.	H dimensions	Edges (k)	PUs	LLR bitwidth	Clock (MHz)	FPGA	Algorithm	Iterations	Throughput (decoded bps)	Throughput (encoded bps)	ELBs (k)	E_b/N_0 (dB)	Bandwidth eff.	Run-time flexibility
[41]	1022 x 8176	32.7	36	N 6	193	Xilinx Virtex 2	Sum-product (modified)	15	172 M	197 M*	38.3 *	3.78	0. 88	None
[49]	4608 x 9216	27.6	54	U* 5	56	Xilinx Virtex E	-	18	27 M*	54 M	15.9 *	1.85	0. 50	None
[42]	125 x 480	-	605	U* 3	61	Altera Stratix	Min-sum (modified)	15	481 M*	650 M	66.6	4.69	0. 74	None
[50]	432 x 1440	4.32	576	U 4	138	Altera Stratix 2	Min-sum	16 **	668 M**	954 M*	31.6 *	3.41	0. 70	None
[51]	4044 x 8088	-	24	U 6	44	Xilinx Virtex 2	Log-BP	25	40 M	80 M*	72.6 *	-	0. 50	None
[52]	1022 x 8176	32.7	18	-	200	Altera Cyclone 2	BP-based	18	70 M	80 M*	8.0 *	3.8	0. 88	None
[52]	1022 x 8176	32.7	18	-	200	Altera Stratix 2	BP-based	18	560 M	640 M*	38.0 *	3.8	0. 88	None
[53]	648 x 1296	3.89	36	U* 4	128	Xilinx Virtex 2	Min-sum	8.4 **	86.6 M**	173 M*	10.8 *	2.74	0. 50	3 codes in 802.11 WiFi
[54]	1728 x 3200	-	32	U* 5	180	Xilinx Virtex 4	Turbo decoding algorithm	10	103 M*	223 M	10.7 *	2.33	0. 46	None
[55]	6144 x 12288	36.9	288	U 6	96	Altera Stratix 2	Min-sum	15	149 M*	298 M	92.0 **	1.48	0. 50	None
[56]	576 x 1152	3.46	32	U 2	64	Xilinx Virtex 2	Min-sum (modified)	8.5 **	38 M**	76.1 M*	5.2 *	3.27	0. 50	3 codes in 802.16 WiMAX
[45]	512 x 1024	3.07	1536	NA	212	Xilinx Virtex 4	Stochastic	NA	353 M*	706 M	54.5 **	2.43	0. 50	None
[28] T	1022 x 8176	3.46	1728	U 2	138	Xilinx Virtex 5	Min-sum (modified)	6.8	11.7 G	23.4 G*	78.0 *	-	0. 50	None
[28] E	1022 x 8176	3.46	1728	U 2	138	Xilinx Virtex 5	Min-sum (modified)	8.5 **	9.35 G**	18.7 G*	78.0 *	3.43	0. 50	None
[57]	-	-	8	U* 6	160	Xilinx Virtex 5	-	20	21.6 M	25.9 M*	38.0 *	-	0. 83	Complete 802.16 WiMAX
[58]	188 x 2209	8.84	235	U* 6	50	Altera Stratix	Min-sum (modified)	20	108 M*	118 M	23.5 *	5.12 **	0. 92	None
[59]	1552 x 3104	-	48	N 6	98	Altera Cyclone 2	Belief propagation	30	26 M*	52 M	33.0	1.48	0. 50	None
[60]	731 x 4161	16.6	2	U 9	-	Xilinx Virtex 2P	Min-sum with correction	10 *	1.45 M		1.2 **	3.52 **	0. 82	Any code
[61]	4158 x 9036	27	54	U 6	100	Xilinx Virtex 2	Min-sum with correction	60 *	15 M	30 M*	53.3 *	1.17 *	0. 50	3 custom codes
[17]	768 x 1536	4.61	144	U 5	211	Xilinx Virtex 4	Min-sum with scaling	-	397 M	794 M*	18.2 *	-	0. 50	None
[17]	432 x 1296	4.75	81	U 8	160	Xilinx Virtex 4	Min-sum with correction	15 *	95 M	143 M*	19.3 *	2.49 *	0. 67	Complete 802.11 WiFi
[62]	2304 x 1152	-	12	U* 7	155	Altera Stratix 2	Min-sum	8	233 M	465 M*	17.3 *	1.94	0. 50	None
[62]	2304 x 1152	-	12	U* 7	128	Altera Stratix 2	Min-sum	8	768 M	1.54 G*	69.1 *	1.94	0. 50	None
[63]	3048 x 6096	-	72	-	64	Xilinx Virtex E	-	24	32 M*	64 M	12.3 *	-	0. 50	None
[64]	3600 x 16200	45	45	-	70.8	Xilinx Virtex 2P	Min-sum with scaling	15 *	36.3 M*	46.7 M	22.0 *	-	0. 78	Complete DVB-S2
[64]	3600 x 16200	45	180	-	73.2	Xilinx Virtex 2P	Min-sum with scaling	15 *	149 M*	191 M	70.6 *	-	0. 78	Complete DVB-S2
[47]	519 x 1038	3.11	9	U* 4	26.3	Xilinx Virtex E	-	18	36 M*	72 M	19.4 *	-	0. 50	None
[65]	4095 x 4095	262	130	U 1	191	Xilinx Virtex E	Soft majority logic	5	1.56 G*	1.9 G	21.3 **	4.36	0. 82	None
[66]	-	-	32	U 8	74	Xilinx Virtex 4	Normalized BP-based	15	5 M**	10 M*	30.6 **	-	0. 50	Any code

TABLE II: Comparison of FPGA LDPC decoders *(continued...)*

Ref.	H dimensions	Edges (k)	PUs	LLR bitwidth	Clock (MHz)	FPGA	Algorithm	Iterations	Throughput (decoded bps)	Throughput (encoded bps)	ELBs (k)	E_b/N_0 (dB)	Bandwidth eff.	Run-time flexibility
[67] T	324 x 648	1.94	972	U 1	188	Xilinx Virtex 5	Simplified MP	3.8	16.2 G	32.4 G*	28.5 *	-	0. 50	None
[67] E	324 x 648	1.94	972	U 1	188	Xilinx Virtex 5	Simplified MP	10	6.16 G**	12.3 G*	28.5 *	5.41	0. 50	None
[68]	600 x 1200	3.6	1800	U 2	123	Xilinx Virtex 4	Min-sum (modified)	8.9 **	8.3 G**	16.6 G*	58.1 *	3.38	0. 50	None
[68]	324 x 648	1.94	972	U 2	113	Xilinx Virtex 5	Min-sum (modified)	8.4 **	4.33 G**	8.67 G*	44.0 *	3.64	0. 50	None
[69] T	640 x 1920	6.4	40	U 8	150	Xilinx Virtex 4	Joint row-column	8	1.33 G*	2 G	-	-	0. 66	None
[69] E	640 x 1920	6.4	40	U 8	150	Xilinx Virtex 4	Joint row-column	50	211 M*	320 M**	-	2.29	0. 66	None
[69] T	640 x 1920	6.4	1	U 8	300	Xilinx Virtex 4	Joint row-column	8	74.6 M*	112 M	-	-	0. 66	None
[69] E	640 x 1920	6.4	1	U 8	300	Xilinx Virtex 4	Joint row-column	50	11.8 M*	17.9 M**	-	2.29	0. 66	None
[43]	486 x 972	2.43	18	-	131	Xilinx Virtex 4	Min-sum (nonbinary)	20	50 M		87.8 *	2.49	0. 50	None
[70]	-	-	27	U 8	100	Altera Cyclone 2	Min-sum (modified)	-	175 M*	350 M	13.5 *	-	0. 50	Within code families
[36]	528 x 1056	3.34	1584	NA	222	Xilinx Virtex 4	Stochastic	NA	348 M*	697 M**	68.2 *	2.45	0. 50	None
[71]	1022 x 8176	4.75	12	U 11	228	Xilinx Virtex 5	Sum-product (modified)	-	522 M		29.7 **	2.61	0. 66	None
[72]	384 x 2048	12.3	33	U 6	100	Xilinx Virtex 2P	Message passing	10	19.5 M*	24 M**	13.7 *	4.07 ***	0. 81	None
[73]	768 x 1536	4.61	9	U 8	162	Xilinx Virtex 2	Min-sum	3	114 M*	229 M	2.4 *	-	0. 50	None
[73]	756 x 3969	16.8	-	U 6	200	Xilinx Virtex 4	Normalized min-sum	15	82.4 M*	102 M	10.0 *	-	0. 81	None
[73]	1022 x 8176	32.7	144	U 6	212	Xilinx Virtex 4	Normalized min-sum	15	625 M*	714 M	27.2 *	3.76	0. 88	None
[74]	600 x 1200	3.6	1800	U 3	100	Xilinx Virtex 4	Min-sum	10	6 G	12 G*	69.0 *	3.76	0. 50	None
[75]	768 x 1536	4.61	9	U 8	149	Xilinx Virtex 2	Min-sum	3	49.6 M*	99.1 M	2.9 *	-	0. 50	None
[48]	768 x 1536	4.61	9	U* 8	100	Xilinx Virtex 2	Min-sum	7	5.88 M*	11.8 M**	1.8 *	3.36 ***	0. 50	None
[76]	768 x 1536	4.61	9	U 8	84 *	Xilinx Virtex 2	Sum-product (modified)	20	4.3 M*	8.59 M**	3.9 *	2.37	0. 50	None
[76]	768 x 1536	4.61	9	U 8	79.1 *	Xilinx Virtex 2	Sum-product (modified)	20	4.04 M*	8.08 M**	3.4 *	2.37	0. 50	None
[76]	768 x 1536	4.61	9	N 8	80.5 *	Xilinx Virtex 2	Sum-product (modified)	20	4.21 M*	8.42 M**	4.9 *	2.96	0. 50	None
[77]	336 x 672	2.18	96	U 5	100	Xilinx Virtex 5	Min-sum with correction	10	475 M*	950 M	71.4 *	3.02	0. 50	Within code families
[78]	-	-	-	U* 4	27	Xilinx Virtex E	-	-	15 M		1.7 *	-	-	None
[79]	768 x 1536	4.61	144	U 5	121	Xilinx Virtex 2	Min-sum (modified)	20	63.5 M*	127 M	20.4 *	2.79	0. 50	None
[80]	298 x 980	2.83	2	U 6	136	Altera Cyclone	Improved BP	-	7 M		1.0	4.4 **	0. 70	None
[81]	-	-	-	-	140	Xilinx Virtex 5	Min-sum	-	96 M		210 *	-	0. 67	Complete DVB-S2
[81]	-	-	-	-	140	Xilinx Virtex 5	Min-sum	-	206 M		388 *	-	0. 25	Complete DVB-S2
[82]	-	-	-	U* 9	180	Xilinx Virtex 5	Min-sum with correction	5	600 M	-	64.6 **	3.84	-	Complete CCSDS-C2

TABLE II: Comparison of FPGA LDPC decoders *(continued...)*

Ref.	H dimensions	Edges (k)	PUs	LLR bitwidth	Clock (MHz)	FPGA	Algorithm	Iterations	Throughput (decoded bps)	Throughput (encoded bps)	ELBs (k)	E_b/N_0 (dB)	Bandwidth eff.	Run-time flexibility
[82]	-	-	-	U* 9	150	Altera Stratix 2	Min-sum with correction	12	30 M	-	8.8 **	3.71	-	Complete CCSDS-C2
[83]	1152 x 2304	7.30	-	U* 8	160	Xilinx Virtex 4	Min-sum with correction	15	71 M	142 M*	36.4 **	1.84	0. 50	Complete 802.16 WiMAX
[83]	1920 x 2304	7.68	-	U* 8	160	Xilinx Virtex 4	Min-sum with correction	15	169 M	204 M*	36.4 **	3.66	0. 83	Complete 802.16 WiMAX
[84]	-	-	-	-	240	-	Min-sum with correction	-	866 M		-	-	-	Complete 802.11 WiFi
[85]	-	-	-	-	-	-	-	-	1 G	-	-	-	-	Complete ITU G.hn
[86]	-	-	-	-	-	-	-	10	-	2.1 G	-	-	-	Complete 802.11ad WiGig
[87]	2048 x 32640	-	-	U* 1	126	Altera Cyclone 4	Hard decision	12	8 G	8.54 G*	40.0	-	0. 94	None
[88]	-	-	-	-	-	Xilinx Spartan 6	-	-	200 M		101 *	-	-	320 custom codes
[88]	-	-	-	-	-	Xilinx Spartan 6	-	-	25 M		22.4 *	-	-	320 custom codes
[89]	972 x 1944	-	-	-	-	-	Min-sum with correction	15	-	-	-	1.66	0. 50	Complete 802.11 WiFi

LDPC decoder may have a significant impact upon its performance. The selected FPGA dictates the number of logic elements, memory blocks and I/O pins that are available for all processing and routing. Additionally, some FPGAs facilitate higher clock frequencies than others when implementing the same design, depending on the process technology employed. Unfortunately it is impossible to fairly compare the capabilities of all FPGAs numerically. For this reason, Table II simply states which FPGA has been employed for each LDPC decoder considered.

3) Algorithm: As discussed in Section II-C, several variations of the LDPC decoding algorithm exist. Some algorithms vary from each other only slightly, while others may employ vastly different mathematical concepts. Furthermore, different authors may use different terms to describe the same algorithm, making this parameter difficult to compare. Table II therefore only includes the terms used by the authors to describe their algorithms and no direct comparison between them is inferred.

4) Iterations: The limit placed on the maximum number of decoding iterations has a significant effect upon the processing throughput and error correction performance of decoders operating without early stopping functionality, as well as in cases where the received frame is too corrupted to be decoded successfully. Decreasing the maximum number of iterations will increase the LDPC decoder's processing throughput in terms of the maximum achievable bitrate, but runs the risk of allowing errors to remain in the recovered codeword that could have otherwise been corrected. Generally, it can be assumed that the number of iterations used in each considered design was selected by the author to offer the most desirable trade-off between error correction performance and processing throughput, subject to the influence of the other parameters outlined above. It is also worth noting that the maximum number of iterations is perhaps the easiest parameter to change at runtime. Owing to this, some designs (eg. [28], [67], [69]) are presented with two sets of results, namely one employing a low number of iterations for maximum processing throughput (marked with a 'T' in Table II), and one with a high number for maximum error correction (marked with an 'E' in Table II).

Table II presents the fixed number of iterations that are employed in designs without early stopping functionality, while the average number of iterations is presented for designs employing early stopping. However, some papers proposing early stopping designs (eg. [60], [61], [64]) do not present an average number of iterations, only providing the maximum limit imposed, as indicated with an asterisk in Table II. Likewise, some papers (eg. [50], [53]) do not state the number of iterations employed, but this parameter can be inferred as a function of other parameters and characteristics. These cases are marked with a double-asterisk (**) in Table II.

B. Characteristics

In this section, we consider all those characteristics of FPGA-based LDPC decoders, which we plan to quantify. Seven main characteristics are identified, namely processing throughput, processing latency, hardware resource requirements, transmission energy efficiency, processing energy efficiency, bandwidth efficiency and flexibility, as seen Fig. 1. Each of these is described in turn in the following sections.

1) Processing throughput: Perhaps the most frequently-stated characteristic of an FPGA-based LDPC decoder is its processing throughput, which is the number of bits that it can

process per second. A high processing throughput is required for high-speed data transfers and video streaming applications, amongst other uses. A base station serving many users requires the sum of the individual throughputs to be high, so that each user receives a satisfactory service.

In an LDPC decoder it is important to note the difference between encoded and decoded processing throughput. We refer to the number of codeword bits processed per second as the encoded processing throughput, while we use decoded processing throughput to quantify the number of message word bits per second. For example, half of the codeword bits generated by a $1/2$-rate LDPC code are parity bits, which carry no information of their own. Therefore if the encoded processing throughput is 2 Gbps then the corresponding decoded processing throughput would be 1 Gbps. Ultimately, it is the decoded processing throughput that matters most to the user of the decoder, so we have deemed this to be the more important characteristic in comparisons. For designs where the author has only presented encoded processing throughput, we have inferred the decoded processing throughput by multiplying by the coding rate, as denoted by an asterisk in Table II. In some cases it is unclear whether the stated processing throughput is encoded or decoded. This is reflected in the Table II by allowing the stated processing throughput to span both columns. A double asterisk is used in Table II to identify designs in which the processing throughput was not explicitly stated, but has been inferred from other stated parameters and characteristics.

2) Processing latency: The processing latency of an FPGA-based LDPC decoder is the amount of time it requires to process a complete LDPC codeword. Low processing latency is therefore important for interactive cloud computing and safety-critical operations, where an immediate response is crucial. It may be observed that processing latency is strongly linked to processing throughput, since the processing latency can often be calculated as the message word length K divided by the decoded processing throughput. However, some decoder designs achieve a high processing throughput by decoding more than one codeword simultaneously. In these cases, the associated processing latency would be much higher than that of a decoder which achieves the same processing throughput while decoding only a single codeword at a time. For example, a decoder that decodes a single 1000-bit message word with a processing throughput of 2 Gbps would have a processing latency of 0.5 μs, while two 1 Gbps decoders operating in parallel would achieve the same processing throughput, but would have a processing latency of 1 μs. Processing latency is a key characteristic of an FPGA-based LDPC decoder, however most authors do not explicitly state it in their results, and it is therefore not included in Table II.

3) Hardware requirements: When implemented on an FPGA, the size and complexity of an LDPC decoder's design is represented by how much of the FPGA's hardware resources it utilises. Larger designs require more resources and therefore a bigger, more expensive FPGA, making smaller designs preferable.

The ELB metric described in Section II-F can be used to compare the hardware resource requirements of designs implemented on different FPGAs. However, the resource requirements stated by the various authors of LDPC-based FPGA decoder designs often do not directly translate to ELBs, hence requiring further analysis to be performed as follows:

- The conversion from 6LUTs to 4LUTs described in Section II-F is first employed to ensure that all measurements of LUTs consider an approximately equivalent quantity of hardware.
- Subsequently, if the hardware requirement of a design is quantified only in terms of either 4LUTs or FFs, then we assume a numerically equal number of ELBs.
- If the hardware requirement of a design is quantified in terms of both 4LUTs and FFs, then we assume that ELBs = $\max$(4LUTs, FFs). These cases are identified using a single asterisk in Table II.
- For designs based on Xilinx FPGAs having complex multi-element slices, we have derived a "utilisation" figure of merit, which quantifies how many LUTs/FFs are commonly used per slice. We obtained this by calculating the average utilisation of designs for which both the number of slices and the number of LUTs/FFs used is stated. These utilisation figures were found to be approximately 0.83 for LUTs and 0.36 for FFs, demonstrating that the majority of slices are used for their LUTs. For designs where the hardware utilisation is presented only in terms of slices, we assume ELBs = slices $\times$ 4LUTs per slice $\times$ 0.83. These cases are indicated in Table II using a double asterisk.

4) Transmission energy efficiency: Another fundamental figure of merit for an LDPC decoder is its error correction capability, as a function of the channel's signal to noise power ratio per bit E_b/N_0, which is typically expressed in decibels. If a codeword is transmitted using a high energy per bit E_b, then the energy of the noise corrupting each bit becomes relatively smaller, causing the BER at the receiver to decrease. However, energy-efficient transmitters are desirable, because they are cheaper to run and can operate for longer without requiring new batteries, particularly since transmission energy consumption is dominant in transmitter hardware. It is therefore desirable for an LDPC decoder to be capable of correcting errors and achieving a satisfactorily low BER, even at low E_b/N_0 values.

The error correction performance of a decoder is typically characterised in the form of a BER curve, showing how the BER is reduced as the channel E_b/N_0 increases. In order to convert these plots into a comparable metric, we specified a desirable target BER of 10^{-4}. For each considered design, the E_b/N_0 required by the decoder in order to achieve this BER was noted. In some publications however (eg. [17], [61]), the error correction performance is quantified using the Frame Error Rate (FER) rather than BER. In these cases, we assumed that a BER of 10^{-4} equates to a FER of 10^{-2} [61], based on the observation that the considered designs typically have a message word length K of the order of 1000 bits, as well as a minimum Hamming distance of the order of 10 bits. These cases are indicated using a single asterisk in the E_b/N_0 column of Table II.

Quantifying the BER versus E_b/N_0 facilitates a fair comparison of transmission energy for LDPC codes having different coding rates R, since it considers the transmission energy per message word bit. However, some publications present the BER as a function of the SNR E_s/N_0, which does not allow a fair comparison of codes having different coding rates, since it considers the transmission energy per codeword bit, $E_s = E_b \times R$. The corresponding E_b/N_0 can therefore be obtained by dividing by the SNR E_s/N_0 by the coding rate R, which is achieved in logarithmic terms according to

$$E_b/N_{0\ [\mathrm{dB}]} = \mathrm{SNR}_{\ [\mathrm{dB}]} - 10\log_{10}(R). \tag{18}$$

Entries calculated in this way are denoted in Table II using a double asterisk. Unfortunately some authors have erroneously labelled the x-axis of BER plots as SNR, when E_b/N_0 would be more appropriate. Some of these cases were clarified via private correspondence with the authors. However, in some cases there is other evidence that the presented results are in terms of E_b/N_0 rather than SNR, such as comparisons with benchmarkers or capacity bounds. In these cases, E_b/N_0 is assumed and identified using a triple asterisk (***) in Table II.

5) Processing energy efficiency: As for any electronic system, low processing energy consumption is desirable in the design of FPGA-based LDPC decoders. However, only a few publications ([28], [56], [73]) have included energy consumption measurements, hence this characteristic cannot be considered in our comparisons.

6) Bandwidth efficiency: The bandwidth efficiency of a communication system is given by the ratio of the information throughput that it can convey to the corresponding bandwidth required. For example, a scheme that conveys 500 bits per second over a channel having a bandwidth of 1 kHz has a bandwidth efficiency of 0.5 (bits/s)/Hz. For BPSK-modulated codewords using ideal Nyquist pulse shaping filters, bandwidth efficiency is numerically equal to the LDPC coding rate R. In this regard, LDPC codes with higher coding rates are more desirable, since they make more efficient use of their channel's bandwidth.

7) Flexibility: Flexibility is a desirable characteristic, because it allows an FPGA-based LDPC decoder to support different parity check matrices, having different coding rates, block lengths and node degrees. Some designs may support a selection of related PCMs from within a particular code family, such as the 21 PCMs included in the DVB-S2 standard [5]. Meanwhile, other designs may be completely flexible, supporting any PCM.

Decoders may exhibit flexibility either during their design or during their operation. True run-time flexibility allows a specific codeword to be decoded using a particular PCM, immediately before decoding a different codeword using a different PCM. This allows the communication system to dynamically adapt to time-varying channel conditions, such as by decreasing the coding rate R in high-noise environments in order to improve the BER performance. However this advantage may only be achieved at the cost of requiring a more sophisticated design, typically having higher hardware resource requirements or lower processing throughput. By contrast, decoders that are only flexible at design-time may only be adapted to use a different PCM by reprogramming the FPGA, preventing a high degree of rapid reconfigurability. The degree of design-time flexibility can also be difficult to accurately quantify, since any design that is synthesised from a HDL can be modified and re-synthesised fairly rapidly. Design-time flexibility has therefore not been considered in this survey.

IV. Discussions

The data presented in Section III inspires a great deal of discussions and visualisation of the relationships amongst the various parameters and characteristics of FPGA-based LDPC decoders. This section commences by characterising the fundamental trade-off between desirable characteristics in Section IV-A, before identifying the parameters that affect each characteristic in Section IV-B.

A. Trade-offs

As seen in Fig. 1 and discussed in Section III-B, the main measurable characteristics of an FPGA-based LDPC decoder are processing throughput, processing latency, hardware resource utilisation, transmission energy efficiency, processing energy efficiency, bandwidth efficiency and flexibility. Of these, it is the processing throughput, hardware resource utilisation, flexibility and transmission energy efficiency, which provide the clearest and most fundamental trade-off, since the other characteristics are all in some way dependent on these. The relationship amongst these four characteristics is plotted in Fig. 9.

Note that all scatter plots presented in this paper are organised so that a decoder with desirable values for all characteristics would correspond to a data point in the top-right corner. In Fig. 9, the x-axis is plotted with the values reversed, so that decoders with smaller hardware resource requirements (preferred) are further to the right than larger ones. Meanwhile the y-axis is plotted as normal, so that decoders with the highest processing throughput are at the top. In this way, points above the trend line are superior to the average case, whilst points below it are inferior, notwithstanding the values of their other characteristics.

It can be seen in Fig. 9 that most designs can only excel in at most three of the four characteristics presented. The trend line presents the average processing throughput vs size trade-off, and decoders that perform above this line generally tend to suffer from poor transmission energy efficiency, whilst decoders with a high energy efficiency tend to either have larger hardware resource requirements or lower processing throughput than the average case. Any decoders that perform well in all three of these characteristics tend to be totally inflexible to any PCM changes at run-time.

The five points in Fig. 9 having the highest processing throughput are from [28], [74], [68] and [67], all of which employ fully-parallel architectures. The design of [67] has the smallest hardware resource requirement of the four, owing to its use of only one bit per LLR. By contrast, the designs of [28], [68] and [74] use two or three bits per LLR, which is reflected in their relative hardware resource requirements.

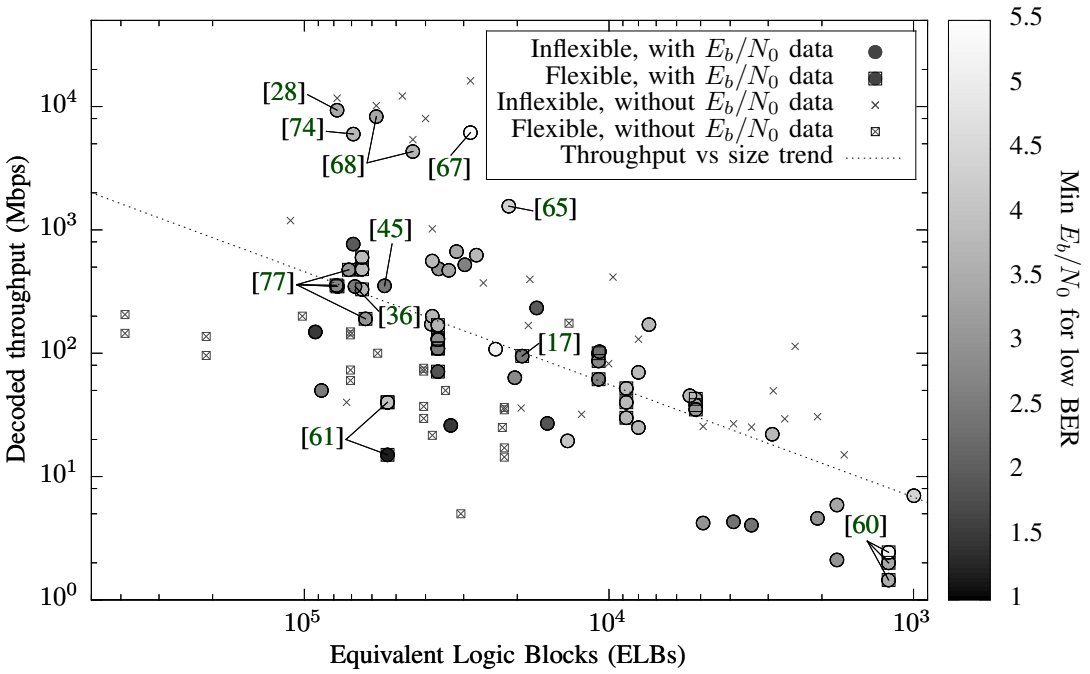


Fig. 9. Processing throughput vs. hardware requirements vs. transmission energy efficiency vs. flexibility

None of these high-throughput decoders have any run-time flexibility, as is typical of fully-parallel architectures. The next highest processing throughput is achieved by the design of [65], which adopts a partially-parallel rather than fully-parallel architecture, but also uses only one bit per LLR. The effect of using these small numbers of bits can be seen in these decoders' poor transmission energy efficiency, since reducing the resolution of the LLRs impedes the associated error correction capability.

In addition to employing single-bit LLRs, the design of [65] achieves a high processing throughput by decoding two frames at once. The designs presented in [77] use a similar technique, processing three, four or even six frames in parallel using multiple decoder copies in the same FPGA. Owing to this, these designs have a larger processing throughput than the average case, while also being flexible and having reasonable error correction performance. However, as discussed in Section III-B2, the processing latency of these decoders is much higher than their processing throughput would imply, making them less suitable for time-critical applications.

The decoders presented in [36] and [45] both achieve good transmission energy efficiency, while also having higher processing throughputs (or lower hardware requirements) than the average case. Both of these designs use stochastic bitstreams to represent the LLRs, facilitating a fully-parallel architecture having single-wire serial transmission between nodes, greatly simplifying the hardware design.

The points in the bottom-right of Fig. 9 correspond to the designs presented in [60], which employ a fully-serial architecture and so have very low hardware resource requirements, but also low processing throughput. However, these designs also have the benefit of being truly run-time flexible for any LDPC code. By contrast, the other flexible designs shown in Fig. 9, such as [17] and [61], are only flexible for a set of related PCMs.

In addition to the trade-offs described above, Fig. 9 also demonstrates that it is difficult to consider all of the characteristics of an FPGA-based LDPC decoder at once. For example, Fig. 9 does not consider the capabilities of the FPGA that each decoder is implemented using. In particular, more recent FPGAs may be able to operate identical designs at higher clock speeds than older FPGAs. This could be crudely factored into the results by dividing the processing throughput by the clock frequency, but doing so would then negate the impact of other parameters such as the critical path length. Furthermore, no consideration is given in Fig. 9 to the processing latency of each considered design. Note, however, that by plotting the decoded processing throughput rather than the encoded processing throughput, the coding rate and the bandwidth efficiency of the LDPC code has been taken into at least partial consideration.

B. Relationships between parameters and each characteristic

Having established the fundamental trade-off that exists between the main characteristics of FPGA-based LDPC decoders, namely processing throughput, processing latency, hardware requirements and transmission energy efficiency, the following subsections present discussions of the parameters that affect each one. A discussion of bandwidth efficiency is combined with transmission energy efficiency in Section IV-B4, but a quantitative discussion of flexibility and processing energy efficiency could not be made, owing to the lack of the required information in the publications considered.

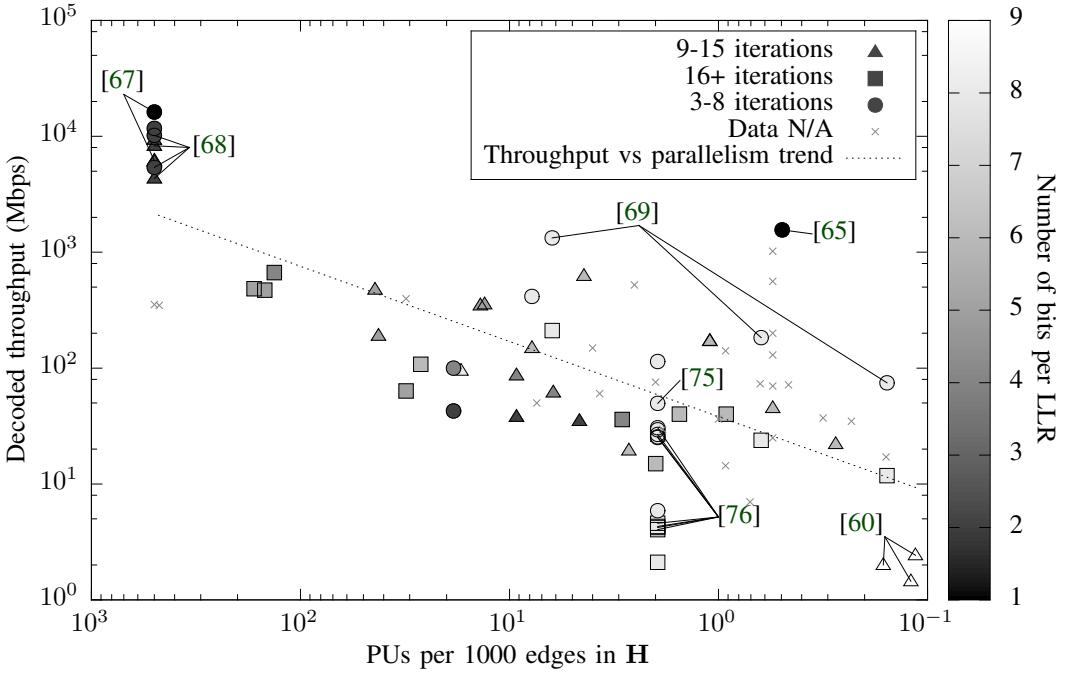


Fig. 10. Factors affecting the processing throughput

1) Processing throughput: Fig. 10 characterises the strong relationship between an FPGA-based LDPC decoder's degree of parallelism and its decoded processing throughput, confirming the expectation that designs having more parallel processors can decode a higher number of bits per second. Note that in Fig. 10 the number of parallel processing units has been divided by the number of edges in the PCM $\mathbf{H}$, to remove the dependence on the LDPC code size. The shading and shapes of the markers in Fig. 10 also indicate the influence that the number of bits per LLR and the number of decoding iterations have on the processing throughput, respectively. Points above the trend line typically employ a small number of bits per LLR or iterations, evidenced by their dark shading or circular point shape. By contrast, slower-than-average designs typically employ a larger number of bits per LLR or iterations, and therefore have lighter shading or a square shape.

Perhaps the most prominent points in Fig. 10 are the light grey circles belonging to [69], which achieve a much higher processing throughput than the trend line, despite using 8 bits per LLR. This may be explained by this design's use of layered belief propagation with the aid of a novel joint row-column processor, which decreases the processing time of each iteration and helps to avoid memory conflicts, thereby increasing the processing throughput.

The light triangles in the bottom-right represent the fully-serial decoders presented in [60], which achieve a low processing throughput owing to their low number of processors. Conversely, the dark points in the top-left represent the fully parallel decoders of [67] and [68], which achieve a very high processing throughput by using few bits, few iterations, a large degree of parallelism and operate on the basis of the MSA [25]. The fact that the MSA can facilitate a higher processing throughput than more complicated alternatives such as the SPA [24] is also demonstrated by comparing the results of [75] and [76], which present two very similar designs that vary in algorithm. The design in [76] suffers from a 4-5 Mbps processing throughput drop compared to [75], caused by its employment of the SPA instead of the MSA, as well as by using a non-uniform quantisation scheme for the LLRs.

The point furthest above the trend line corresponds to the design of [65], which achieves a high processing throughput by using only a single bit per LLR, five iterations per frame and by decoding two frames simultaneously. This design also exploits the properties of quasi-cyclic LDPC codes to implement an efficient partially-parallel architecture, reducing the number of processing units required to achieve its high processing throughput.

2) Processing latency: As discussed above, processing latency is not treated as a quantifiable characteristic in our analysis, because the majority of publications do not quantify this characteristic of their design. However, the processing latency is dependent on the processing throughput, the message word length K, the scheduling and the number of frames that are decoded in parallel.

Some of the decoders considered, such as that of [65] and [77], process multiple frames in parallel by instantiating several independent copies of the decoder on the same FPGA. In these cases the total processing throughput and resource requirement could be divided by the number of decoders, in order to produce results that correspond to the processing latency of an equivalent design that only considers one frame at a time. However, other designs, such as [42], process multiple frames by making use of spare time within the decoding schedule, with the result that the hardware cost does

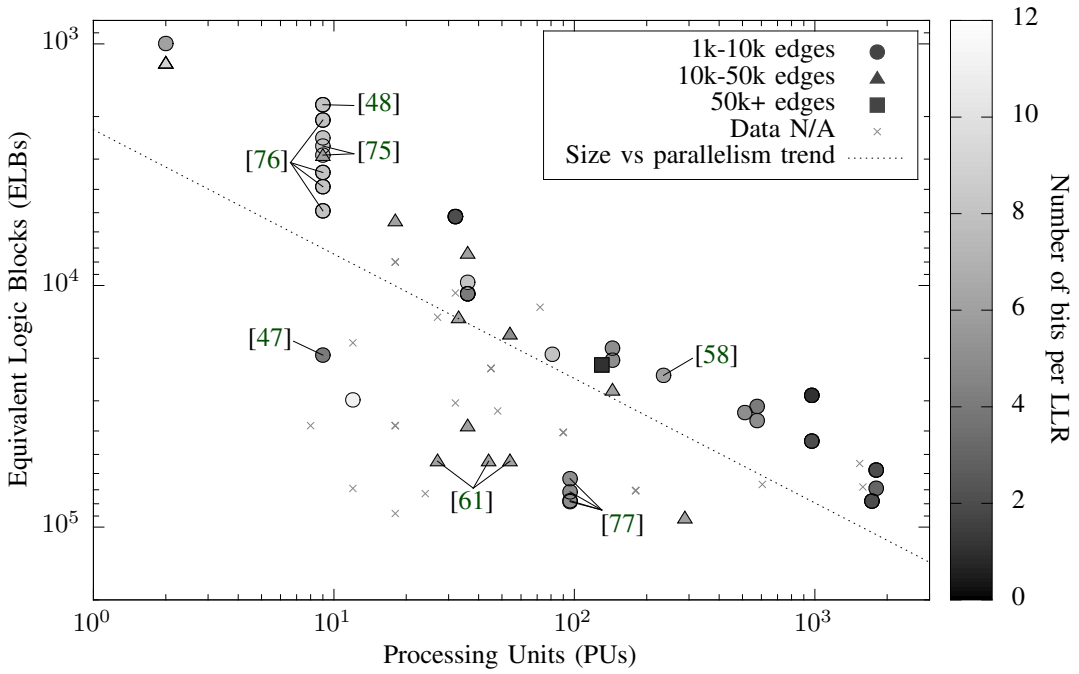


Fig. 11. Factors affecting the hardware requirements

not increase linearly with the processing throughput. Owing to this, it is not possible to normalise the data to only consider the processing throughput and hardware resources required for decoding one frame at a time, so the processing latency cannot be fairly inferred.

3) Hardware requirements: Unsurprisingly, the major contributing factor to the hardware resource requirement of an FPGA-based LDPC decoder design is its degree of parallelisation, as shown in Fig. 11. Additionally, Fig. 11 shows that the number of bits employed per LLR and the number of edges employed in the parity check matrix also have some influence on the hardware resource requirement, though the effects of these parameters are quite varied. This may be attributed to the difficulty of accurately comparing the hardware resource requirements of different designs, as well as suggesting that other factors are involved. It is however noticeable that there is a general reduction in the number of bits per LLR employed in designs with increased parallelism. This may be explained by the explosion in routing complexity upon increasing the number of PUs, which would be exacerbated by the requirement for data buses having large operand widths.

The dark grey circles corresponding to the designs of [77] towards the bottom of Fig. 11 seemingly have a much larger hardware resource requirement than would be expected, considering the number of processing units, the number of PCM edges and the number of bits employed per LLR. However, these designs are each run-time flexible for a different family of codes, having HDL code that is automatically generated. This additional flexibility results in decoders that are not as fully optimised as one that was designed specifically for a single PCM, explaining the associated hardware overhead. This is confirmed by the observation that the run-time flexible designs of [61] also correspond to a set of points positioned very far below the trend line.

The results of [75], [48] and [76] all sit above the trend line, despite employing a large number of bits per LLR, as well as a moderate PCM size. This may be partially attributed to their implementation of quasi-cyclic LDPC codes, using partially-parallel architectures, leading to a very efficient use of hardware resources. Additionally, the smallest hardware resource requirement of these designs is achieved by one that uses the MSA rather than the SPA, illustrating that this algorithm requires fewer hardware resources.

The design of [47] requires more FPGA resources than the trend line would suggest, which is remarkable considering its small PCM and number of bits per LLR. At first glance this may be attributed to its use of the uncommon array-based LDPC code. However, the design of [58] also uses an array-based code but sits above the trend line, despite employing a large number of bits per LLR and a large PCM. On closer inspection, it can be observed that the design of [47] employs a simple FPGA from an old generation, suggesting that its comparably large hardware resource requirement stems from inefficient FPGA synthesis.

4) Transmission energy efficiency and bandwidth efficiency: The minimum SNR per bit E_b/N_0 at which it becomes theoretically possible to reliably send information over a channel depends on the target bandwidth efficiency and therefore on the coding rate of the FEC code employed. A code having a lower coding rate may achieve a lower minimum transmission energy, owing to the increased number of parity bits that it employs for error correction. For this reason, we consider the transmission energy efficiency and the bandwidth efficiency jointly in this subsection.

For each FPGA-based LDPC decoder considered, the theoretical Discrete-input Continuous-output Memoryless Channel (DCMC) capacity [90] was calculated, with consideration of the coding rate, modulation type and channel model employed. This was then subtracted from the value recorded in Table II for the specific E_b/N_0 at which a low BER is achieved, in order to quantify the performance loss imposed by implementation factors. Here, a low performance loss is achieved by a decoder that can function very close to the theoretical limit, demonstrating that it is very good at correcting errors and very efficient in terms of transmission power and bandwidth.

Almost all of the publications considered characterised the error correction performance of their FPGA-based LDPC decoder designs using BPSK modulation for transmission over an AWGN channel. This allows the BER performance of these designs to be presented together graphically, as shown in Fig. 12. Here, the plotted line represents the DCMC capacity, while each plotted point corresponds to a different considered decoder design. The performance loss associated with each point may be obtained as its horizontal distance from the DCMC capacity curve. It can hence be seen that despite requiring drastically different E_b/N_0 levels to achieve the same BER, the designs of [41], [52], [60], [61] can all be considered to offer a strong error correction performance, when their bandwidth efficiency is also taken into consideration. Fig. 12 also illustrates that the designs of [42], [56], [67], [68], [74], [80] are comparatively poor at correcting errors, and therefore have a low transmission energy efficiency. This is at least partly due to the fact that these designs trade off their error correction performance against other desirable characteristics, as will be explained below. Note that this analysis can be readily extended to LDPC decoders designed for other modulation schemes or channel models. This may be achieved by plotting the corresponding DCMC capacity curve and characterising the error correction performance with respect to this bound.

It is well-known that LDPC codes having longer message word lengths K are capable of performing closer to the DCMC capacity [8]. Furthermore, a higher performance loss occurs for more sparse PCMs, since these have fewer edges over which to transfer information during the decoding process. Motivated by this, Fig. 13 plots the performance loss of each design versus the number of edges in its PCM $\mathbf{H}$, combining the message word length K with the complexity of the factor graph. As shown in Fig. 13, the number of edges in the PCM $\mathbf{H}$ is the largest contributing factor to the error correction performance loss. As may be expected, the performance loss is also influenced by the number of iterations performed and the number of bits used per LLR, as shown in Fig. 13. It may be observed that designs like those of [67], [65] and [74] perform poorly compared to the trend line, owing to their employment of a small number of bits per LLR or iterations. By contrast, a good performance may be observed for designs employing a large number of both, such as [69].

The specific code construction principles used can explain some of the unexpected results seen in Fig. 13. The design of [50] performs closer to the DCMC capacity than would be expected from a general decoder using the same small number

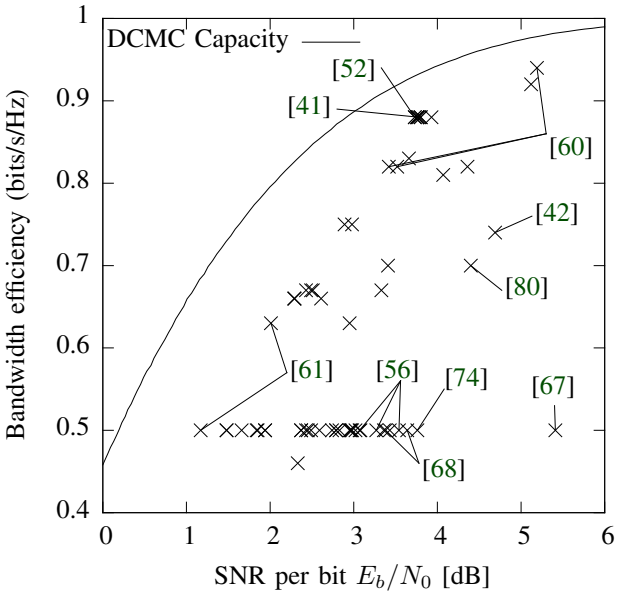


Fig. 12. Decoder performance loss from capacity

of iterations and bits per LLR. However, the PEG algorithm explained in Section II-D5 was used to construct the LDPC code it uses, increasing its error correction capability at the cost of producing an unstructured factor graph, which is not optimised for hardware implementation. A "cycle elimination algorithm" was used in [61] to similar effect, while the designs of [60] achieve high performance due to the completely unstructured $\mathbf{H}$ matrix used.

The lowest error correction performance loss is achieved by the design of [41], which uses a structured quasi-cyclic code. In addition, this design also uses a sophisticated non-uniform quantisation scheme for the representation of LLRs, it employs a moderate number of bits per LLR and iterations, as well as implementing the full SPA. By contrast, the designs of [76] and [48] operate further away from capacity than may be expected, which is due to their use of the MSA.

V. Recommendations and Further Work

This section presents an overview of the future development effort required of designers of FPGA-based LDPC decoders. Firstly, Section V-A provides a guide to the stages involved in designing an FPGA-based LDPC decoder. Following this, Section V-B then provides a set of recommendations for future publications which will facilitate more comprehensive comparisons amongst FPGA-based LDPC decoders in the future. Finally, Section V-C presents a list of future research opportunities that we expect to be of significant benefit to the field.

A. Recommended design methodology

As discussed above, the complex relationships between the parameters and characteristics of FPGA-based LDPC decoders imply that it is not possible to identify a single design which is superior to all others in every way. Having said this, the

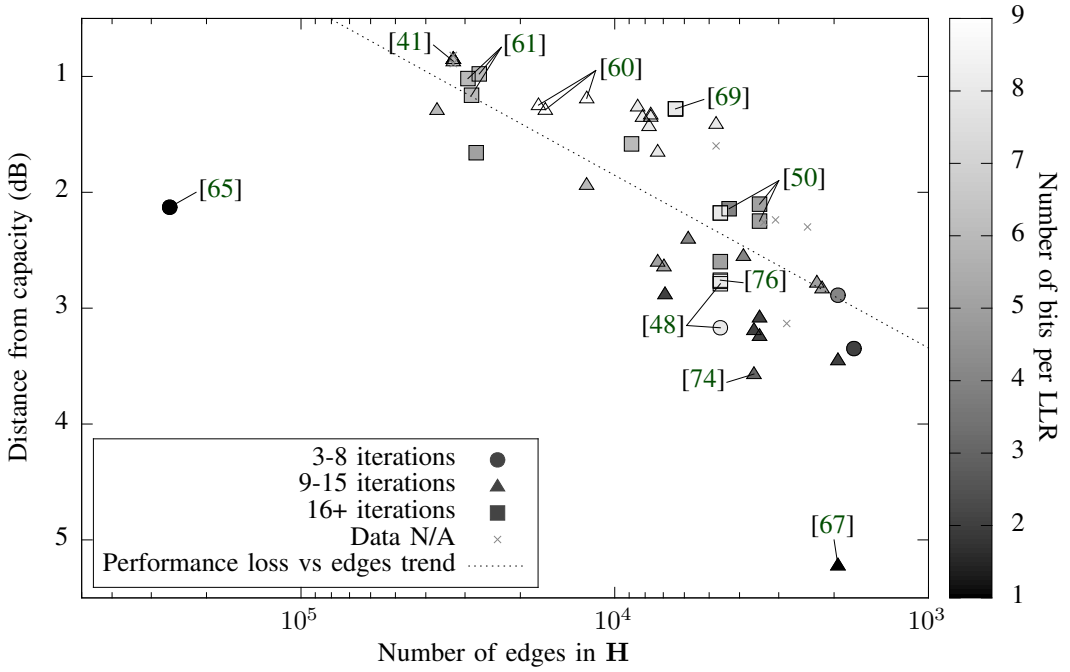


Fig. 13. Factors affecting the error correction performance

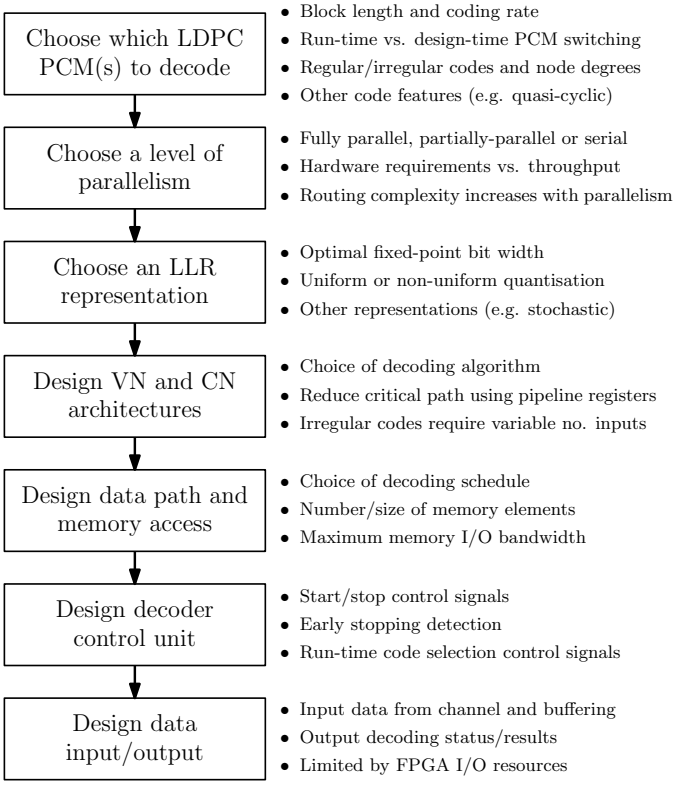


Fig. 14. Stages to consider when designing an FPGA-based LDPC decoder

flowchart presented in Fig. 14 outlines a recommended series of stages for a prospective designer to complete, as a means of assisting their design process. The bullet points accompanying each stage list some of the key issues to be considered whilst completing each design element. More details about these issues can be found throughout Sections II-C — II-E.

B. Recommendations

In the process of collecting the data presented in this paper, it has become apparent that fairer comparisons amongst FPGA-based LDPC decoders could be facilitated in the future, by setting conventions for the type and format of data to present when proposing a new design. The following list represents our attempt at this. Our recommendation for future publications of FPGA-based LDPC decoders is to:

- provide values for every parameter and characteristic presented in Table II;
- ensure that all presented characteristics correspond to the same set of parameters, and if more than one parameter set is employed for demonstrating the flexibility of the design, include an equal number of full characteristic sets;
- state both the encoded and the decoded processing throughput, as well as the formula used for calculating them;
- state the processing latency of the decoder, or signify that it can be derived simply from the processing throughput and message word length K;
- provide BER simulation curves obtained using the physical hardware, plotting the results against E_b/N_0 [dB] and explicitly stating the channel model and modulation type used, preferably BPSK modulation for transmission over an AWGN channel;
- if possible, provide multiple BER plots for different maximum numbers of iterations;

- provide mathematical detail about the algorithm used and endeavour to use established terminology if the same formulae have been used before;
- provide power/energy consumption measurements obtained during BER simulation;
- when mentioning flexibility, explicitly state whether the changes can be made at run-time or whether they require a new synthesis run;
- endeavour to make it possible to compare new designs to old ones by selecting a benchmarker, and implementing a new design using exactly the same set of parameters on the same FPGA.

In addition to adhering to the above list of guidelines to facilitate fairer comparisons between different designs, it would be of significant benefit if authors of FPGA-based LDPC decoder designs were at liberty to make their source code freely available online. Open-source code can be readily found for many of the signal processing blocks used in communications systems, but unfortunately there are very few freely-available FPGA-based LDPC decoder designs. This inevitably hinders innovation within the field, since every prospective designer is required to commence by implementing a basic structure, rather than improving an existing design. Additionally, if a reader of a published design had access to the source code, it would significantly aid their comprehension of the novel techniques that are being described. Finally, making source code freely available facilitates the employment of current FPGA-based LDPC decoder designs as benchmarkers for future designs.

C. Further work

Performing the analysis described above has enabled us to identify several opportunities for further research and development in the field of FPGA-based LDPC decoders, as discussed in the following subsections.

1) Flexible decoders: Perhaps the biggest gap illustrated by the trade-offs described in Section IV-A is for high-speed decoders having run-time flexibility and low hardware resource cost. Run-time flexibility has huge advantages for commercial applications, since it allows a decoder to dynamically support the variety of different LDPC codes within a particular communications standard, without incurring the overhead of the time and technical intervention that is required to reprogram an FPGA. Further to this, flexible decoders can adapt automatically depending on the channel conditions without any user input, increasing the efficiency of FPGA-based LDPC decoders in consumer applications. Run-time flexibility can also be useful for research purposes, reducing the number of times an FPGA has to be re-synthesised when testing multiple different codes.

As seen in Section IV-A, decoders having a fully-serial architecture can be flexible with little or no extra hardware resource cost, but suffer in terms of their low processing throughput. Meanwhile, the extra hardware required to make a fully-parallel decoder flexible renders this approach impractical, regardless of their capacity for high processing throughputs. Initial research therefore suggests that partially-parallel decoder architectures utilising semi-structured (e.g. quasi-cyclic) LDPC codes such as those in [77] have the greatest potential for flexibility and high processing throughputs. Recent research into hierarchical quasi-cyclic codes as in [56] and [53] could be of particular interest.

2) Schedules: Unfortunately, different publications have used different terminology to describe the decoding schedules adopted in their decoder designs, so a direct comparison could not be easily drawn between them in this paper. However, there is an opportunity to investigate the effects of using different schedules in two otherwise equal FPGA-based LDPC decoders, assessing their effects not only on BER performance and complexity as in previous research on scheduling, but also on processing throughput, hardware resources, processing energy and flexibility. In particular, none of the reviewed decoders operate on the basis of calculating residuals as in IDS, implying that this schedule is largely under-represented within the field despite claims of its superiority to others [22]. Further research is required to determine whether these claims are valid in practical implementations, and to investigate the architectural constraints that employing IDS would impose on an FPGA-based LDPC decoder design.

3) Stochastic decoders: The two stochastic decoders presented in this report, [36] and [45], performed well in terms of processing throughput, BER performance and hardware requirements. Stochastic designs are associated with their own set of advantages and challenges, offering another opportunity for further research. The serial transmission of messages between processing nodes facilitates a higher grade of feasibility for fully-parallel designs, and allows the error correction performance to be dynamically traded for processing throughput by simply increasing the number of bits used for each message.

4) Low processing energy consumption: It is unfortunate that the majority of the designs reviewed in this report did not present any information about the decoder's energy consumption. As with all electronic devices, low energy consumption is a key figure of merit in communication systems, since it dictates how long mobile devices can function for between battery recharges, as well as dictating the cost and environmental impact of operating base station equipment. This provides a motivation to investigate the factors behind energy consumption in FPGA-based LDPC decoders, possibly by implementing some of the published designs and measuring their energy consumption directly. Drawing upon these results, FPGA-based LDPC decoders having low energy consumption could then be designed.

5) Low processing latency: Similarly to processing energy consumption, processing latency is a crucial characteristic of communications hardware that was curiously under-represented in the works reviewed here. While the processing latency may be approximated for many designs as a function of the processing throughput and message word length, the fact that it was rarely quantified implies that it was rarely a design focus. Some applications of FPGA-based LDPC decoders may require ultra-low processing latency above all other characteristics, suggesting that this is a gap in the market that is currently unfilled. Further research could be conducted to determine whether this is indeed the case, before devising new designs having ultra-low processing latency. Such designs

would require large processing throughputs without processing multiple frames in parallel, so would be likely to employ very parallel architectures and low-complexity algorithms. In this case, the cost of the high processing latency would be a higher hardware resource consumption and a lower transmission energy efficiency.

VI. Conclusions

In this paper, we have assessed the practicalities and limitations of FPGA-based LDPC decoders. Section II presented a tutorial on LDPC codes, covering their structure, encoding process, decoding process and construction techniques. A number of practical decoder implementation decisions were then highlighted, before providing background information on the structure of FPGAs and the differences between those produced by the main two FPGA vendors. In Section III, the results from an extensive survey were presented in a condensed form, featuring only a subset of the rows and columns available in the online version. The remainder of Section III was then devoted to describing the parameters and characteristics used in the evaluation, discussing the significance of each and how it was measured. Section IV then illustrated, characterised and discussed the complex interplay between all of these parameters and characteristics, using plots of the results to show how each one was affected by the others. Subsequently, using the experience gained from compiling the survey results, Section V presented a list of recommendations for future publications of FPGA-based LDPC decoder designs, in order to facilitate fairer, more comprehensive comparisons in future. Finally, we have identified a number of opportunities for future FPGA-based LDPC decoder designs.

Perhaps the most significant conclusion that can be drawn from the research described in this paper is that it is extremely difficult to predict how two different FPGA-based LDPC decoder designs might compare, when they are implemented using different codes, architectures, algorithms, schedules and hardware. This in itself lends further weight to the advantage of using FPGAs for prototyping designs, utilising their re-programmability in an efficient design-implement-test development cycle. To do so requires accurate comparisons amongst competing designs to be made, which can only be achieved using the list of recommendations provided in Section V-B. However, even having completed this process, it may still be difficult to say which design is superior, as there is such a complex interplay of characteristics that each will inevitably have its own advantages and disadvantages.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, no. Jan., pp. 21–28, 1962.

[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, p. 1645, Aug. 1996.

[3] IEEE, "IEEE 802.11n-2009 Standard for Information technology - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)," 2009.

[4] ——, "IEEE 802.16-2004 Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems," 2004.

[5] ETSI, "ETSI EN 302 307 v1.3.1 Digital Video Broadcasting (DVB); Second generation," 2013. [Online]. Available: https://www.dvb.org/standards/dvb-s2

[6] CCSDS, "CCSDS 131.0-B-2 Recommendation for Space Data System Standards; TM Synchronization and Channel Coding," 2011.

[7] V. Oksman and S. Galli, "G.hn: The new ITU-T home networking standard," *IEEE Commun. Mag.*, vol. 47, no. 10, pp. 138–145, Oct. 2009.

[8] G. D. Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, 2001.

[9] Y. Cai, S. Jeon, K. Mai, and B. V. K. V. Kumar, "Highly parallel FPGA emulation for LDPC error floor characterization in perpendicular magnetic recording channel," *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3761–3764, 2009.

[10] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed Stochastic Decoding of LDPC Codes," *IEEE Trans. Signal Process.*, vol. 59, no. 11, pp. 5617–5626, Nov. 2011.

[11] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3385–3400, 2014.

[12] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, "Stochastic Decoding of LDPC Codes over GF(q)," *IEEE Trans. Commun.*, vol. 61, no. 3, 2013.

[13] S. S. Tehrani, C. Jego, B. Zhu, and W. J. Gross, "Stochastic Decoding of Linear Block Codes With High-Density Parity-Check Matrices," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5733–5739, 2008.

[14] L. Zhang, L. Gui, Y. Xu, and W. Zhang, "Configurable Multi-Rate Decoder Architecture for QC-LDPC Codes Based Broadband Broadcasting System," *IEEE Trans. Broadcast.*, vol. 54, no. 2, pp. 226–235, 2008.

[15] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. Commun.*, vol. 57, no. 11, pp. 3258–3268, 2009.

[16] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "High throughput low-density parity-check decoder architectures," in *IEEE Glob. Telecommun. Conf.*, no. 3. San Antonio, TX, USA: IEEE, Nov. 2001, pp. 3019–3024.

[17] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable LDPC decoder architectures for regular and irregular codes," *J. Signal Process. Syst.*, vol. 53, no. 1-2, pp. 73–88, May 2008.

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

[19] L. Zhang, J. Huang, and L. L. Cheng, "Reliability-based high-efficient dynamic schedules for belief propagation decoding of LDPC codes," in *IEEE Int. Conf. Signal Process.*, no. 1. Beijing, China: IEEE, Oct. 2012, pp. 1388–1392.

[20] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for LDPC decoding," *IEEE Trans. Inform. Theory*, vol. 53, no. 11, pp. 4076–4091, Nov. 2007.

[21] Y.-M. Chang, A. I. V. Casado, M.-C. F. Chang, and R. D. Wesel, "Lower-complexity layered belief-propagation decoding of LDPC codes," in *IEEE Int. Conf. Commun.* Beijing, China: IEEE, May 2008, pp. 1155–1160.

[22] A. I. Vila Casado, M. Griot, and R. D. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *IEEE Int. Conf. Commun.* Glasgow, Scotland: IEEE, Jun. 2007, pp. 932–937.

[23] ——, "Improving LDPC decoders via informed dynamic scheduling," in *IEEE Inform. Theory Work.* Tahoe City, CA, USA: IEEE, Sep. 2007, pp. 208–213.

[24] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[25] F. Angarita, J. Valls, V. Almenar, and V. Torres, "Reduced-complexity min-sum algorithm for decoding LDPC codes with low error-floor," *IEEE Trans. Circuits Syst. I, Reg. Pap.*, vol. 61, no. 7, pp. 2150–2158, Jul. 2014.

[26] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Trans. Circuits Syst. I, Reg. Pap.*, vol. 51, no. 6, pp. 1106–1113, Jun. 2004.

[27] C. Spagnol, W. Marnane, and E. Popovici, "FPGA implementations of LDPC over GF(2m) decoders," in *IEEE Work. Signal Process. Syst.*, no. 8. Shanghai, China: IEEE, Oct. 2007, pp. 273–278.

[28] 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, Mar. 2012.

[29] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping set distribution of LDPC code ensembles," *IEEE Trans. Inform. Theory*, vol. 51, no. 3, pp. 929–953, Mar. 2005.

[30] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.

[31] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[32] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[33] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, "Near-Shannon-limit quasi-cyclic low-density parity-check codes," *IEEE Trans. Commun.*, vol. 52, no. 7, pp. 1038–1042, Jul. 2004.

[34] E. Eleftheriou and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[35] E. Yeo, B. Nikolic, and V. Anantharam, "Architectures and implementations of low-density parity check decoding algorithms," in *Midwest Symp. Circuits Syst.* Tulsa, OK, USA: IEEE, Aug. 2002, pp. 437–440.

[36] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, 2008.

[37] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible LDPC decoder," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 54, no. 6, pp. 542–546, Jun. 2007.

[38] T. Zhang, Z. Wang, and K. K. Parhi, "On finite precision implementation of low density parity check codes decoder," in *IEEE Int. Symp. Circuits Syst.* Sydney, Australia: IEEE, May 2001, pp. 202–205.

[39] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727–1737, Jan. 2001.

[40] X. Zuo, R. G. Maunder, and L. L. Hanzo, "Design of Fixed-Point Processing Based LDPC Codes Using EXIT Charts," in *IEEE Veh. Technol. Conf.*, San Francisco, CA, USA, Jan. 2011.

[41] Z. Cui and Z. Wang, "A 170 Mbps (8176, 7156) quasi-cyclic LDPC decoder implementation with FPGA," in *IEEE Int. Symp. Circuits Syst.*, no. x. Kos, Greece: IEEE, May 2006, pp. 5095–5098.

[42] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *IEEE Int. Symp. Circuits Syst.* Kos, Greece: IEEE, May 2006, pp. 1–4.

[43] Y. Sun, Y. Zhang, J. Hu, and Z. Zhang, "FPGA implementation of nonbinary quasi-cyclic LDPC decoder based on EMS algorithm," in *Int. Conf. Commun. Circuits Syst.* Milpitas, CA, USA: IEEE, Jul. 2009, pp. 1061–1065.

[44] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, 2007.

[45] S. S. Tehrani, S. Mannor, and W. J. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *IEEE Work. Signal Process. Syst.* Shanghai, China: IEEE, Oct. 2007, pp. 255–260.

[46] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. L. Hanzo, "Survey results for 'A survey of FPGA-based LDPC decoders'," 2015. [Online]. Available: http://dx.doi.org/10.5258/SOTON/384946

[47] P. Bhagawat, M. Uppal, and G. Choi, "FPGA based implementation of decoder for array low-density parity-check codes," in *IEEE Proc. Int. Conf. Acoust. Speech Signal Process.* Philadelphia, PA, USA: IEEE, Mar. 2005, pp. 29–32.

[48] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, and S. Goto, "Partially-parallel LDPC decoder based on high-efficiency message-passing algorithm," in *IEEE Proc. Int. Conf. Comput. Des.* San Jose, CA, USA: IEEE Comput. Soc, Oct. 2005, pp. 503–510.

[49] T. Zhang and K. K. Parhi, "A 54 Mbps (3,6)-regular FPGA LDPC decoder," in *IEEE Work. Signal Process. Syst.* San Diego, CA, USA: IEEE, Oct. 2002, pp. 127–132.

[50] K. Wang, N. Liu, B. Sun, and H. Sun, "A configurable FPGA implementation of PEG-based PS-LDPC decoder," in *Int. Conf. Pervasive Comput. Signal Process. Appl.* Harbin, China: IEEE, Sep. 2010, pp. 670–674.

[51] Y. Chen and D. E. Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder," in *IEEE Glob. Telecommun. Conf.* San Francisco, CA, USA: IEEE, Dec. 2003, pp. 113–117.

[52] F. Demangel, N. Fau, N. Drabik, F. Charot, and C. Wolinski, "A generic architecture of CCSDS low density parity check decoder for near-earth applications," in *Proc. Conf. Des. Autom. Test Eur.* Nice, France: European Design and Automation Association, Apr. 2009, pp. 1242–1245.

[53] V. A. Chandrasetty and S. M. Aziz, "A highly flexible LDPC decoder using hierarchical quasi-cyclic matrix with layered permutation," *J. Networks*, vol. 7, no. 3, pp. 441–450, Mar. 2012.

[54] P. Saunders and A. Fagan, "A high speed, low memory FPGA based LDPC decoder architecture for quasi-cyclic LDPC codes," in *Int. Conf. F. Program. Log. Appl.* Madrid, Spain: IEEE, Aug. 2006, pp. 1–6.

[55] Y.-H. Chien and M.-K. Ku, "A high throughput H-QC LDPC decoder," in *IEEE Int. Symp. Circuits Syst.* New Orleans, LA, USA: IEEE, May 2007, pp. 1649–1652.

[56] V. A. Chandrasetty and S. M. Aziz, "A multi-level hierarchical quasi-cyclic matrix for implementation of flexible partially-parallel LDPC decoders," in *IEEE Int. Conf. Multimed. Expo*. Barcelona, Spain: IEEE, Jul. 2011, pp. 1–7.

[57] F. Charot, C. Wolinski, N. Fau, and F. Hamon, "A new powerful scalable generic multi-standard LDPC decoder architecture," in *Int. Symp. Field-Programmable Cust. Comput. Mach.* Palo Alto, CA, USA: IEEE, Apr. 2008, pp. 314–315.

[58] J. Sha, M. Gao, Z. Zhang, L. Li, and Z. Wang, "An FPGA implementation of array LDPC decoder," in *IEEE Asia Pac. Conf. Circuits Syst.* Singapore: IEEE, Dec. 2006, pp. 1675–1678.

[59] Z. Cao, J. Kang, and P. Fan, "An FPGA implementation of a structured irregular LDPC decoder," in *IEEE Int. Symp. Microw. Antenna Propag. EMC Technol. Wirel. Commun.*, vol. 1. Beijing, China: IEEE, Aug. 2005, pp. 1050–1053.

[60] S. M. E. Hosseini, K. S. Chan, and W. L. Goh, "A reconfigurable FPGA implementation of an LDPC decoder for unstructured codes," in *Int. Conf. Signals Circuits Syst.* Nabeul, Tunisia: IEEE, Nov. 2008, pp. 1–6.

[61] L. Yang, H. Liu, and C. J. R. Shi, "Code construction and FPGA implementation of a low-error-floor multi-rate low-density parity-check code decoder," *IEEE Trans. Circuits Syst. I, Reg. Pap.*, vol. 53, no. 4, pp. 892–904, 2006.

[62] H. Ding, S. Yang, W. Luo, and M. Dong, "Design and implementation for high speed LDPC decoder with layered decoding," in *WRI Int. Conf. Commun. Mob. Comput.* Yunnan: IEEE, Jan. 2009, pp. 156–160.

[63] Y. Pei, L. Yin, and J. Lu, "Design of irregular LDPC codec on a single chip FPGA," in *IEEE Proc. Circuits Syst. Symp. Emerg. Technol.*, vol. 1. Shanghai, China: IEEE, May 2004, pp. 221–224.

[64] M. Gomes, G. Falcão, V. Silva, V. Ferreira, A. Sengo, and M. Falcão, "Flexible parallel architecture for DVB-S2 LDPC decoders," in *IEEE Glob. Telecommun. Conf.* Washington, DC, USA: IEEE, Nov. 2007, pp. 3265–3269.

[65] X. Chen, Q. Huang, S. Lin, and V. Akella, "FPGA based low-complexity high-throughput tri-mode decoder for quasi-cyclic LDPC codes," in *Annu. Allert. Conf. Commun. Control Comput.* Monticello, IL, USA: IEEE, Sep. 2009, pp. 600–606.

[66] C. Beuschel and H. Pfleiderer, "FPGA implementation of a flexible decoder for long LDPC codes," in *2008 Int. Conf. F. Program. Log. Appl.* Heidelberg, Germany: IEEE, Sep. 2008, pp. 185–190.

[67] V. A. Chandrasetty and S. M. Aziz, "FPGA Implementation of a LDPC Decoder using a Reduced Complexity Message Passing Algorithm," *J. Networks*, vol. 6, no. 1, pp. 36–45, Jan. 2011.

[68] ——, "FPGA implementation of high performance LDPC decoder using modified 2-bit min-sum algorithm," in *Int. Conf. Comput. Res. Dev.* Kuala Lumpur, Malaysia: IEEE, May 2010, pp. 881–885.

[69] Z. He, S. Roy, and P. Fortier, "FPGA implementation of LDPC decoders based on joint row-column decoding algorithm," in *IEEE Int. Symp. Circuits Syst.* New Orleans, LA, USA: IEEE, May 2007, pp. 1653–1656.

[70] A. Blad and O. Gustafsson, "FPGA implementation of rate-compatible QC-LDPC code decoder," in *Eur. Conf. Circ. Theory Des.* Linkoping, Sweden: IEEE, Aug. 2011, pp. 777–780.

[71] S. S. Khati, P. Bisht, and S. C. Pujari, "Improved decoder design for LDPC codes based on selective node processing," in *World Congr. Inform. Commun. Technol.* IEEE, Oct. 2012, pp. 413–418.

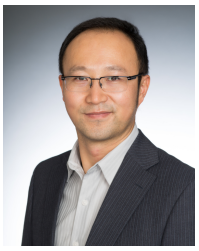
[72] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured low- density parity-check codes by hardware emulation," in *IEEE Glob. Telecommun. Conf.*, no. 2. San Francisco, CA, USA: IEEE, Nov. 2006, pp. 1–6.

[73] 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. Pap.*, vol. 58, no. 1, pp. 98–111, 2011.

[74] R. Zarubica, S. G. Wilson, and E. Hall, "Multi-Gbps FPGA-based low density parity check (LDPC) decoder design," in *IEEE Glob.*

Telecommun. Conf., no. 1. Washington, DC, USA: IEEE, Nov. 2007, pp. 548–552.
[75] Y. Dai, Z. Yan, and N. Chen, "Optimal overlapped message passing decoding of quasi-cyclic LDPC codes," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 5, pp. 565–578, 2008.
[76] N. Chen, Y. Dai, and Z. Yan, "Partly parallel overlapped sum-product decoder architectures for quasi-cyclic LDPC codes," in *IEEE Work. Signal Process. Syst.* Banff, AB, Canada: IEEE, Oct. 2006, pp. 220–225.
[77] H. Li, Y. S. Park, and Z. Zhang, "Reconfigurable architecture and automated design flow for rapid FPGA-based LDPC code emulation," in *Proc. ACM/SIGDA Int. Symp. F. Program. Gate Arrays*. Monterey, CA, USA: ACM, Feb. 2012, pp. 167–170.
[78] C. Spagnol, W. Marnane, and E. Popovici, "Reduced complexity, FPGA implementation of quasi-cyclic LDPC decoder," in *Proc. Eur. Conf. Circ. Theory Des.*, vol. 1. Cork, Ireland: IEEE, Aug. 2005, pp. 289–292.
[79] M. Karkooti and J. R. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," in *Proc. Int. Conf. Inform. Technol. Coding Comput.* Las Vegas, NV, USA: IEEE, Apr. 2004, pp. 579–585.
[80] L. Xiong, Z. Tan, and D. Yao, "The moderate-throughput and memory-efficient LDPC decoder," in *2006 8th Int. Conf. Signal Process.* Beijing, China: IEEE, Nov. 2006, pp. 1–4.
[81] Softjin Technologies, "LDPC decoder for DVB-S2." [Online]. Available: http://www.softjin.com/IP_Datasheet_PDF_version/LDPC_Decoder_datasheet.pdf
[82] Unicore Systems Ltd, "CCSDS C2 LDPC encoder/decoder IP cores," 2011. [Online]. Available: http://unicore.co.ua/uploads/File/CCSDS_XX_user_manual(netlist).pdf
[83] ——, "IEEE 802.16e (WiMAX) LDPC decoder IP core," 2009. [Online]. Available: http://unicore.co.ua/uploads/File/ldpc_dec_brief.pdf
[84] Blue Rum Consulting Limited, "802.11n/802.11ac LDPC decoder," 2013. [Online]. Available: http://www.bluerum.co.uk/consulting/datasheets/BRC008_LdpcDecRtlDs.pdf
[85] Turbo Concept, "ITU G.hn LDPC decoder." [Online]. Available: http://www.turboconcept.com/prod_tc4400.php
[86] Creonic GmbH, "IEEE 802.11ad WiGig LDPC decoder product brief," 2014. [Online]. Available: http://www.creonic.com/images/product_briefs/PB_Creonic_IEEE_802_11ad_WiGig_LDPC_Decoder_IP.pdf
[87] IPrium Ltd., "I.6 LDPC encoder/decoder IP core short description," 2013. [Online]. Available: https://www.iprium.com/bins/pdf/iprium_ug_i6_ldpc_codec.pdf
[88] TrellisWare Technologies, "Flexible low-density parity-check (F-LDPC)," 2014. [Online]. Available: http://www.trellisware.com/products/fec-products/f-ldpc/
[89] Logic Fruit Technologies, "LDPC decoder IP specification," 2010. [Online]. Available: http://www.logic-fruit.com/resource/LDPCDecoderIP.pdf
[90] L. L. Hanzo, S. X. Ng, T. Keller, and W. Webb, *Quadrature Amplitude Modulation*. Chichester: Wiley-IEEE Press, 2004.

Peter Hailes studied Electronic Engineering with Mobile and Secure Systems with the department of Electronics and Computer Science at the University of Southampton, and graduated with a first-class masters degree in 2013. He then stayed on to undertake research towards a Ph.D. in advanced hardware implementations of LDPC decoders. His other research interests include field-programmable gate arrays, error correction coding, embedded hardware/software design and high-level software development.



Lei Xu has been working with Altera for more than 7 years within system solution engineering and marketing. His current role is wireless system architect, instrumental to define and drive strategic direction and solution roadmap of wireless business unit of Altera. Previously he has been working on various wireless system solutions in Altera such as DPD, MIMO, Turbo SIC, etc. Prior to that, he has been working as the system algorithmic/architecture expert in VIA technology and Agilent technology on various wireless and broadcasting systems, such as DAB, DVB, GSM/WCDMA, WiFI and LTE. He holds BSEE and MSEE from Tsinghua University, China and PhD of Wireless Communication from University of Southampton, UK and has published 20+ leading journal and conference papers and holds 11 patents.

Robert G. Maunder (S03-M08-SM12) has been with the department of Electronics and Computer Science at the University of Southampton, UK, since October 2000. He was awarded the B.Eng. (Hons.) degree in electronic engineering in 2003, as well as a Ph.D. degree in wireless communications in 2007. He became a lecturer in 2007 and an Associated Professor in 2013. His research interests include joint source/channel coding, iterative decoding, irregular coding, and modulation techniques.

Bashir M. Al-Hashimi (M99-SM01-F09) is a Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering at University of Southampton, UK. He is ARM Professor of Computer Engineering and Co-Director of the ARM-ECS research centre. His research interests include methods, algorithms and design automation tools for energy efficient of embedded computing systems. He has published over 300 technical papers, authored or co-authored 5 books and has graduated 31 PhD students.

Lajos Hanzo (http://www-mobile.ecs.soton.ac.uk) FREng, FIEEE, FIET, Fellow of EURASIP, DSc received his degree in electronics in 1976 and his doctorate in 1983. In 2009 he was awarded an honorary doctorate by the Technical University of Budapest, while in 2015 by the University of Edinburgh. During his 38-year career in telecommunications he has held various research and academic posts in Hungary, Germany and the UK. Since 1986 he has been with the School of Electronics and Computer Science, University of Southampton, UK, where he holds the chair in telecommunications. He has successfully supervised about 100 PhD students, co-authored 20 John Wiley/IEEE Press books on mobile radio communications totalling in excess of 10 000 pages, published 1500+ research entries at IEEE Xplore, acted both as TPC and General Chair of IEEE conferences, presented keynote lectures and has been awarded a number of distinctions. Currently he is directing a 60-strong academic research team, working on a range of research projects in the field of wireless multimedia communications sponsored by industry, the Engineering and Physical Sciences Research Council (EPSRC) UK, the European Research Council's Advanced Fellow Grant and the Royal Society's Wolfson Research Merit Award. He is an enthusiastic supporter of industrial and academic liaison and he offers a range of industrial courses. He is also a Governor of the IEEE VTS. During 2008 - 2012 he was the Editor-in-Chief of the IEEE Press and a Chaired Professor also at Tsinghua University, Beijing. His research is funded by the European Research Council's Senior Research Fellow Grant. For further information on research in progress and associated publications please refer to http://www-mobile.ecs.soton.ac.uk Lajos has 22 000+ citations.