A Flexible FPGA-Based Quasi-Cyclic LDPC Decoder

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 diverse communications systems, owing to their strong error correction performance and their suitability to parallel hardware implementation. A great deal of research effort has been invested into the implementation of LDPC decoder designs on Field-Programmable Gate Array (FPGA) devices, in order to exploit their high processing speed, parallelism and re-programmability. Meanwhile, a variety of Application-Specific Integrated Circuit (ASIC) implementations of multi-mode LDPC decoders exhibiting both inter-standard and intra-standard reconfiguration flexibility are available in the open literature. However, the high complexity of the adaptable routing and processing elements that are required by a flexible LDPC decoder has resulted in a lack of viable FPGA-based implementations. Hence in this work, we propose a parameterisable FPGA-based LDPC decoder architecture, which supports run-time flexibility over any set of one or more Quasi-Cyclic (QC) LDPC codes. Additionally, we propose an offline design flow, which may be used to automatically generate an optimised HDL description of our decoder, having support for a chosen selection of codes. Our implementation results show that the proposed architecture achieves a high level of design-time and run-time flexibility, whilst maintaining a reasonable processing throughput, hardware resource requirement and error correction performance.**

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

GLOSSARY

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
BRAM	Block RAM
BS	Barrel Shifter
CMEM	CND Memory
CN	Check Node
CND	Check Node Decoder
CNPU	Check Node Processing Unit
ELB	Equivalent Logic Block
FEC	Forward Error Correction
FF	Flip-Flop
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IDS	Informed Dynamic Scheduling
IMMB	Intrinsic Message Memory Bank
LBP	Layered Belief Propagation
LDPC	Low-Density Parity Check
LLR	Logarithmic-Likelihood Ratio
NPU	Node Processing Unit
PCM	Parity-Check Matrix
QC	Quasi-Cyclic
VMEM	VND Memory
VN	Variable Node
VND	Variable Node Decoder
VNPU	Variable Node Processing Unit

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://doi.org/10.5258/SOTON/405769.

NOMENCLATURE

$\mathbf{H}$	Full PCM
$\mathbf{H}_b$	QC base PCM
m	Number of CNs/rows in $\mathbf{H}$
M	Largest value of m in supported PCM set
m_b	Number of rows in $\mathbf{H}_b$
M_B	Largest value of m_b in supported PCM set
n	Number of VNs/columns in $\mathbf{H}$
N	Largest value of n in supported PCM set
n_b	Number of columns in $\mathbf{H}_b$
N_B	Largest value of n_b in supported PCM set
d_c	CN/row degree
D_C	Largest value of d_c in supported PCM set
d_v	VN/column degree
D_V	Largest value of d_v in supported PCM set
z	QC block expansion factor
Z	Largest value of z in supported PCM set
R	Coding rate
s	Circular shift value
P	Number of parallel VNPUs
P_c	Number of parallel CNPUs
Q	Parallelism reduction factor
G	Ratio of VNs to CNs
G_{min}	Smallest value of G in supported PCM set
W	Bit-width of LLRs
L	Employment of linked CNs for the current PCM
F	Requirement for linked CNs for the PCM set
f_{max}	Maximum clock frequency
t_v	Clock cycles per VN
t_{vp}	Clock cycles per P VNs
t_{vb}	Clock cycles per Z VNs
t_c	Clock cycles per CN

t_{cp}	Clock cycles per P CNs
t_{cb}	Clock cycles per Z CNs
t_i	Clock cycles per iteration
I_a	Average number of iterations required

I. Introduction

LDPC codes [1] constitute a class of Forward Error Correction (FEC) block codes that have been the focus of much research in the communications community over the past two decades. The parametrisation of a particular LDPC code is completely defined by its Parity-Check Matrix (PCM), which describes the specific logical combination of the transmitted message bits into parity checks. These PCMs are sparse matrices having far more zero entries than non-zero, which allows LDPC codes to be iteratively decoded using a distributed low-complexity message-passing algorithm [2]. This iterative decoding process has been shown to facilitate information rates very close to the Shannon limit [3], which has motivated the inclusion of LDPC codes in many modern communications standards such as IEEE 802.11 (WiFi) [4], IEEE 802.16 (WiMAX) [5], and DVB-S2 [6]. Owing to this, hundreds of FPGA-based LDPC decoder designs have been published over the last two decades, which are comprehensively surveyed and compared in [7]. Of the many conclusions drawn by this survey, one of the most compelling is the absence of a fully flexible FPGA-based LDPC decoder architecture, which can be designed to support run-time switching between any given set of one or more LDPC codes, having diverse PCMs.

Run-time flexibility is particularly advantageous for commercial applications, since it is a requirement for a decoder to dynamically support the variety of different PCMs within a targeted communications standard [8], without requiring the extra time and technical intervention that is involved in re-programming an FPGA. Furthermore, commercial devices typically support more than one standard and run-time flexibility can allow the corresponding LDPC codes to be supported on the same FPGA [9]. Further to this, flexible decoders can adapt automatically depending on the channel conditions [10], allowing reliable communications to be maintained by dynamically adapting the code rate. Run-time flexibility can also be useful for research purposes [11], eliminating the requirement for an FPGA to be re-synthesised when testing multiple different LDPC PCMs. It is therefore desirable to have a decoder design that exhibits run-time flexibility over a selection of PCMs, within one code family or among several different families. A variety of Application-Specific Integrated Circuit (ASIC) implementations of multi-mode LDPC decoders are available in the open literature [9], [12]–[14], however, the high complexity of the adaptable routing and processing elements that are required by a flexible LDPC decoder has resulted in a lack of viable FPGA-based implementations.

The survey presented in [7] characterises the complex interactions amongst the numerous parameters that must be selected for an FPGA-based LDPC decoder design, as well as with the characteristics that are manifested for the resultant implementation. These interactions complicate the process of designing an FPGA-based LDPC decoder [15], which is exacerbated when the decoder must be designed to flexibly support multiple PCMs at run-time. However, in this paper we show that if the architecture of the decoder design is proposed in a generalised form that is not specific to any one PCM, much of this design process may be automated. In this work, we demonstrate this concept by presenting a novel generalised FPGA-based LDPC decoder architecture that can flexibly support any set of one or more Quasi-Cyclic (QC) PCMs [16] at design time, with the ability to switch between them within a single clock cycle at run time. We also present a novel offline design flow, which automates the design of a decoder that adopts the proposed architecture, without requiring the user to have in-depth knowledge of the architectural decisions involved. The chosen PCMs may originate from one family or several different families, and may vary in terms of any of their parameters.

The structure of this paper is as follows. We commence in Section II by providing the required background information regarding both general and QC LDPC codes, along with some of the key features of LDPC decoder designs. Section III then presents our novel flexible FPGA-based LDPC decoder architecture, paying particular attention to the features that enable run-time switching of PCMs. Following this, Section IV then describes our novel offline design process, which grants automated design-time flexibility and generates an efficient decoder design with minimal user input. Section V then presents the characteristics of several implementations of the proposed architecture, comparing their performance with benchmarkers, where possible. Finally, Section VI provides some concluding remarks. This structure is depicted in Fig. 1.

II. Background

Before commencing the discussion of our flexible FPGA-based LDPC decoder architecture and design flow, this section presents a brief introduction to the key concepts that motivate some of the design decisions. We begin in Section II-A with a brief description of general LDPC codes, including their characteristics and the manner in which they are decoded, before Section II-B discusses the QC codes targeted in this work. Following this, Section II-C provides a discussion of some of the key features of LDPC decoders, including the optimisations that can be achieved by targeting QC codes.

A. LDPC codes

The parametrisation of any LDPC code can be completely defined by its sparse PCM $\mathbf{H}$. The number of rows in a PCM is denoted by m, which also represents the number of parity bits employed by the code. Meanwhile, the number of columns in a PCM is denoted by n, which quantifies the encoded frame length in bits, where $n > m$. The coding rate R of the PCM is given by $R = 1 - {}^{m}/_{n}$, where $0 < R < 1$. Each row and column contains a number of non-zero elements, where that number is referred to as the degree of the row or column. If the row degree d_c is the same for every row, and the column degree d_v is the same for every column, then the PCM is said to be regular. Otherwise, the PCM is irregular and the notations d_c and d_v may be used to represent the average row and column

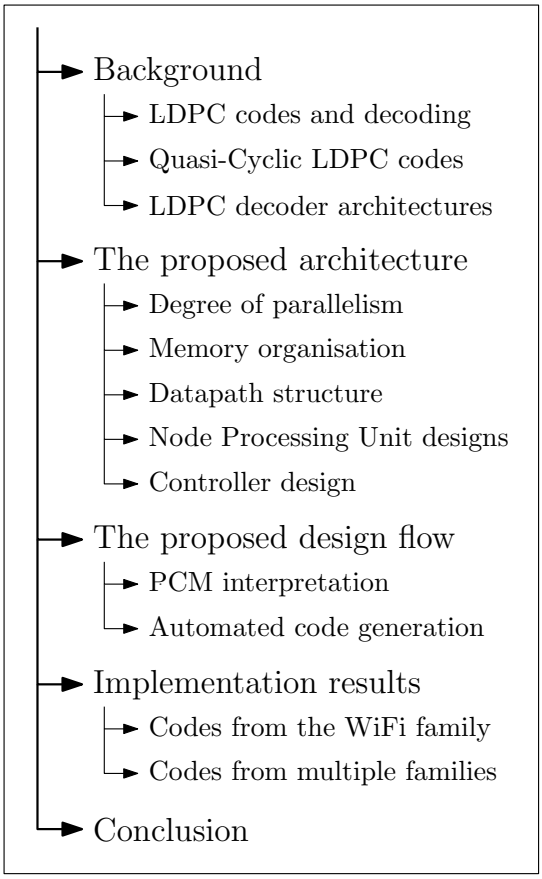


Fig. 1. Structure of this paper.

degrees, respectively. Since the number of non-zero elements must be the same when viewed from the perspective of the columns or rows, we have $m \times d_c = n \times d_v$, giving $d_c > d_v$. To illustrate the above, an example PCM $\mathbf{H}$ associated with $n = 18$ and $m = 9$ is given in Fig. 2a. Note that $\mathbf{H}$ in Fig. 2a is irregular, with an average row degree of $d_c = 4.33$ and an average column degree of $d_v = 2.17$.

During the LDPC decoding process, computations are formed on the basis of the rows and columns of the PCM. The computations for the i-th row c_i take inputs from the results of the computations performed previously by the set of columns v_j for which $H_{ij} = 1$, and vice versa. In this way, the rows and columns are processed in an iterative manner in an order dictated by a particular *schedule*, examples of which include flooding [17], Layered Belief Propagation (LBP) [18], and Informed Dynamic Scheduling (IDS) [19]. The results of the row and column calculations are extrinsic messages, which are commonly represented in the form of Logarithmic-Likelihood Ratios (LLRs) [20], which provide soft information regarding the likelihood of the corresponding bit being a 0 or 1. When using the factor graph representation of a PCM proposed by Tanner [21], each row is represented by a Check Node (CN) and each column is represented by a Variable Node (VN), with an edge linking the i-th CN to the j-th VN wherever $H_{ij} = 1$. The decoding process can therefore be pictured as the iterative exchange of extrinsic LLRs along the edges between VNs and CNs, where new extrinsic LLRs are calculated within the VNs (which perform the PCM column calculations) and the CNs (which perform the PCM row calculations). Fig. 3 presents the factor graph representation of the PCM given in Fig. 2a, where v_j represents the j-th VN, and c_i represents the i-th CN. The connections labelled $\tilde{P}_j$ above the VNs in Fig. 3 pertain to the intrinsic LLR associated with the j-th codeword bit.

B. Quasi-cyclic codes

The majority of LDPC codes employed in communications standards adopt a Quasi-Cyclic (QC) construction [22], which possesses several properties that facilitate the design of hardware-efficient decoder architectures, as will be discussed in Section II-C.

The PCM of a QC LDPC code is a specially structured realisation of the generalised unstructured PCM $\mathbf{H}$, which is defined by a base matrix $\mathbf{H}_b$, where each element of $\mathbf{H}_b$ represents a square submatrix in $\mathbf{H}$ of dimensions $z \times z$ [23]. Accordingly, $\mathbf{H}_b$ contains n_b columns and m_b rows, where $n_b \times z = n$ and $m_b \times z = m$. Similarly to the expanded PCM $\mathbf{H}$, the coding rate is given by $R = 1 - {}^{m_b}/_{n_b}$, where $n_b > m_b$. Each of the submatrices formed from the elements of $\mathbf{H}_b$ are either null matrices, containing all zeroes, or circularly-shifted identity matrices, having precisely one non-zero entry in each row and column. It can hence be seen that the PCM $\mathbf{H}$ presented in Fig. 2a is quasi-cyclic, since it is composed of a 3×6 grid of 3×3 submatrices, which are all either null matrices or circularly-shifted identity matrices. Fig. 2b presents the corresponding QC base matrix $\mathbf{H}_b$, having $z = 3$, $n_b = 6$, and $m_b = 3$. Note that a value of -1 in $\mathbf{H}_b$ corresponds to a null submatrix, whereas a non-negative value represents the number of positions s that the elements of the identity matrix are circularly shifted to the right in the corresponding submatrix. Groups of z rows or z columns of $\mathbf{H}$ corresponding to one row or column of $\mathbf{H}_b$ are henceforth referred to as *block-rows* or *block-columns*, respectively.

C. LDPC decoder architectures

One of the many LDPC decoder parameters discussed in [7] is the level of processing parallelism, which is defined by the number of parallel Node Processing Units (NPUs) that are utilised to simultaneously process rows or columns of the PCM. The level of parallelism employed can have a large effect on the decoder's hardware resource requirement, processing throughput, as well as its potential to have run-time flexibility [9]. A decoder's level of parallelism may broadly be categorised as fully-parallel, partially-parallel or serial.

Fully-parallel decoders [24]–[26] implement every CN and VN within an LDPC code's factor graph separately in dedicated hardware units. This extremely high level of parallelism can facilitate a high processing throughput and low processing latency, though this is achieved at the cost of a high hardware resource requirement [27]. Furthermore, the connections between each processing unit must either be fixed, thereby rendering the decoder only suitable for a single PCM, or require significant further hardware resources to implement flexible routing, for example by utilising a Beneš network [12].

$$\mathbf{H} = \begin{bmatrix} 0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&0 \\ 0&1&0&0&0&1&1&0&0&1&0&0&0&0&0&1&0&0 \\ 0&0&1&1&0&0&0&1&0&0&1&0&0&0&0&0&1&0 \\ 1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0 \\ 1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1 \end{bmatrix}, \quad \mathbf{H_b} = \begin{bmatrix} -1&1&-1&0&2&1 \\ 1&2&0&0&-1&0 \\ 2&-1&1&-1&2&0 \end{bmatrix}.$$

(a) (b)

Fig. 2. Example parity-check matrices. (a) Expanded binary PCM exhibiting QC construction. (b) QC base matrix representation.

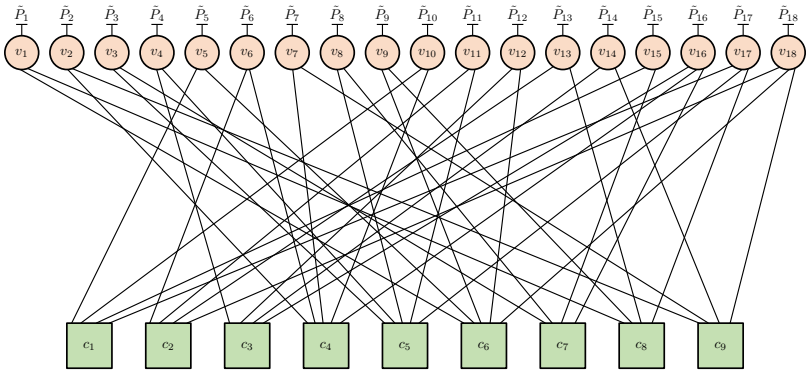


Fig. 3. Factor graph for the example QC LDPC code of Fig. 2a

These factors render the concept of a fully-parallel decoder architecture featuring run-time flexibility highly impractical.

Conversely, serial decoders [11], [28] implement only a single Check Node Processing Unit (CNPU) and a single Variable Node Processing Unit (VNPU) in hardware, thereby requiring a small amount of hardware resources. These NPUs must be used multiple times per decoding iteration in a time-multiplexed manner, where internal memories are utilised to temporarily store the extrinsic LLRs for each row and column calculated by the NPUs over the course of the iterative decoding process [24]. In an FPGA-based decoder implementation, it is common to use the FPGA's built-in Block RAMs (BRAMs) to store these messages [29], using addresses dictated by the positions of the non-zero entries in the PCM $\mathbf{H}$ and stored in lookup tables. Accordingly, serial decoders can naturally offer full run-time flexibility simply by changing the stored memory address values. However, due to the large number of operations required for each decoding iteration, serial decoders suffer from very low processing throughputs, which typically do not meet the requirements of modern communication standards.

Partially-parallel decoder architectures strike a compromise between serial and fully-parallel architectures by implementing a number P of parallel NPUs, where $1 < P < n$. Each decoding iteration is then split into several stages, wherein P VNs or CNs are processed simultaneously. This facilitates higher processing throughputs than are possible with serial architectures, while avoiding the excessive hardware resource requirements of fully-parallel architectures. Similarly to serial architectures, FPGA-based partially-parallel decoders utilise BRAMs to store interim calculation results, facilitating run-time flexibility through changing address lookup tables. However, the increased level of parallelism means that the distribution of values into BRAMs must be chosen carefully, such that when processing P rows (or columns) simultaneously, no more than one location within each column (or row) BRAM is required [30]. For unstructured codes there is no way of guaranteeing that it will be possible to avoid contention for the BRAMs in this way [31].

The semi-structured nature of QC codes described in Section II-B above may be used to optimise the design of a partially-parallel decoder in a number of ways, particularly when the parallelism P of the decoder is chosen to match the width z of the block-rows and block-columns of the PCM. Firstly, the in-memory representation of each PCM is substantially simplified by using the reduced PCM $\mathbf{H}_b$, rather than the full PCM $\mathbf{H}$. In particular, the position of all non-zero entries in the full PCM $\mathbf{H}$ can be calculated from the locations and values of the shift values in the reduced PCM $\mathbf{H}_b$ [23]. Secondly, by processing block-rows or block-columns in parallel, every row or column that is processed simultaneously will always have the same degree. For irregular codes, this reduces the total number of degrees that must be stored by a factor of z, while permitting the sharing of control signals for each NPU. Additionally, since every non-zero element within a submatrix is situated on its own row and column, it can be ensured that there will never be an occasion in which more than one location of any BRAM is contended

for at the same time. These properties form the basis of the proposed flexible partially-parallel LDPC decoder architecture to be detailed in Section III.

III. The Proposed Flexible LDPC Decoder Architecture

In this section, we detail the proposed FPGA-based LDPC decoder architecture, which has both run-time and design-time flexibility. More specifically, the architecture proposed here represents a framework for a decoder capable of decoding any chosen set of one or more QC LDPC PCMs, as outlined in Section I. Owing to this, the discussion of this section is presented in generalised terms, where the number of block-columns and block-rows n_b and m_b, the expansion factor z and the regularity and node degrees d_c and d_v are considered as variable parameters. Section IV will present a design flow that takes this general architecture and a set of one or more PCMs as inputs, in order to generate a specific LDPC decoder design by selecting desirable values for the parameters described in this section.

For the reasons stated in Section II, the architecture proposed in this section is partially-parallel in nature. This facilitates a great deal of design-time flexibility in terms of the trade-off between the processing throughput and the hardware resource requirements.

A top-level block diagram of the proposed architecture is presented in Fig. 4. The decisions motivating the various aspects of this design are discussed in the following subsections. More specifically, the parallelism and decoding schedule are discussed in Section III-A, while the memory organisation is discussed in Section III-B. Following this, the datapath, programmable barrel shifters, node processing units, and controller are discussed in Sections III-C, III-D, III-E, and III-F, respectively.

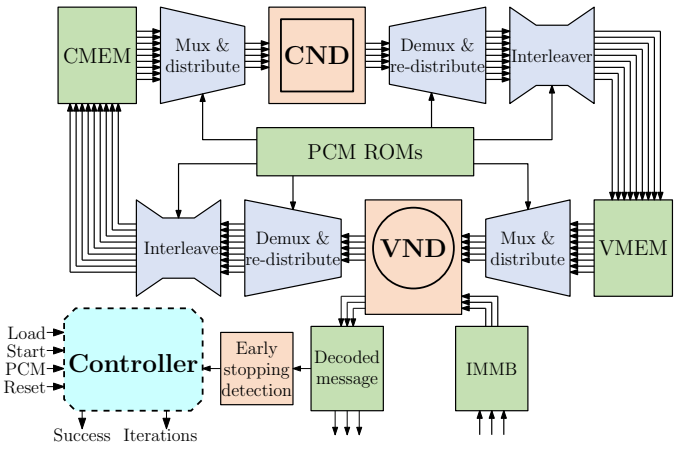


Fig. 4. Block diagram of the proposed flexible LDPC decoder architecture.

A. Parallelism and decoding schedule

In order to strike a favourable trade-off between the processing throughput and the hardware resource requirements, the proposed architecture is designed to minimise the amount of time that any processing element spends in a state where it is not producing usable results. Accordingly, the proposed decoder implements a modified flooding schedule, in which every VN and CN is processed once per iteration. However, unlike traditional flooding [17], groups of VNs and CNs are processed simultaneously, with adjacent groups within the PCM processed sequentially. In order to do so, as shown in Fig. 4, the decoder has a separate Variable Node Decoder (VND) and Check Node Decoder (CND). These may be operated in parallel, using the separate datapaths of Section III-C, along with the separate memories VMEM and CMEM of Section III-B.

The VND contains P VNPUs, each of which processes one column of the PCM in $t_v = 1$ clock cycle, with the result that P VNPUs can process P columns of the PCM in $t_{vp} = 1$ clock cycle. Each VNPU has a number of inputs and outputs equal to $D_V + 1$, where D_V is the largest column degree within the set of supported PCMs, while the additional input and output are used for intrinsic messages from the channel and the estimation of the decoded bit, respectively [32]. Implementing this maximum number of inputs and outputs is essential to ensure that each VNPU can be used to decode any column in the set of supported PCMs. Similarly, the CND contains P_c CNPUs, where $P_c \leq P$ as explained below. Each CNPU has D_C inputs and outputs, where D_C is the maximum row degree within the set of supported PCMs. This allows each CNPU to process any one row of any of the supported PCMs in $t_c = 1$ clock cycle.

Motivated by the discussion in Section II, the parallelism of the proposed partially-parallel architecture reflects the dimension z of the submatrices within the target QC PCMs. Accordingly, the parallelism factor P is given by

$$P = \left\lceil \frac{Z}{Q} \right\rceil, \tag{1}$$

where Z represents the maximum submatrix dimension z from the set of supported PCMs. Here, Q is an optional integer parallelism reduction factor, where $1 \leq Q \leq Z$. The value of Q can be chosen at design-time, allowing the decoder's trade-off between processing throughput and hardware resource usage to be controlled. When $Q = 1$, we have $P = Z$, allowing the decoder to process an entire block-column in $t_{vb} = t_{vp} = 1$ clock cycle. Decoders having larger values of Q require a higher number of clock cycles per block-column, according to $t_{vb} = Q$. Note that selecting a value of $Q = Z$ leads to a parallelism factor of $P = 1$, which results in a fully-serial decoder architecture.

The value of P_c is calculated as a factor of P, according to the relative numbers of rows and columns in the PCM within the supported set having the lowest ratio of n_b/m_b. Note that each block-column is processed using $t_{vb} = Q$ clock cycles as described above. Since the number of block-columns n_b is larger than the number of block-rows m_b, the minimum number of clock cycles required for an entire decoding iteration t_i is given by $t_i = n_b \times Q$. Since the block-rows and block-columns are processed simultaneously, the total number of clock cycles t_{cb} available for the processing of each block-row is equal to $t_{cb} = Q \times G_{min}$. Here, G_{min} is

the minimum value of G within the set of supported PCMs, where

$$G = \lfloor n_b / m_b \rfloor. \tag{2}$$

In cases where $G_{min} \geq 2$, the required number P_c of CNPUs within the CND may be reduced according to

$$P_c = \left\lceil \frac{P}{G_{min}} \right\rceil, \tag{3}$$

without increasing the number t_i of clock cycles required per iteration. More explicitly, when using P_c CNPUs, a group of P rows of the PCM may be calculated in $t_{cp} = G_{min}$ clock cycles. This reduction in the required number of CNPUs is particularly valuable, since the average row degree d_c of a PCM is always larger than the average column degree d_v, meaning that the maximum row degree D_C within a set of PCMs is typically larger than the maximum column degree D_V. Since CNPUs have D_C inputs and outputs while VNPUs have $D_V + 1$ inputs and outputs, the typical hardware resource requirement of a CNPU is much larger than that of a VNPU. Due to this, a reduction in P_c can lead to a significant reduction in a decoder's overall hardware resource usage.

B. Memory organisation

As discussed in Section II-C, FPGA-based partially-parallel LDPC decoders may take advantage of the FPGA's built-in BRAMs, in order to store the extrinsic LLRs calculated by the VND until they are needed by the CND, and vice versa. The diagram seen in Fig. 4 shows that the proposed architecture splits this extrinsic memory into two sections, named VMEM and CMEM. During each clock cycle, the VND will read up to $P \times D_V$ number of W-bit values from the VMEM and write the same number back into the CMEM. At the same time, the CMEM will read up to $P_c \times D_C$ number of W-bit values from the CMEM and write the same number back into the VMEM. This implies the requirement for a large memory bandwidth, which would restrict the achievable degree of parallelism without the careful attention to the memory management, to be described below. Note that the proposed architecture represents all LLRs using $W = 4$-bit two's complement integers, as recommended in [33].

In the proposed architecture, we denote the maximum value of n_b within the set of supported PCMs as N_B, while the maximum value of m_b within the set of supported PCMs is denoted as M_B. Note that the values of N_B and M_B do not necessarily have to originate from the same PCM. The VMEM and CMEM both then comprise N_B distinct BRAMs, each having $M_B \times Q$ address locations of width $P \times W$. The rationale for this is as follows. The number of available BRAMs on an FPGA is typically smaller than the number of rows and columns within a typical PCM $\mathbf{H}$, necessitating the grouping of extrinsic LLRs from multiple rows or columns into each BRAM word [29]. In the simplified case where $P_c = P$, it may be observed that the P extrinsic LLRs in a PCM submatrix will always be read or written simultaneously, whenever that submatrix is in the active block-row or block-column. This motivates the concatenation of P adjacent LLRs into one BRAM word, so that each set of Q words within a BRAM stores the LLRs of one complete submatrix of $\mathbf{H}$.

Using this approach, the number of BRAMs required is equal to the maximum number of PCM submatrices that may be required at once, namely D_{max}. Here, the maximum possible column degree D_{Vmax} would occur when a column in the base PCM $\mathbf{H}_b$ contains entirely non-null values, giving $D_{Vmax} = M_B$. By the same logic, we have $D_{Cmax} = N_B$. Since $N_B > M_B$, we have $D_{max} = N_B$, which results in both the VMEM and CMEM requiring N_B BRAMs, as stated previously. This ensures compatibility with any QC PCM, since the CND requires the ability to simultaneously read N_B submatrices from the CMEM and write N_B submatrices to the VMEM. Note that this arrangement assumes the availability of dual-port BRAMs, which support simultaneous read and write operations at two separate memory addresses, as can be found on most modern FPGAs [34].

Since the PCM is read in a row-centric manner by the CND and in a column-centric manner by the VND, the extrinsic LLRs must be stored within the BRAMs in a non-linear fashion, to avoid situations in which more than one address of any BRAM is required at any one time. In order to address this, we propose the layout of Fig. 5, which exemplifies one of the memories for a decoder designed to support the example PCM of Fig. 2. Here, the notation for each submatrix B_A indicates that the extrinsic LLRs associated with that submatrix are stored at address A of BRAM B. In this example we have $Q = 1$, which results in each BRAM containing $M_B \times Q = 3$ locations, each storing $Z \times W$ bits. For $Q > 1$, the distribution of submatrices and BRAMs would remain unchanged, although each submatrix would be split into Q adjacent memory locations.

Block-rows \ Block-columns	**1**	**2**	**3**	**4**	**5**	**6**
1	1_1	2_1	3_1	4_1	5_1	6_1
2	2_2	3_2	4_2	5_2	6_2	1_2
3	3_3	4_3	5_3	6_3	1_3	2_3

Fig. 5. VMEM and CMEM locations of the submatrices of an example QC PCM.

C. Datapath

In the following discussions, we detail the VND and CND datapaths.

VND: The input to the VND datapath is provided by the N_B groups of P W-bit LLRs gleaned from the VMEM, as described in Section III-B. These must be routed into the P VNPUs, which each have D_V inputs. The datapath must then take the P groups of D_V outputs from the VND and route them back into N_B groups of P messages for writing into the CMEM. The proposed VND datapath is exemplified in Fig. 6, for the case of a decoder designed to support a single

example code, having parameters of $Q = 1$, $P = 8$, $N_B = 7$, and $D_V = 4$. The flow of data is from the VMEM on the left, through the VND in the middle, to the CMEM on the right. The eight different colours each represent the $P = 8$ columns of the active block-column, with each line representing one LLR. The total number of W-bit LLRs passed by each stage is displayed at the top of Fig. 6. Note that the intrinsic LLR inputs and the decoded bit outputs for each VNPU have been omitted for simplicity. These will be detailed during the discussion of the VNPU architecture in Section III-E.

Using the memory organisation system of Section III-B, it can be seen that the third block-column within the PCM is being processed in Fig. 6, since BRAMs 3, 4, 5, and 6 of the VMEM and CMEM contain the active submatrices. The programmable multiplexer, labelled **Mux** in Fig. 6, ensures that only these D_V groups are passed to the next stage. Additionally, it can be seen that the degree of the active block-column is 3, since a null submatrix is stored in BRAM 5, as represented by the dashed lines at its output. Instead of forwarding the unwanted BRAM contents to the VND, the multiplexer instead provides the corresponding wires with a value that represents the input LLRs being "turned off". For a VNPU, this "off" value is equal to LLR values of 0, as will be explained further in Section III-E. In Fig. 6, this is represented by the black lines emanating from the third output group of the multiplexer.

Subsequently, the D_V groups of P LLRs output from the multiplexer are distributed into P groups of D_V LLRs on a per-VNPU basis by the **Distributor**. Note that the action of the Distributor is only shown in Fig. 6 for three VNPUs, for the sake of avoiding obfuscation. Due to the action of the multiplexer described above, this distributor unit does not require any run-time programmability in the VND datapath. These messages are then processed by the VNPUs, as described in more detail in Section III-E.

Once processed by the VNPUs, the messages are re-distributed to their original D_V groups by the **Re-distributor**, which performs the inverse operation of the Distributor. They are then cyclically-shifted by D_V programmable Barrel Shifters (BSs) in the **Interleaver**, as will be described in Section III-D. Finally, a programmable de-multiplexer (labelled **De-mux**) then performs the inverse operation to the multiplexer at the start of the datapath, to place the output values at the input ports for the correct D_V BRAMs within the CMEM.

CND: The datapath for the CND performs largely the same functions as the VND datapath, albeit with some additional complexities. As described in Section III-A, P_c CNPUs are employed to process P rows over $t_{cp} = G_{min}$ clock cycles, where G_{min} and P_c are defined in (2) and (3), respectively. The spread of the operation over $t_{cp} = G_{min}$ clock cycles is achieved through the use of Flip-Flops (FFs) and additional multiplexers embedded in the datapath between the CMEM and the CND, in order to ensure the CND is only presented with P_c inputs at a time. De-multiplexers and additional FFs are then employed between the CND and the VMEM so that all P messages are available during the same clock cycle for writing back into the VMEM.

An additional optimisation is performed in the proposed architecture to reduce the hardware resource usage of decoders designed to have run-time flexibility over a set of PCMs having a wide variety of coding rates R. The reasoning behind this is as follows. Many standardised LDPC code families (such as those in WiFi [4], WiMAX [5], and WiGig [35]) vary the coding rates $R = 1 - {}^{m_b}/_{n_b}$ of different related codes by changing m_b, while leaving n_b unchanged. Since $m_b \times d_c = n_b \times d_v$, PCMs with lower values of m_b will hence typically have larger row degrees d_c. More specifically, high-rate codes tend to have a low number of rows, each having high degrees; by contrast, low-rate codes tend to have a high number of rows, each having low degrees. In order to address this, the proposed architecture allows two low-degree CNPUs to be combined to process one high-degree row. Doing so doubles the number of clock cycles t_{cb} required per block-row, although this cost is offset by the smaller number m_b of block-rows present in these high-rate PCMs. This optimisation halves the number of inputs and outputs that may be required by each CNPU within the proposed architecture. The internal operation of two low-degree CNPUs that have been linked in order to provide the functionality of one high-degree CNPU is detailed in Section III-E.

In order to support this functionality in the CND datapath, an additional Boolean parameter L is introduced for each supported PCM, as well as an additional Boolean decoder parameter F. For each PCM i, the parameter $L^i = 1$ indicates that each row should be processed using two linked CNPUs. It is calculated according to

$$L^i = \begin{cases} 1, & \text{if } G^i \geq G_{min} \times 2 \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where G_{min} is defined in Section III-A as the minimum value of G^i for all PCMs i within the chosen set. However, as will be shown in Section III-E, the hardware required to facilitate the linking of two CNPUs slightly increases the hardware resource requirement and the critical path length of the CNPU. For this reason, these flexible CNPUs should not be synthesised unless the chosen PCM set necessitates it. More specifically, flexible CNPUs are only synthesised at design-time if at least one of the PCMs has a value $L^i = 1$, in which case we set the Boolean decoder parameter $F = 1$.

An example of these optimisations is presented in the flexible CND datapath of Fig. 7 for a decoder with $F = 1$, when used to implement both a low-rate and a high-rate code. Here, both codes have $n_b = 7$ and the minimum value of G is calculated from the low-rate code, giving $G_{min} = 2$. The selection of $Q = 1$ leads to $P = Z = 8$, and thus $P_c = 4$, according to (3). Fig. 7a represents the lower $R = {}^1/_2$-rate code where $L^i = 0$, meaning that $P_c = 4$ CNPUs of degree ${}^{D_C}/_2 = 3$ are employed to process the $P = 8$ rows within $t_{cb} = t_{cp} = G = 2$ clock cycles. Conversely, Fig. 7b represents the higher $R = {}^3/_4$-rate code, where $L^i = 1$, meaning that the $P_c = 4$ CNPUs are combined to form ${}^{P_c}/_2 = 2$ linked CNPUs of degree $D_C = 6$, requiring $t_{cb} = 2t_{cp} = 2G = 4$ clock cycles to process the block-row. However, as this high-rate code also has half the number of block-rows m_b of the low-

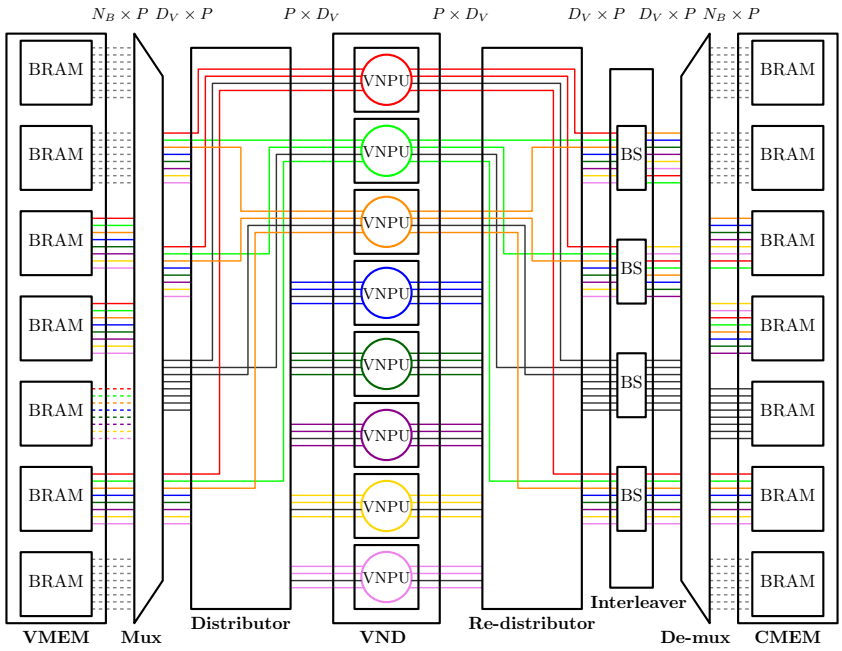


Fig. 6. Example of a VND datapath.

rate code, this does not produce any change in the number of clock cycles t_i required per decoding iteration.

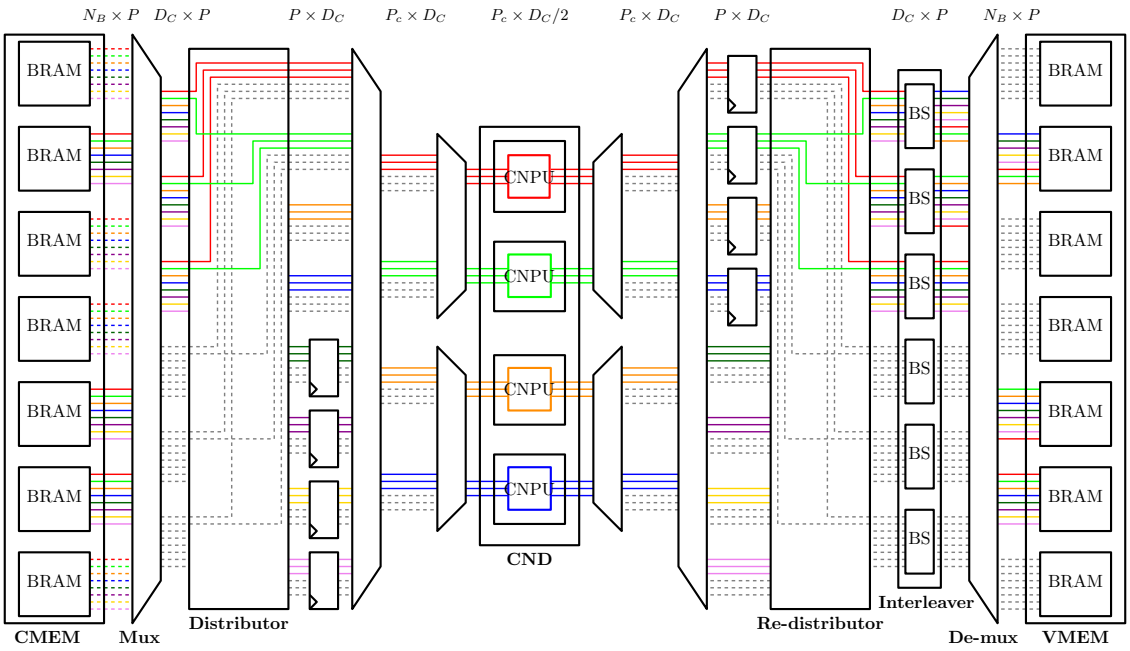
Looking firstly at the low-rate configuration of Fig. 7a, it may be observed that only ${}^{D_C}/_2 = 3$ of the $N_B = 7$ BRAMs in the CMEM contain non-null submatrices for the active block-row, namely those representing block-columns 2, 5, and 6. In this configuration, the first multiplexer, labelled **Mux** in Fig. 7, must ensure that these inputs form the top half of its $D_C = 6$ outputs, while its bottom ${}^{D_C}/_2 = 3$ outputs are effectively "don't care" values, represented by dashed grey lines in Fig. 7a. Similarly to the VND datapath explained previously, when the active block-row has a degree less than ${}^{D_C}/_2$ this multiplexer would also "turn off" the outputs from the extra null submatrices. For a CNPU, this "off" value corresponds to an output LLR value equal to $(2^{W-1} - 1)$, as will be explained further in Section III-E.

In this configuration, the operation of the **Distributor** is the same as that in the VND datapath, with each of its P output groups of $D_C = 6$ LLRs comprised of ${}^{D_C}/_2 = 3$ extrinsic messages (or "off" values) followed by ${}^{D_C}/_2$ "don't care" values. Note that the action of the Distributor is only shown in Fig. 7 for two CNPUs, for the sake of avoiding obfuscation. The first FFs and multiplexer after the distributor perform the function of reducing the $P = 8$ groups by a factor of $G_{min} = 2$ to form $P_c = {}^{P}/_{G_{min}} = 4$ groups. It can be seen that in the first clock cycle, the multiplexer will select the top $P_c = 4$ rows (red, light green, orange, and blue), which will be processed and then latched by the FFs before the re-distributor. In the second clock cycle, the multiplexer will select the latched values of the following $P_c = 4$ rows (dark green, purple, yellow, and pink), which will then be processed and re-distributed alongside the top values. The multiplexers at the input and output of the CND select the top ${}^{D_C}/_2$ lines from each P_c group, presenting P_c groups of ${}^{D_C}/_2$ values to each CNPU as required. The rest of the datapath operates identically to the VND datapath described previously, including the $D_C = 6$ programmable BSs that rotate the messages, and the de-multiplexer to provide the VMEM inputs.

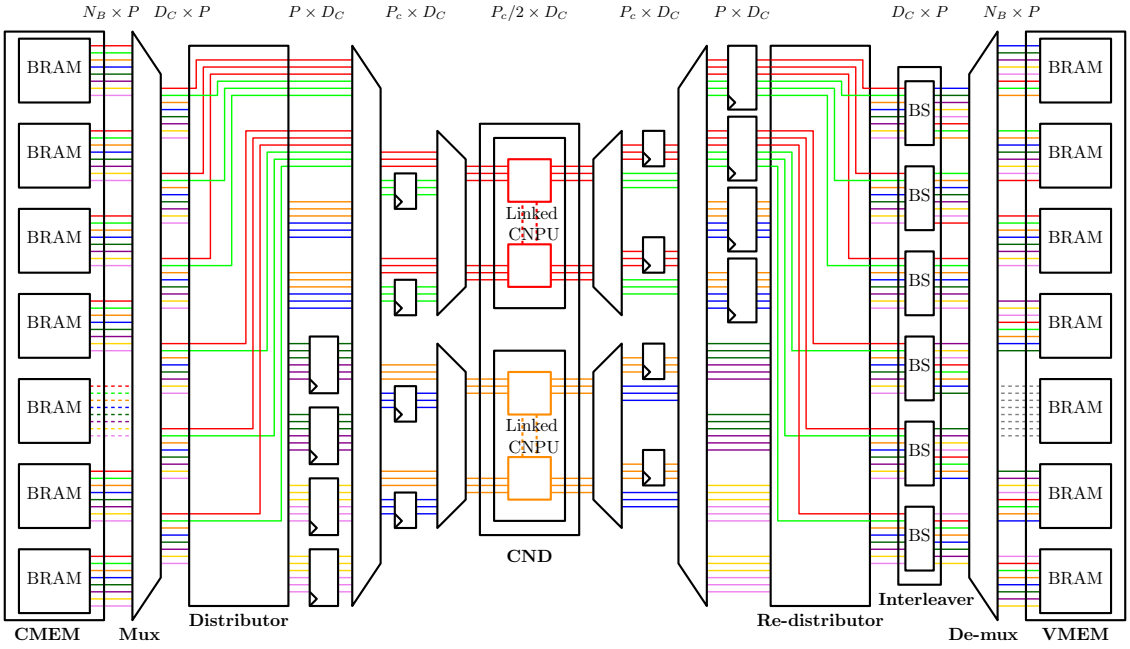
The high-rate configuration shown in Fig. 7b behaves slightly differently in several distinctive ways. Firstly, due to the fact that up to $D_C = 6$ input submatrices contain non-zero elements, the initial multiplexer no longer provides "don't care" data on any of its outputs (although it may still provide "off" values, when the active row degree is less than D_C). The action of the distributor also changes to divide each group of $D_C = 6$ messages across two outputs, in order to simplify the routing and control later in the datapath. The first set of FFs and multiplexer after the distributor perform the equivalent function to their counterparts in Fig. 7a, namely reducing the $P = 8$ groups to $P_c = 4$ groups. The following FFs and multiplexer act in a similar way to route the top ${}^{D_C}/_2 = 3$ messages to the top half of the merged CNPU, while routing the bottom ${}^{D_C}/_2 = 3$ messages of the same row to the bottom half of the merged CNPU. It can therefore be seen that the first and third rows (red and orange) will be processed in the first clock cycle, followed by the second and fourth rows (light green and blue) in the second clock cycle, and so on. These FFs are avoided in the low-rate configuration using extra multiplexers, which are not shown in Fig. 7 for simplicity.

D. Programmable barrel shifters

Once a group of P messages has been processed by the VND (or CND), they must be converted from a column- (or row-) centric representation to a row- (or column-) centric representation, before being written into the CMEM (or VMEM). This ensures that the messages are in the correct order to be read by the CND (or VND). This may be performed by cyclically shifting the messages for each submatrix by the



(a)



(b)

Fig. 7. Example of a CND datapath with conditional CNPU sharing. (a) Low-rate codes with $L = 0$, (b) High-rate codes with $L = 1$.

corresponding shift value s in $\mathbf{H}_b$, which is performed by the programmable BSs in the interleaver, as mentioned in Section III-C.

Since the shift value s may be any integer $0 \leq s < Z$, each BS must have Z inputs and outputs. This means that when the parallelism reduction factor Q is utilised to employ a reduced number $P < Z$ of NPUs, additional FFs must be used at the BS inputs to hold each group of P NPU outputs until all z outputs have arrived and the shifting can be completed. Similarly, additional FFs are employed to hold the Z BS outputs after they have been shifted, so that groups of P outputs may be presented to the memories in each clock cycle.

The design of a BS that supports only a single submatrix size $z = Z$ is trivial compared to that of one which can programmatically adapt to multiple submatrix sizes at run-time [9]. Accordingly, several competing designs have been proposed in the literature [14], [36]–[40], each with their own strengths and weaknesses. The design proposed in [39] offers a straightforward solution with a short critical path length, although its ability to support any value of $z \leq Z$ at run-time results in a significantly higher hardware resource requirement

than is necessary, when the fixed subset of supported z values is known at design time. Accordingly, this work employs a modified version of the *fine cyclic shift network* proposed in [40], whereby each BS is optimally designed for the specific set of z values it is intended to support. In this design, the Z inputs are cyclically shifted by s, regardless of the current value of z. Each output B_e, $0 \leq e < Z$, may then be selected from this shifted input according to the current value of z, using a u-to-1 multiplexer, where u is the number of supported z values greater than e. In order to optimise the synthesis for FPGA implementation, the specific Hardware Description Language (HDL) description of the BSs employed in the proposed decoder is automatically tailored for the PCM set specified at design time, using the software described in Section IV.

E. NPU designs

The design of the VNPUs and CNPUs can also have a significant effect on the overall hardware resource requirements of an LDPC decoder, as well as its critical path, and hence maximum clock frequency (f_{max}), processing throughput, and latency. In addition, this effect may be influenced further by the manner in which the HDL description of each NPU is written and inferred into hardware. In order to investigate this, various structures and implementations of VNPUs and CNPUs were synthesised and compared in order to find a design with an optimised combination of low critical path length and low hardware resource requirements. The chosen designs are summarised in this section.

VNPU design: The VNPU has D_V *a priori* LLR inputs and one intrinsic LLR input provided by the channel. It also has D_V extrinsic LLR outputs and one *a posteriori* LLR output used as the basis of a decision for the corresponding LDPC-encoded bit. Each of the D_V *a priori* LLR inputs to the VNPU architecture corresponds to one of the D_V extrinsic LLR outputs, which is vertically aligned in Fig. 8. For each of its D_V extrinsic LLR outputs, the VNPU is required to calculate the sum of the intrinsic LLR input and all *a priori* LLR inputs besides the corresponding one. The *a posteriori* LLR output is obtained as the sum of the intrinsic LLR input and all *a priori* LLR inputs. As described in Section III-C, when a VNPU is used to calculate a PCM column having a degree less than D_V, the unused inputs can be "turned off" by supplying an input LLR value of 0, since this does not affect the above-mentioned summations.

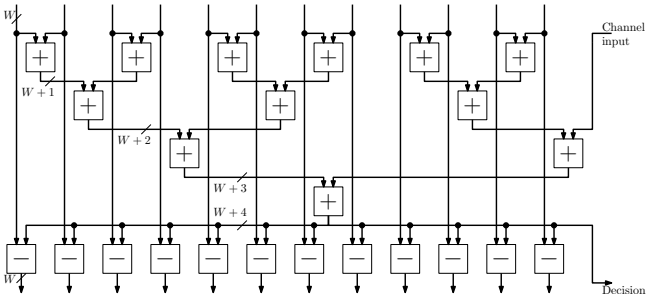


Fig. 8. The chosen VNPU structure, shown with $D_V = 12$.

As shown in Fig. 8, the chosen architecture generates the total sum of all D_V+1 W-bit LLR inputs using a tree structure of additions in order to ensure the shortest critical path, as will be described in more detail in Section IV-C. Once this has been calculated, the inverse (minus) operation is used to calculate each output as the total sum without its corresponding input. Note that the result of each addition requires the expansion of the bit width by one bit in order to avoid overflow, although each output is saturated back down to W bits following the subtraction.

Through extensive simulations, it was found that manually defining each internal addition in the HDL code yielded an NPU associated with a lower critical path than that of a functionally-identical design that was specified using high-level coding structures (i.e. nested "for" loops). The method conceived for automatically generating this HDL code for VNPUs having any number of inputs and outputs is discussed in Section IV.

CNPU design: The CNPU has D_C *a priori* LLR inputs, each of which correspond to one of its D_C extrinsic LLR outputs. In order to simplify the decoding hardware, the CN operation from the min-sum algorithm [41] is employed in the proposed decoder. More specifically, this algorithm calculates each of the D_C extrinsic LLR outputs as the minimum of all *a priori* LLR inputs besides the corresponding one, multiplied by the sign of their cumulative product. Similarly to the VNPU, when using a CNPU to process a PCM column having a degree that is lower than D_C, the extra LLR inputs can be "turned off" by supplying an LLR value of $(2^{W-1}-1)$, which is the maximum positive value that can be represented by a W-bit two's complement integer. These inputs will not affect the outputs, since their magnitude will not uniquely represent a minimum, while their positive sign will not affect the cumulative sign product.

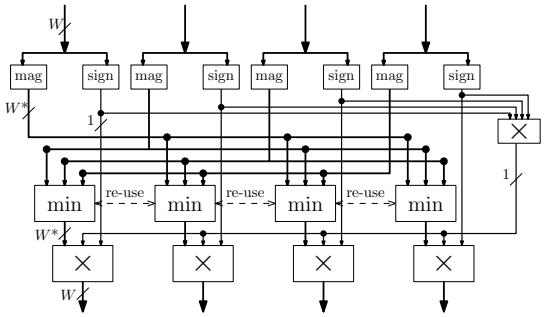


Fig. 9. The chosen CNPU structure, shown with $D_C = 4$.

The function of a CNPU is more complex than that of a VNPU, since the min operation cannot be inverted. Owing to this, the chosen CNPU architecture of Fig. 9 calculates each output separately, employing D_C tree structures to find the minimum of the magnitudes of each combination of D_C-1 inputs. However, to reduce the total hardware resource requirements, these tree structures are linked together to make the maximum possible re-use of the already calculated minima between trees, as in [42]. At the same time, the total cumulative sign of all D_C inputs is calculated. Each output is then calculated as its corresponding minimum value multiplied both

by the total sign and by the sign of its original input. Note that in Fig. 9 the mag operation returns the magnitude of the input, while a bus width of $W*$ represents W unsigned bits having a range 0 to $(2^W - 1)$. The signed W-bit outputs are saturated in the range (-2^{W-1}) to $(2^{W-1} - 1)$ as normal.

Flexible CNPUs: Section III-C demonstrates the need for two low-degree CNPUs to have the capability to optionally link together, in order to form one high-degree CNPU. This functionality may be added to the chosen CNPU architecture by including additional outputs representing the minimum magnitude of all D_C inputs, along with their cumulative sign. These outputs are then used in an optional additional stage within the linked node, such that each output is based on the minimum of its own $D_C - 1$ inputs and the D_C inputs from the paired node. This arrangement is shown in Fig. 10, which is similar to Fig. 9 but with the additional stages listed here. The additional inputs from the linked node are shown in red, while the additional outputs are shown in blue. The value of L for the current PCM is used as a control signal to dictate whether the CNPU is operating as part of a pair or not. Note that these additions slightly increase both the hardware resource usage and the critical path of each CNPU, hence they are only synthesised in the proposed architecture if the decoder parameter of $F = 1$ indicates that they are needed by at least one of the supported PCMs, as detailed in Section III-C.

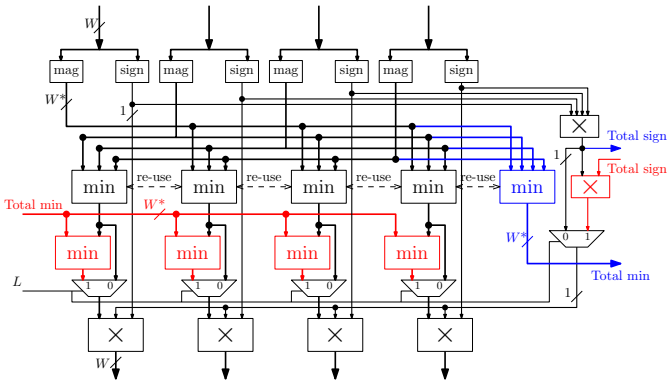


Fig. 10. Flexible CNPU structure, shown with $D_C = 4$. Inputs from a paired CNPU are shown in red, while outputs to the paired CNPU are shown in blue.

F. Controller

The controller, depicted in the bottom-left of Fig. 4, controls the progress of the overall iterative decoding process. This must provide external control signals for starting, stopping, and reseting the decoding process, as well as for selecting which of the supported PCMs to use. The controller also has to determine whether the decoding process has led to a legitimate codeword. These aspects are described in turn in the following discussions.

Control signals: The **load** input signal may be used to indicate that a new set of input intrinsic LLRs should be loaded into the Intrinsic Message Memory Bank (IMMB). Once this has been performed, the **reset** input signal may be used to reset all internal components to their initial values, before asserting the **start** input signal to commence the iterative decoding process.

Early stopping detection: The proposed architecture includes an early stopping detection unit for determining when a valid codeword has been found. This module comprises M parity-check registers, which are reset at the beginning of each decoding iteration. During the iterative decoding process, the i-th register is exclusive-OR'd with the decision provided by the VNPU that processes column j if $H_{ij} = 1$. At the end of each decoding iteration, if the first m parity check registers (where $m \leq M$ relates to the number of parity checks in the currently active PCM) all contain 0, the decoding process is deemed to have been successful and the **success** output is set. Otherwise, the parity check registers are reset and another decoding iteration begins. The total number of decoding iterations performed is recorded and output on the **iterations** signal of Fig. 4. This allows the operator of the decoder to determine at run-time when to terminate a decoding process, using the start and reset control signals described previously.

PCM selection: One of the key features of the proposed decoder design is its ability to switch the specific PCM it is currently using to decode the message stored in the IMMB within a single clock cycle. In order to implement this level of flexibility, the supported PCMs are fully characterised in a hardware-optimised form at compile-time, and written into ROMs. These PCM ROMs are used extensively within the datapaths for the VND and CND described in Section III-C to control the routing and shifting of *a priori* and extrinsic LLRs between memory banks and NPUs. The value of the multi-bit **PCM** input signal is used to select which ROMs are currently in use at any given time. For each PCM, there are two parallel sets of PCM ROMs, namely one for the VND, which views the base PCM $\mathbf{H}_b$ as n_b columns of d_v values, as well as one for the CND, which views the base PCM as m_b rows of d_c values. Within each set, there are multiple ROMs, which are used to store the values and locations of the non-null entries of $\mathbf{H}_b$.

IV. The Proposed Offline Design Flow

In this section, we detail the proposed design flow, which facilitates the automated design-time flexibility of the proposed LDPC decoder architecture. Our design flow automatically generates all of the HDL files required to implement an LDPC decoder having run-time flexibility over a set of one or more QC PCMs selected by the user. This design flow involves extracting the key parameters and values from the set of PCMs, generating optimised NPU structures and automatically producing a robust HDL description from which we can then synthesise the hardware. A flowchart depicting this design flow is presented in Fig. 11.

We begin in Section IV-A with a discussion of how the parameters may be extracted from a user-specified set of PCMs, before Section IV-B discusses the generation of a correspondingly optimised HDL description for the architecture of Section III. A particular aspect of this process is described in Section IV-C, namely the NPU tree generation.

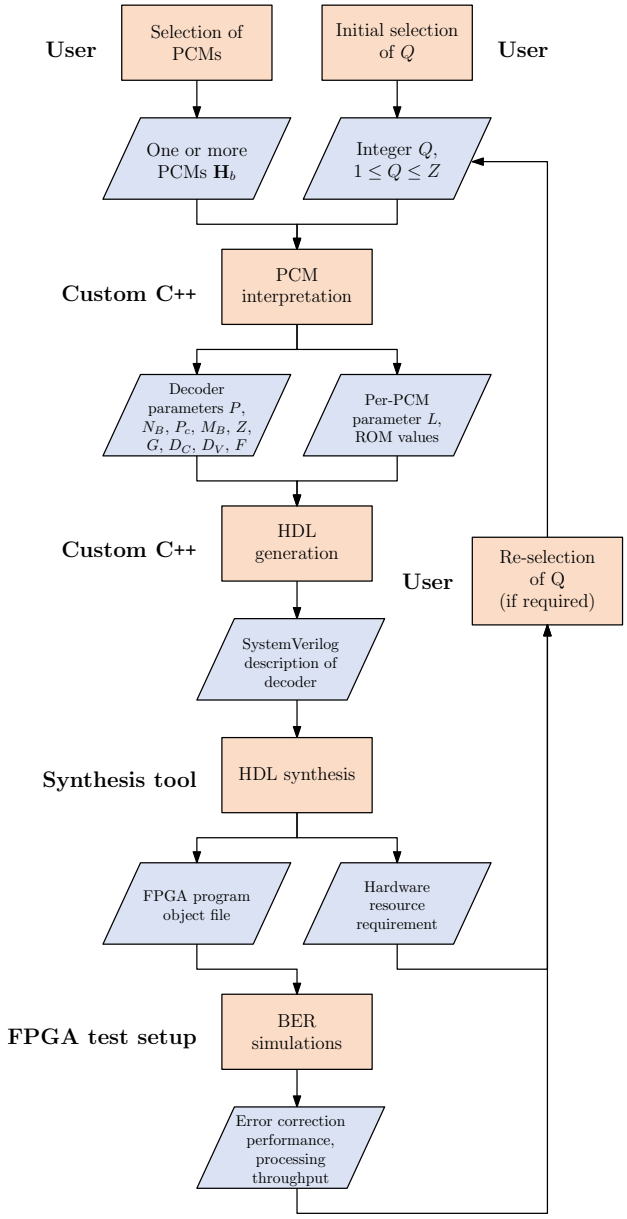


Fig. 11. The design and implementation stages involved in the generation of an FPGA-based LDPC decoder having run-time flexibility.

A. PCM interpretation

Before the HDL of the proposed design can be generated, a small amount of offline computation is required to extract the required data from the PCM(s) chosen by the user, as well as to convert the elements within each PCM into an optimised format for storage in the decoder's ROMs.

As depicted at the top of Fig. 11, the first role of our PCM interpretation software is to calculate the decoder's parameters based on the characteristics of the user-specified PCM set and the selected parallelism reduction factor Q. These parameters, along with their derivations, were enumerated throughout the description of the proposed architecture in Section III. Specifically, these are: the parallelism factors P and P_c; the maxima of the various PCM parameters N_B, M_B, Z, D_C, and D_V; the ratio of VNPUs to CNPUs G_{min}; the flexibility parameter F; and an additional per-PCM flexibility parameter L.

The second task of the PCM interpretation software depicted in Fig. 11 is to extract the locations and shift values of the non-null submatrices in each PCM $\mathbf{H}_b$ and arrange them in a format compatible with the ROMs introduced in Section III-F. Here, each row r_i, where $i \in [1, m_b]$, in each $\mathbf{H}_b$ must be cyclic-shifted to the right by $i-1$ places, to ensure that all addresses reflect the manner in which the extrinsic LLRs associated with each submatrix are stored in the VMEM and CMEM, as described in Section III-B. The ROMs are then populated with values regarding the presence, location, and value of the non-null entries in the shifted $\mathbf{H}_b$.

B. HDL generation

In order to programmatically generate SystemVerilog code within a custom application, a complete SystemVerilog generation library has been written in C++. In this way, the offline design flow can generate the complete SystemVerilog description of an LDPC decoder having the proposed architecture, using the parameters and values calculated in the previous stage, as depicted in Fig. 11. In addition to automatically generating robust code using the appropriate parameters, this approach permits the optional inclusion of certain elements which may not be required in all applications. These elements include the PCM selection input signal, which is not required when the decoder is designed to only support a single PCM, as well as the flexible CNPUs, which are not required when the decoder flexibility parameter has the value $F = 0$. Additionally, as described in Section III-D, the design of the programmable BSs employed by the decoder may be optimised for the specific set of supported submatrix sizes z. The output files generated by this library may be read and edited manually if desired, before being used by any synthesis tool that supports SystemVerilog HDL. As shown in Fig. 11, synthesis of these files by an appropriate tool will produce a measurement of the hardware requirements of the resultant decoder, while its processing throughput and error correction performance may be characterised through Bit Error Rate (BER) simulations on an FPGA test device. These characteristics may be used to guide the selection of a different value for the parallelism reduction factor Q if desired, which will then require the design-flow to be repeated, as depicted in Fig. 11.

C. NPU tree generation

It was observed in Section III-E that fully specifying the desired tree structures within the nodes resulted in hardware with a preferred combination of resource usage and operating frequency, when compared to structures determined entirely by the synthesis tool. However, in order to implement this in the proposed automated design flow, the design of this tree structure must be algorithmically defined.

A minimum-depth tree structure may be formed by exploiting the observation that any positive integer y can be calculated as the sum of two positive integers x_1 and x_2, where x_1 is the highest power of two less than y. This process may then be applied recursively by using x_1 or x_2 as y, generating a

tree of 2-input functions having a critical path containing at most $\lceil \log_2(y) \rceil$ additions. In this manner, the internal structure of the VNPUs can be programmatically defined by setting $y = D_V + 1$, yielding the result depicted in Fig. 8 for $D_V = 12$. This structure may also be used repeatedly in the CNPUs, where the $\min$ function is used instead of additions. However, rather than creating a single tree for the combination of D_C elements, instead D_C trees of $(D_C - 1)$ elements are required. In order to reduce excessive hardware usage, the nodes are designed to re-use calculated results as many times as possible, using the method described in [42]. The HDL generation application mentioned previously uses this technique to generate the complete precise description of the required hardware for NPUs of any degree, without relying on the variable results which may be produced by synthesis tools.

Note that an alternative structure proposed in [43] would further reduce the number of $\min$ operations required to $3 \times (D_C - 2)$. However, for most values of D_C, this structure also produces a longer critical path than the repeated tree structure of [42], thereby reducing the maximum operating frequency f_{max}. Since the hardware requirement of each individual $\min$ operation is small compared to the hardware requirement of the decoder as a whole, it may be expected that the repeated tree structure of [42] yields a greater hardware efficiency.

V. Implementation Results

In this section, we characterise several instances of the proposed decoder, having a variety of configurations, which we compare to relevant benchmarkers, where possible. In order to highlight the practicality of the proposed decoder, our comparisons focus on the QC PCMs of four major wireless communications standards, namely IEEE 802.11n/ac (WiFi) [4], IEEE 802.16e (WiMAX) [5], IEEE 802.15.3c (Wireless Personal Area Network, henceforth referred to as WPAN) [44], and IEEE 802.11ad (WiGig) [35].

The remainder of this section is structured as follows. Firstly, Section V-A discusses the methods employed for quantifying the characteristics of the proposed decoder, in order to be comparable with the survey of [7]. Following this, parametrisations of the proposed decoder that are targeted at PCMs from the WiFi family are characterised in Section V-B. Finally, Section V-C characterises parametrisations of the proposed decoder that are targeted at PCMs from more than one family simultaneously. It should be noted that, since the proposed architecture has a very high degree of flexibility, it would be impractical to characterise every combination of PCMs that it can be configured to support. Owing to this, the results presented here are selected as representative samples of possible combinations, in order to highlight key features and issues related to the proposed architecture's performance in practical applications.

A. Method

For each chosen set of QC PCMs, a value was selected for the parallelism reduction factor Q, and then the offline design process presented in Section IV was utilised to produce a SystemVerilog description of a decoder having the architecture presented in Section III. This description was then synthesised using Altera Quartus II for an Altera Stratix IV EP4SGX530 FPGA. In this way, we were able to quantify both the maximum operating frequency f_{max} and the hardware resource usage. More specifically, we use the Equivalent Logic Blocks (ELBs) metric proposed in [7] for characterising the hardware resource usage of each synthesised decoder, in order to facilitate a comparison with the results of that survey.

In addition to this, bit-accurate C++ BER simulations were performed, in order to characterise the error-correction performance of the synthesised decoder for each of its target PCMs. These simulations assumed Binary Phase-Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel, in common with the majority of previous FPGA implementations of LDPC decoders [7]. Our simulations considered a maximum of 18 decoding iterations per frame and a minimum of 100 frame errors per BER measurement, in order to ensure statistical significance. Using these results, the transmission energy efficiency may be characterised as the value of the channel's signal to noise power ratio per bit E_b/N_0 at which a desirable target BER of 10^{-4} is achieved, as employed in [7]. Finally, the average number of decoding iterations I_a required to achieve this BER performance at this E_b/N_0 value was characterised and used to calculate the decoded processing throughput T, according to

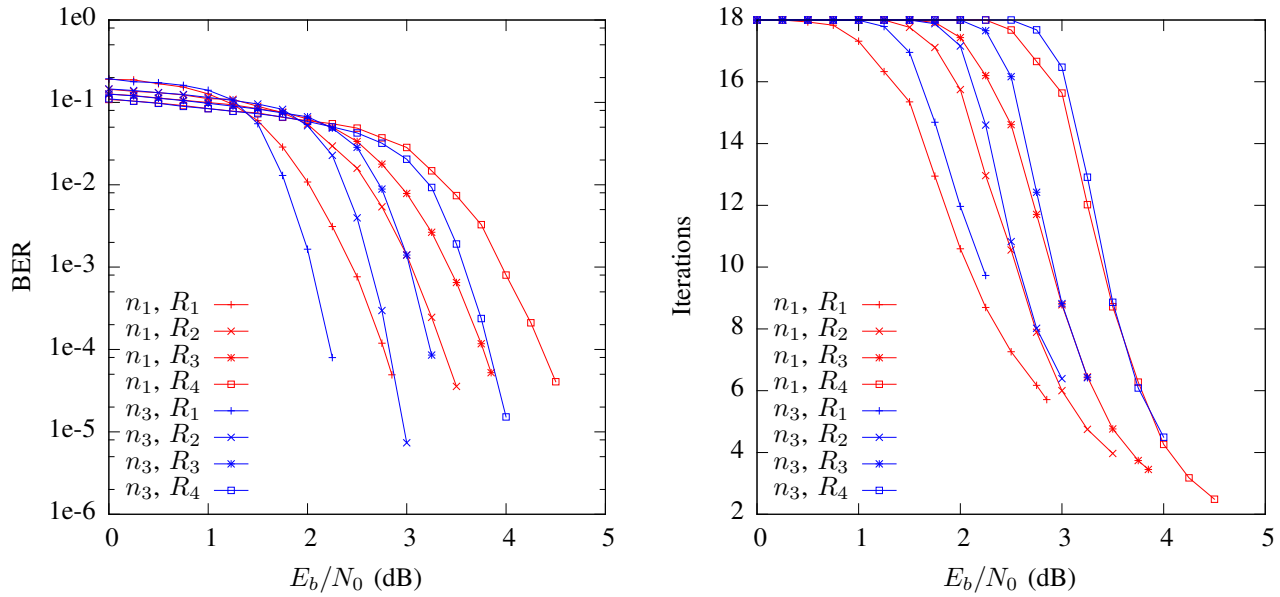
$$T = \frac{f_{max} \times n \times R}{t_i \times I_a}, \tag{5}$$

where n is the encoded frame length, R is the coding rate, and t_i is the number of clock cycles required per decoding iteration. Since the proposed architecture processes one frame at a time, the processing latency can be calculated as the ratio of the message word length $k = n - m$ to the decoded processing throughput T. An example BER plot for the eight WiFi PCMs having the shortest and longest block lengths is presented in Fig. 12a. The accompanying plot of the average number of iterations required per frame is presented in Fig. 12b.

B. Decoders targeted at WiFi LDPC PCMs

The standard on which WiFi is based, namely IEEE 802.11n/ac [4], defines 12 QC LDPC PCMs, based on each combination of three different frame lengths ($n_1 = 648$, $n_2 = 1296$, and $n_3 = 1944$) and four different coding rates ($R_1 = 1/2$, $R_2 = 2/3$, $R_3 = 3/4$, and $R_4 = 5/6$). All 12 of the PCMs have $n_b = 24$ block-columns, resulting in three different values of $z = n/n_b$ corresponding to the three frame lengths. Furthermore, the number of block-rows m_b employed by each PCM may be calculated according to $m_b = n_b \times (1 - R)$.

The FPGA-based LDPC decoder presented in [45] employs a parameterised architecture and an offline design flow similar to those proposed in Sections III and IV. The authors state that the decoder parameters are "limited run-time reconfigurable by over allocation and selective enabling", however no further



(a) (b)

Fig. 12. Simulation results for codes from the WiFi family having frame lengths $n_1 = 648$ bits and $n_3 = 1944$ bits, with decoding rates $R_1 = 1/2$, $R_2 = 2/3$, $R_3 = 3/4$, and $R_4 = 5/6$. (a) BER results, assuming BPSK transmission over an AWGN channel, with a maximum of 18 decoding iterations per frame. (b) Average number of decoding iterations required per frame for finding a legitimate codeword.

elaboration is provided. The only characteristics presented in [45] for the WiFi LDPC code correspond to a non-flexible decoder parametrisation, which was designed for the single PCM having $n = n_3 = 1944$ and $R = R_1 = 1/2$. In the survey of [7], the design of [46] provided the only other FPGA-based LDPC decoder that supports run-time flexibility across all 12 of the WiFi PCMs. However, [46] only quantifies the processing throughput, hardware resource requirements and transmission energy efficiency for this decoder when the PCM associated with $n = n_2 = 1296$ and $R = R_2 = 2/3$ is active.

In order to facilitate comparisons with the FPGA-based LDPC decoders of [45] and [46], four separate instances of the decoder proposed in this work were implemented and characterised for WiFi PCMs. We refer to our first instance as decoder F1, which targets the single PCM with the largest frame length $n_3 = 1944$ and lowest coding rate $R_1 = 1/2$, facilitating a direct comparison with the results from [45]. Our second decoder F2 targets the three PCMs with $R = R_1 = 1/2$, having frame lengths $n_1 = 648$ to $n_3 = 1944$. Thirdly, decoder F3 targets the four PCMs with frame length $n_1 = 648$, having rates $R_1 = 1/2$ to $R_4 = 5/6$. Finally, decoder F4 targets all 12 PCMs in the WiFi family, facilitating a direct comparison to the results of [46].

Table I characterises F1, F2, F3, and F4, as well as the designs of [45] and [46]. Note that without the use of the parallelism reduction factor Q, the large values of z in the WiFi LDPC code family would result in extremely high hardware resource requirements for F1, F2, and F4. Accordingly, we selected values of Q for each of these decoders such that they all employ $P = 27$ VNPUs, as shown in Table I. Furthermore, the results of Table I are plotted in Fig. 13, together with the results of the survey conducted in [7].

As described above, decoder F1 may be directly compared to the benchmarker of [45], since neither of them possess run-time flexibility and since they both target the same PCM having a frame length of $n_3 = 1944$ and a coding rate of $R_1 = 1/2$. As shown in Table I, decoder F1 suffers from a 63% lower processing throughput than the benchmarker of [45], but requires 17% fewer hardware resources and achieves the target BER with a 0.6 dB lower transmission energy requirement. Additionally, the throughput presented in [45] is achieved by simultaneously decoding four frames using four parallel decoder copies, which does not improve the associated processing latency. Owing to this, we may infer that the processing latency of decoder F1 is 33% lower than that of [45]. Furthermore, [45] does not detail the impact of introducing run-time flexibility upon any of its measured characteristics, whereas the architecture of decoder F1 includes an overhead that is associated with being completely generalisable to any number of QC PCMs.

Similarly, decoder F4 may be directly compared to the design of [46], which offers run-time flexibility for any PCM in the WiFi family. However, in order to maintain a fair comparison, we must consider the PCM having the same coding rate $R_2 = 2/3$ and frame length $n_2 = 1296$ as was used in [46]. The characteristics of decoder F4 when using this frame length and coding rate are indicated accordingly in Fig. 13. Table I shows that for this PCM, decoder F4 achieves a processing throughput that is 167% higher than that of [46], while the error correction performance of the two decoders is similar. However, decoder F4 suffers from a 7.3 times larger hardware resource usage than that of [46]. This cost may be attributed to the fully-flexible QC routing networks employed by the proposed architecture, which may be generalised to

TABLE I: Characteristics of decoders targeted at PCMs from the WiFi family.

Decoder	Q	Active PCM n	Active PCM R	t_i	Number of supported PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
F1	3	1944	$1/2$	72	1	64.8	9.91	128.6	7.56	2.23
F2	3	648	$1/2$	24	3	81.7	6.08	225.3	1.44	2.77
		1296	$1/2$	48		81.7	8.42	162.7	3.98	2.4
		1944	$1/2$	72		81.7	9.91	138.2	7.03	2.23
F3	1	648	$1/2$	24	4	64.1	6.08	222.1	1.46	2.77
		648	$2/3$	24		64.1	4.36	413.0	1.05	3.37
		648	$3/4$	24		64.1	3.68	550.5	0.88	3.77
		648	$5/6$	24		64.1	2.88	781.6	0.69	4.36
F4	3	648	$1/2$	24	12	140.4	6.08	196.8	1.65	2.77
		648	$2/3$	24		140.4	4.36	365.9	1.18	3.37
		648	$3/4$	24		140.4	3.68	487.7	1.00	3.77
		648	$5/6$	24		140.4	2.88	692.3	0.78	4.36
		1296	$1/2$	48		140.4	8.42	142.1	4.56	2.40
		1296	$2/3$	48		140.4	6.28	254.0	3.40	2.99
		1296	$3/4$	48		140.4	5.37	334.2	2.91	3.42
		1296	$5/6$	48		140.4	4.34	459.4	2.35	4.01
		1944	$1/2$	72		140.4	9.91	120.7	8.05	2.23
		1944	$2/3$	72		140.4	7.56	211.0	6.14	2.82
		1944	$3/4$	72		140.4	6.61	271.5	5.37	3.23
		1944	$5/6$	72		140.4	5.58	357.3	4.53	3.83
[45]	-	1944	$1/2$	-	1	77.8	10	348	11.2	2.83
[46]	-	1296	$2/3$	-	12	19.3	15	95	9.09	2.49

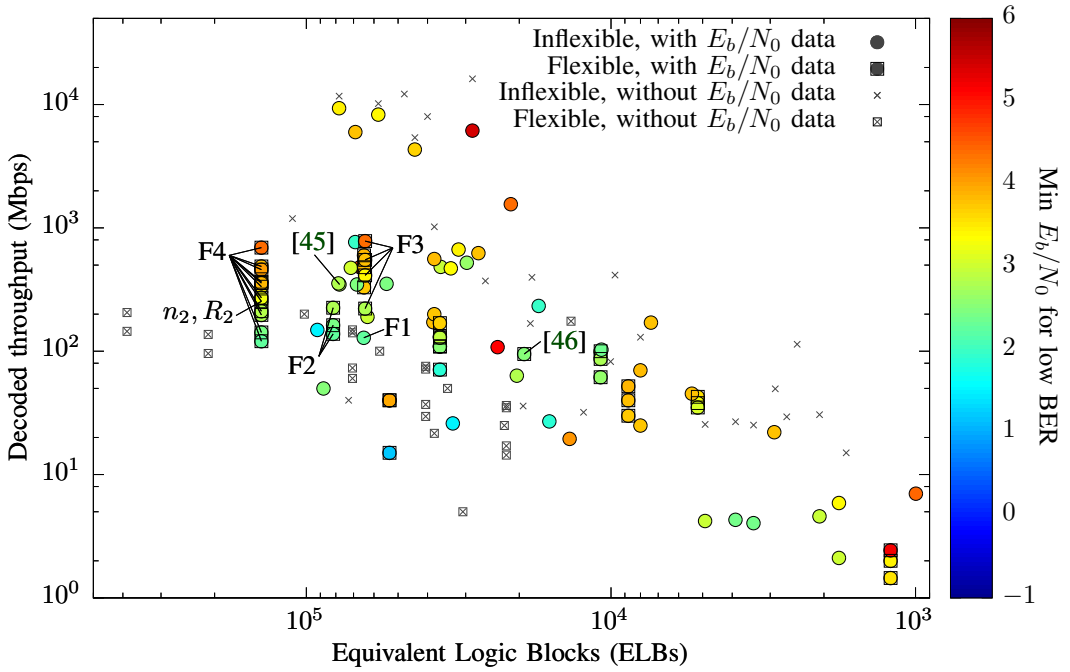


Fig. 13. Trade-offs of decoders targeted at PCMs from the WiFi family.

multiple code families with a wider variety of submatrix sizes and node degrees than those of the WiFi family, as discussed in Section III-D. By contrast, the design of [46] is specialised for the WiFi family of PCMs and exploits several of their common features in order to minimise the routing hardware resource usage.

By comparing the characteristics of decoders F2 and F3, it may be observed that supporting run-time flexibility for a selection of frame lengths n in conjunction with a single coding rate R incurs a greater hardware resource usage cost than supporting one frame length for multiple coding rates. This may be attributed to the programmable BSs, which always require Z inputs and outputs, regardless of the node-level parallelism P. Supporting multiple frame lengths requires

support for a larger number of submatrix sizes z, which causes the internal arrangement of each BS to become increasingly complex, as will be demonstrated further in Section V-C. This problem is exacerbated by the limitations imposed by FPGA synthesis, in which the programmable routing networks and logic elements must be utilised to implement hundreds of multiplexers per BS. The comparatively low hardware resource requirement of decoder F3 also demonstrates the effectiveness of the methods discussed in Section III-C and Section III-E, which reduce the potential impact of increasing the number of CNPU inputs D_C, as required when supporting higher-rate codes.

A further analysis of the proposed design is presented in Fig. 14, wherein the main characteristics of decoders F1, F2, and F4 are plotted radially with respect to the characteristics of [45], for the case of decoding the same PCM with frame length $n_3 = 1944$ and coding rate $R_1 = 1/2$. Here, superior values for each characteristic are plotted outwards on a logarithmic scale. Fig. 14 shows that each of the decoders presented in this section attain a lower processing throughput than that of [45], but with a smaller latency, greater error correction performance, and equal or greater run-time flexibility. Fig. 14 also illustrates the trade-off between flexibility and hardware resource usage, showing that decoders that support a greater number of PCMs at run-time suffer from a higher resource requirement, and vice versa.

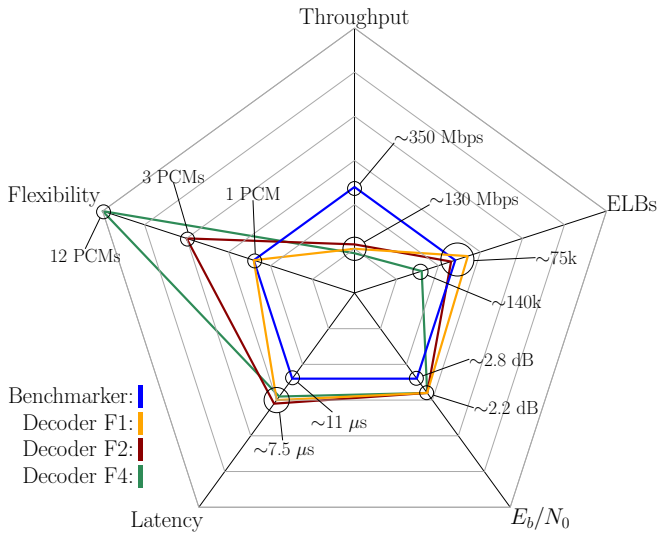


Fig. 14. Comparison of decoders targeted at PCMs from the WiFi family. Characteristics are normalised relative to the benchmarker from [45].

C. Decoders targeted at PCMs from multiple families

In addition to providing run-time flexible support for multiple PCMs from within the same LDPC code family, the architecture presented in Section III is also capable of supporting PCMs from multiple different code families. Furthermore, the automated design flow presented in Section IV ensures that the design process is no more complicated than that of one which supports codes from only a single family.

In order to demonstrate this key feature, multiple instances of the proposed decoder were again implemented and characterised using the methods described in Section V-A. Firstly, decoder S1 targets the PCMs having the lowest frame length and coding rate from each of the WiFi, WiMAX, WiGig, and WPAN families. In order to maximise the throughput, this decoder employs $Q = 1$, which gives a parallelism of $P = 42$ due to the large submatrix size of $z = 42$ in the WiGig code family. However, this high degree of parallelism results in unused hardware resources when decoding the supported WiFi, WiMAX, and WPAN PCMs, whose submatrix sizes z are 27, 24, and 21, respectively. In order to investigate the impact of this, decoder S2 targets the PCMs with the lowest frame length and coding rate from only the WiFi, WiMAX, and WPAN families. Here, the parallelism P is reduced to 27. Thirdly, decoder S3 was designed to support the PCMs from both the WiGig and WPAN families having the two highest coding rates, with a parallelism of $P = 11$. Furthermore, we implemented two additional decoders to target all 134 of the PCMs from the four families mentioned previously, namely 12 from the WiFi family, 114 from the WiMAX family, 4 from the WiGig family, and 4 from the WPAN family. More specifically, decoder S4 employs $Q = 24$, which results in a parallelism of $P = 4$, since $Z = 96$ is the maximum submatrix size among this selection of PCMs. Finally, decoder S5 supports this same set of 134 PCMs, but adopts $Q = 8$, which leads to $P = 12$.

Throughout the survey of [7], only [11] and [31] were identified as offering LDPC decoder designs having run-time flexibility for more than one family of codes. In particular, these two designs are capable of supporting full run-time flexibility over any LDPC code, regardless of structure. Note that [31] does not detail the specific PCM used to generate its results, instead providing an approximate average result for several PCMs. Furthermore, while the processing throughput measurements presented in [31] were calculated using 15 decoding iterations, the only BER results it presents were generated using 100 decoding iterations, hence they cannot be considered as part of a fair comparison. Additionally, the results presented in [11] were not generated using one of the standardised codes described previously either.

Table II and Fig. 15 compare the processing throughputs, hardware resource requirements, and error correction capabilities of the benchmarkers discussed above, as well as those for a representative subset of the PCMs supported in each of the proposed decoders S1 to S5. These results show that the proposed decoders offer similar processing throughputs to the majority of FPGA-based LDPC decoders considered in [7], while offering a very high degree of run-time flexibility, albeit with a larger hardware resource requirement which will be discussed below.

As anticipated, the reduced parallelism of decoder S2 compared to S1 leads to a reduction in the hardware resource usage, without decreasing the throughput for its supported WiFi, WiMAX or WPAN PCMs. More specifically, the ability of decoder S1 to decode a WiGig PCM in addition to those supported by decoder S2 causes it to require a greater quantity of hardware, without increasing the throughput. Meanwhile, the complementary aspects of the WiGig and WPAN PCMs can be observed in the characteristics of decoder S3, which offers a high throughput at a relatively low hardware resource

TABLE II: Characteristics of decoders targeted at PCMs from multiple families.

Decoder	Q	Active PCM family	Active PCM n	Active PCM R	t_i	Number of supported PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
S1	1	WiFi	648	$^1/_2$	24	4	91.5	6.08	215.2	1.5	2.77
		WiMAX	576	$^1/_2$	24			7.98	145.7	2.0	3.14
		WiGig	672	$^1/_2$	16			6.70	303.7	1.1	3.14
		WPAN	672	$^1/_2$	32			6.76	150.5	2.2	3.02
S2	1	WiFi	648	$^1/_2$	24	3	58.8	6.08	218.6	1.5	2.77
		WiMAX	576	$^1/_2$	24			7.98	148.1	1.9	3.14
		WPAN	672	$^1/_2$	32			6.76	152.9	2.2	3.02
S3	4	WiGig	672	$^3/_4$	64	4	44.9	4.17	129.1	2.6	3.72
		WiGig	672	$^{13}/_{16}$	64			3.23	250.1	2.0	4.30
		WPAN	672	$^3/_4$	64			3.99	135.0	2.5	3.56
		WPAN	672	$^7/_8$	64			3.11	259.7	1.9	4.12
S4	24	WiFi	648	$^5/_6$	168	134	208.0	2.88	105.8	5.1	4.36
		WiFi	1296	$^3/_4$	336			5.37	51.1	19.0	3.42
		WiFi	1944	$^1/_2$	504			9.91	18.5	52.7	2.23
		WiMAX	576	$^5/_6$	144			3.99	79.2	6.1	4.45
		WiMAX	864	$^3/_4$	216			5.46	52.1	12.4	3.75
		WiMAX	1248	$^2/_3$	312			7.64	33.1	25.1	3.29
		WiMAX	1920	$^1/_2$	480			12.26	15.5	62.1	2.61
		WiMAX	2304	$^2/_3$	576			10.24	24.7	62.2	2.94
		WiGig	672	$^1/_2$	176			6.70	27.0	12.4	3.14
		WiGig	672	$^3/_4$	176			4.17	65.1	7.7	3.72
		WPAN	672	$^5/_8$	192			4.84	42.9	9.8	3.12
		WPAN	672	$^7/_8$	192			3.11	93.4	6.3	4.12
S5	8	WiFi	648	$^5/_6$	72	134	228.9	2.88	227.1	2.4	4.36
		WiFi	1296	$^3/_4$	120			5.37	131.5	7.4	3.42
		WiFi	1944	$^1/_2$	168			9.91	50.9	19.1	2.23
		WiMAX	576	$^5/_6$	48			3.99	218.5	2.2	4.45
		WiMAX	864	$^3/_4$	72			5.46	143.7	4.5	3.75
		WiMAX	1248	$^2/_3$	120			7.64	79.1	10.5	3.29
		WiMAX	1920	$^1/_2$	168			12.26	40.6	23.6	2.61
		WiMAX	2304	$^2/_3$	192			10.24	68.1	22.5	2.94
		WiGig	672	$^1/_2$	64			6.70	68.3	4.9	3.14
		WiGig	672	$^3/_4$	64			4.17	164.7	3.1	3.72
		WPAN	672	$^5/_8$	64			4.84	118.2	3.6	3.12
		WPAN	672	$^7/_8$	64			3.11	257.6	2.3	4.12
[31]	-	NA	NA	NA	NA	-	30.6	15	5.0	NA	NA
[11]	-	NA	4095	0.82	NA	-	1.21	10	2.0	1679	3.42
		NA	4095	0.82	NA			10	1.45	2316	3.52
		NA	4095	0.94	NA			10	2.43	1584	5.19

usage.

Finally, decoders S4 and S5 offer a much higher degree of run-time flexibility, at the cost of higher hardware resource requirements. This increased hardware usage is mainly due to the increased complexity of the programmable BSs, which must handle shift values of up to $s = 95$, with 24 possible values of z, namely 19 from the WiMAX family, 3 from the WiFi family, and one each from the WPAN and WiGig families. The impact of this may be observed by comparing decoders S4 and S5, where decoder S4 has 3 times fewer NPUs than S5, but its hardware resource requirement is only 1.1 times smaller. This may be explained by the domination of the overall hardware resource requirement by the BSs, which are used in equal number in both S4 and S5. Flexible partially-parallel LDPC decoders that are designed to support codes from a variety of communications standards have an inherent requirement to employ a large quantity of highly-complex flexible routing, such as the programmable BSs proposed here. This is caused by the requirement to support a large range of submatrix sizes z with a fine granularity, which are often ill-suited to the parallelism of the decoder. For this reason, this issue has received particular attention during the design of the LDPC code for enhanced Mobile BroadBand (eMBB) data in the 3GPP 5G New Radio (NR) [47], which is required to have a much greater of flexibility than the LDPC codes of previous standards. More specifically, it has been proposed that the submatrix sizes z should be multiples of powers of two, since this would permit the use of a Banyan network [36] to alleviate some of the routing complexity, and would ensure that a level of parallelism could be chosen that is suitable for all supported submatrix sizes. These improvements would significantly reduce the dominance of the routing components on the hardware resource requirement of the proposed decoder, which would in turn increase the maximum operating

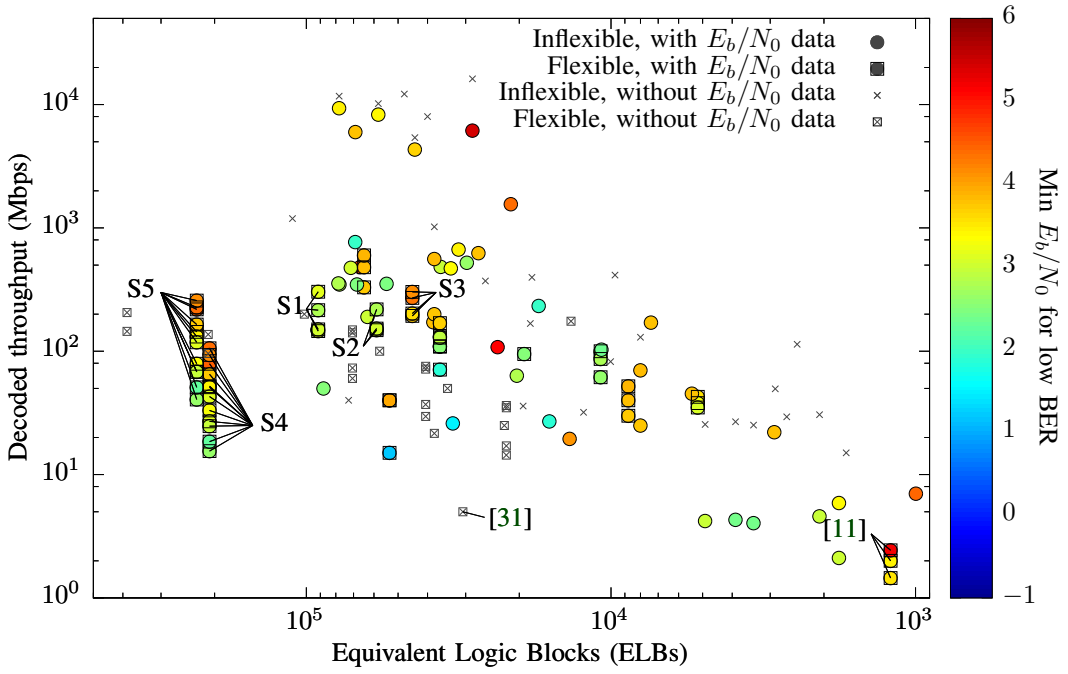


Fig. 15. Trade-offs of decoders targeted at PCMs from multiple families.

frequency and hence throughput.

The results of Table II and Fig. 15 show that all of the proposed decoders offer much higher processing throughputs than those of [11] and [31], albeit using more hardware resources. Although these previous designs offer a higher degree of run-time flexibility than the solutions proposed here, this is achieved at the cost of particularly low processing throughputs. This may be explained by the fully-serial architecture employed by the design of [11], which results in the very low hardware resource requirements and processing throughput shown here. Conversely, [31] proposes a partially-parallel implementation of a decoder which supports any regular or irregular code by compiling the PCM into a hardware-optimised form before loading it onto the FPGA. However, the cost of this degree of flexibility is a large number of clock cycles per iteration, resulting in a low processing throughput. Additionally, depending on how well the target PCMs comply with the compilation process of [31], up to 23% of its parallel processing components remain idle for some of its PCMs, resulting in a large hardware resource requirement with respect to the resultant processing throughput. Finally, the decoder of [31] also requires numerous clock cycles and manual intervention in order to load a new PCM representation onto the FPGA, at run time. This would not facilitate the dynamic switching of PCMs to adapt to changing channel conditions, for example. By contrast, the proposed architecture stores every supported PCM in ROM after compilation, which allows the active PCM to be switched within a single clock cycle. This is facilitated while achieving a higher processing throughput overall, with manageable hardware resource requirements and without sacrificing error correction performance.

VI. Conclusions

This paper has presented the design and implementation of an FPGA-based LDPC decoder having the run-time flexibility to switch between a set of QC PCMs within a single clock cycle. We have also proposed an automated design flow, which gives the proposed architecture the design-time flexibility to support any set of QC PCMs. Furthermore, this design flow automatically generates the HDL description of the decoder, which may be synthesised onto an FPGA. The implementation results presented in Section V indicate that the proposed design achieves a high level of design-time and run-time flexibility, whilst maintaining reasonable performance in terms of processing throughput, processing latency, error correction capability, and hardware resource usage.

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] 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.

[4] 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) Specifications," 2009.

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

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

[7] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Survey of FPGA-Based LDPC Decoders," *IEEE Commun. Surv. Tuts.*, vol. 18, no. 2, pp. 1098–1122, jan 2016.

[8] T. H. Tran, Y. Nagao, H. Ochi, and M. Kurosaki, "ASIC Design of 7.7 Gbps Multi-Mode LDPC Decoder for IEEE 802.11ac," in *2014 14th Int. Symp. Commun. Inform. Technol.*, Incheon, South Korea, sep 2014, pp. 259–263.

[9] M. Awais and C. Condo, *Flexible LDPC decoder architectures*. Hindawi Publishing Corporation, 2012.

[10] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible LDPC Decoder Design for Multigigabit-per-Second Applications," *IEEE Trans. Circuits Syst. I, Reg. Pap.*, vol. 57, no. 1, pp. 116–124, jan 2010.

[11] 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.

[12] 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.

[13] P. Schläfer, C. Weis, N. Wehn, and M. Alles, *Design space of flexible multigigabit LDPC decoders*. Hindawi Publishing Corporation, 2012, vol. 2012.

[14] C.-H. Liu, S.-W. Yen, C.-L. Chen, H.-C. Chang, C.-Y. Lee, Y.-S. Hsu, and S.-J. Jou, "An LDPC decoder chip based on self-routing network for IEEE 802.16e applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 684–694, 2008.

[15] T. Brack, F. Kienle, and N. Wehn, "Disclosing the LDPC code decoder design space," in *Proc. Conf. Des. Autom. Test Eur.*, no. 1, Munich, Germany, mar 2006, pp. 200–205.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] S. Myung, K. Yang, and J. Kim, "Quasi-Cyclic LDPC Codes for Fast Encoding," *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2894–2901, aug 2005.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] R. Zarubica and S. G. Wilson, "A solution for memory collision in semi-parallel FPGA-based LDPC decoder design," in *IEEE Proc. Asilomar Conf. Signals Syst. Comput.*, Pacific Grove, CA, USA, nov 2007, pp. 982–986.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] IEEE 802.11ad-2012, "Standard for Information Technology–Telecommunications and Information Exchange between Systems–Local and Metropolitan Area Networks–Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications."

[36] M. Rovini, G. Gentile, and L. Fanucci, "Multi-size circular shifting networks for decoders of structured LDPC codes," *Electron. Lett.*, vol. 43, no. 17, p. 938, 2007. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el{_}20071157

[37] J. Lin, Z. Wang, L. Li, J. Sha, and M. Gao, "Efficient shuffle network architecture and application for WiMAX LDPC decoders," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 56, no. 3, pp. 215–219, 2009.

[38] D. Oh and K. K. Parhi, "Low-complexity switch network for reconfigurable LDPC decoders," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 1, pp. 85–94, 2010.

[39] X. Chen, S. Lin, and V. Akella, "QSN - A Simple Circular-Shift Network for Reconfigurable Quasi-Cyclic LDPC Decoders," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 57, no. 10, pp. 782–786, oct 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5587879/

[40] Y. Jung, Y. Jung, S. Lee, and J. Kim, "Low-complexity multi-way and reconfigurable cyclic shift network of QC-LDPC decoder for Wi-Fi/WIMAX applications," *IEEE Trans. Consum. Electron.*, vol. 59, no. 3, pp. 467–475, 2013.

[41] 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.

[42] X. Zuo, "Fully Parallel Implementation of Timing-Error-Tolerant LDPC Decoders," Ph.D. dissertation, University of Southampton, Southampton, UK, 2016.

[43] S. Sharifi Tehrani, "Stochastic Decoding of Low-Density Parity-Check Codes," Ph.D. dissertation, McGill University, Montreal, Canada, 2011.

[44] IEEE 802.15.3c-2009, "Standard for Information Technology–Local and Metropolitan Area Networks–Specific Requirements-Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2009.

[45] 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.

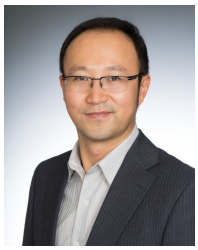
[46] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable LDPC decoder architectures for regular and irregular codes," *J. Signal Process. Syst.*, vol. 53, no. 1-2, pp. 73–88, may 2008.

[47] ZTE, "R1-1701598 Further consideration of Flexibility of LDPC Codes for NR," in *3GPP TSG RAN WG1 Meet. #88*, Athens, Greece, feb 2017.

[48] N. Bonello, S. Chen, and L. L. Hanzo, "Multilevel-Structured Low-Density Parity-Check Codes for AWGN and Rayleigh Channels," *IEEE Trans. Veh. Technol.*, vol. 59, no. 7, pp. 3311–3320, 2010.

[49] F. Guo and L. Hanzo, "Reliability-Ratio Based Weighted Bit-Flipping Decoding for Low-Density Parity Check Codes," *IEEE Electron. Lett.*, vol. 40, no. 21, pp. 1356–1358, 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.