UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES & ENGINEERING
Department of Electronics and Computer Science

Design and implementation of flexible FPGA-based LDPC decoders

by

Peter Hailes

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy

June 19, 2018

Supervisors:
Professor Robert G. Maunder,
Professor Bashir M. Al-Hashimi,
and Professor Lajos Hanzo

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES & ENGINEERING
Department of Electronics and Computer Science

Doctor of Philosophy

Design and implementation of flexible FPGA-based LDPC decoders

by Peter Hailes

Since their rediscovery in the mid-1990s, Low-Density Parity Check (LDPC) error correction decoders have been the focus of a great deal of research within the communications community. They have also become popular channel coding schemes in a plethora of diverse communications standards, as a benefit of their strong error correction performance, low-complexity computations, and their suitability to parallel hardware implementation. Meanwhile, a great deal of research effort has been invested into LDPC decoder designs that exploit the high processing speed and parallelism of Field-Programmable Gate Array (FPGA) devices, which now constitute a cost-effective alternative to Application-Specific Integrated Circuit (ASIC) platforms for LDPC decoder implementations. However, the FPGA-based LDPC decoder designs published in the open literature vary greatly in terms of design choices and performance criteria, making them a challenge to compare and even more challenging to implement.

In this thesis, we explore the key factors involved in FPGA-based LDPC decoder design and present an extensive review of the current literature, analysing and characterising the performance tradeoffs demonstrated across over 140 competing designs. From this survey, we conclude that high-performance FPGA-based LDPC decoder designs supporting the ability to dynamically alter their decoding parameters at run-time are under-represented within the state-of-the-art, despite their necessity in order to comply with many modern communications standards.

Accordingly, this thesis therefore proposes two parameterised FPGA-based LDPC decoder architectures, which both support run-time flexibility over any arbitrary set of one or more Quasi-Cyclic (QC) LDPC codes. Our first architecture adopts a traditional fixed-point message decoding algorithm, but features a variety of design optimisations which reduce the costs of supporting multiple diverse codes. Implementation results of this decoder indicate that it is capable of achieving throughputs that are higher than previous flexible FPGA-based LDPC decoders, even whilst achieving the desired level of flexibility and satisfactorily high error correction performance.

Our second decoder architecture utilises stochastic computing in order to achieve a reduced hardware resource requirement. We demonstrate the first stochastic LDPC decoder to exhibit true run-time flexibility, and detail the numerous novel optimisations that support this feature. Finally, we also propose an offline design flow, which may be used to automatically generate an optimised decoder having either of the two proposed architectures, for any chosen selection of QC codes. This added design-time flexibility greatly enhances the usefulness of the proposed architectures in real applications, facilitating easier experimental investigations of the trade-offs discussed previously.

Academic Thesis: Declaration Of Authorship

I, **Peter Hailes**,

declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

Title of thesis: **Design and implementation of flexible FPGA-based LDPC decoders**

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Either none of this work has been published before submission, or parts of this work have been published in the listed publications.

Signed: [illegible]

Date: 19/6/18

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to express my sincere thanks to my primary supervisor, Professor Rob Maunder, for all of the guidance and support he has provided over the course of this Ph.D.; for his tireless work ethic and unrelenting optimism that have always helped to keep things moving.

I would also like to thank my other supervisors, Professor Lajos Hanzo and Professor Bashir M. Al-Hashimi, for lending their considerable wisdom and experience when required. Thanks also to Lei Xu at Intel for providing the means for this Ph.D. to come about in the first place, and for offering his valuable commercial perspectives to various questions of direction or focus along the way.

Last but by no means least, I would like to express my deep gratitude to my loving wife, Alethia, whose unconditional support and gentle encouragement have been so instrumental in helping me to get this far, and who continues to be a blessing to me every day of my life.

Nomenclature

4LUT	4-input Lookup Table
6LUT	6-input Lookup Table
ALM	Adaptive Logic Module
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
BRAM	Block RAM
BS	Barrel Shifter
CLB	Configurable Logic Block
CN	Check Node
CND	Check Node Decoder
CNPU	Check Node Processing Unit
C-RAN	Centralised Radio Access Network
DC	Decoding Cycle
DCMC	Discrete-input Continuous-output Memoryless Channel
DFF	D-type Flip-Flop
DRE	Distributed Randomisation Engine
ELB	Equivalent Logic Block
EM	Edge Memory
eMBB	enhanced Mobile Broadband
EXIT	Extrinsic Information Transfer
FEC	Forward Error Correction
FER	Frame Error Rate
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HLS	High-Level Synthesis
IDS	Informed Dynamic Scheduling
IM	Internal Memory
IMMB	Intrinsic Message Memory Bank

JKFF	JK-type Flip-Flop
LAB	Logic Array Block
LBP	Layered Belief Propagation
LDPC	Low-Density Parity Check
LE	Logic Element
LFSR	Linear Feedback Shift Register
LLR	Logarithmic-Likelihood Ratio
LUT	Lookup Table
MSA	Min-Sum Algorithm
NDS	Noise-Dependent Scaling
NPU	Node Processing Unit
NR	New Radio
PCM	Parity-Check Matrix
PEG	Progressive Edge Growth
PU	Processing Unit
QC	Quasi-Cyclic
SNR	Signal to Noise Ratio
SPA	Sum-Product Algorithm
VN	Variable Node
VND	Variable Node Decoder
VNPU	Variable Node Processing Unit

Commonly used symbols

	Forward Error Correction
$\mathbf{m}$	vector of original information bits
$\mathbf{c}$	vector of transmitted codeword bits
$\mathbf{x}$	vector of transmitted BPSK symbols
E_s	transmission energy per symbol
E_b	transmission energy per bit
N_0	noise power spectral density
$\mathbf{y}$	vector of received BPSK symbols
$\tilde{\mathbf{c}}$	vector of received codeword LLRs
$\tilde{\mathbf{c}}'$	vector of decoded codeword LLRs
$\hat{\mathbf{c}}$	vector of decoded codeword bits
$\hat{\mathbf{m}}$	vector of recovered information bits
$\mathbf{s}$	vector of syndrome bits
	Parity-Check Matrices
k	number of information bits for a particular PCM
m	number of parity bits (rows) in a particular PCM
n	number of codeword bits (columns) in a particular PCM
R	coding rate
$\mathbf{G}$	generator matrix
$\mathbf{H}$	parity-check matrix
d_v	variable node degree for a particular PCM
d_c	check node degree for a particular PCM
	Quasi-Cyclic Parity-Check Matrices
$\mathbf{H}_b$	quasi-cyclic base parity-check matrix
m_b	number of block-rows within a particular QC PCM
n_b	number of block-columns within a particular QC PCM
z	quasi-cyclic submatrix expansion factor in a particular QC PCM
s	quasi-cyclic submatrix shift amount

	Parity-Check Matrix sets
K	maximum number of information bits in a set of PCMs
M	maximum number of parity bits (rows) in a set of PCMs
N	maximum number of codeword bits (columns) in a set of PCMs
M_B	maximum number of block-rows in a set of QC PCMs
N_B	maximum number of block-columns in a set of PCMs
D_V	maximum variable node degree in a set of PCMs
D_C	maximum check node degree in a set of PCMs
Z	maximum quasi-cyclic submatrix expansion factor in a set of QC PCMs
	LDPC decoding
α	VN to CN messages
β	CN to VN messages
F_{VN}	VN update function
F_{CN}	CN update function
	LDPC decoders
ψ	decoder degree of parallelism
ψ_c	decoder degree of CNPU parallelism
Q	decoder parallelism reduction factor
q	number of subblocks per block for a particular PCM
W	bit width of binary messages
F	employment of linked CNPUs in a decoder
l	employment of linked CNPUs for a particular PCM
f	clock frequency
f_{max}	maximum clock frequency
t_i	clock cycles required per iteration
t_{DC}	clock cycles required per DC
T_a	average number of iterations/DCs required per frame
	NPU architecture
N_{sub}	number of 2-input 1-output subnodes in an NPU
Λ	maximum number of NPU inputs/outputs
λ_a	current active number of NPU inputs/outputs
$\lambda_n^{y,x}$	minimum value of λ_a to use a node for term x at stage y
$\lambda_c^{y,x}$	minimum value of λ_a for term x at stage y to be connected
$S^{y,x}$	intermediate signal x at stage y of dual-tree NPU
L_E	ring buffer length for each EM
L_I	ring buffer length for each IM

List of publications

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

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Flexible FPGA-Based Quasi-Cyclic LDPC Decoder," in *IEEE Access*, vol. 5, pp. 20965–20984, mar 2017.

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Hardware-efficient Node Processing Unit Architectures for Flexible LDPC Decoder Implementations," in *IEEE Trans. Circuits Syst. II, Exp. Briefs.*

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Flexible Stochastic LDPC Decoder Architecture," in preparation for *IEEE Access.*

A. Li, **P. Hailes**, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "1.5 Gbit/s FPGA implementation of a fully-parallel turbo decoder designed for mission-critical machine-type communication applications," in *IEEE Access*, vol. 4, pp. 5452–5473, aug. 2016.

S. Shao, **P. Hailes**, R. G. Maunder, T. Wang, J.-Y. Wu, B. M. Al-Hashimi, and L. Hanzo, "Survey of Turbo, LDPC and Polar Decoder ASIC Implementations," in preparation for *IEEE Commun. Surveys Tuts.*.

Chapter 1

Introduction

Low-Density Parity Check (LDPC) codes represent a class of Forward Error Correction (FEC) codes that may be employed for providing near-theoretical-limit error correction of transmission errors in communication systems. They were first proposed by Gallager in 1962 [1], but were considered to be too complex for practical simulation and implementation at the time of their conception, hence they were left largely untouched for decades. Subsequently, and perhaps partially motivated by the fact that the turbo codes patented during the early 1990s attracted a license-fee, in 1996 LDPC codes were rediscovered by Mackay and Neal [2]. They have since enjoyed a renaissance, becoming the focus of much research within the communications community and demonstrating the potential for excellent practical performance. Given the increased computing power available today, LDPC codes have been adopted in many modern communications standards, including IEEE 802.11n [3], IEEE 802.16e [4], DVB-S2 [5], CCSDS [6], ITU G.hn [7], and enhanced Mobile Broadband (eMBB) data in the forthcoming 3GPP 5G New Radio (NR) [8].

LDPC codes benefit from a number of appealing features that make them very attractive for implementation. The LDPC decoding algorithm can be implemented using low-complexity calculations, resulting in a relatively low design and implementation cost for the processing hardware. The inherent parallelism of the LDPC decoding algorithm also maps intuitively to distributed parallel computation units for high-throughput low-latency implementations. Furthermore, like turbo codes, LDPC codes are decoded iteratively, achieving an error correction performance that is close to the theoretical limit when decoding messages that have large block lengths [9]. However, in contrast to turbo codes, there is a wide variety of possible algorithms and levels of parallelisation that may be considered for the design of LDPC decoders, presenting designers with a range of options that may be selected from in order to achieve the desired characteristics.

However, while the design of the individual processing components is relatively simple, the design of a complete LDPC decoder is subject to a complex interplay between a number of system characteristics, namely the processing throughput, processing latency, hardware resource requirements, error correction capability, processing energy efficiency, bandwidth efficiency and flexibility. These characteristics depend on a number of system parameters, namely the architecture, the LDPC code employed, the algorithm used, and the number of decoding iterations. The complex nature of the relationships between all of these parameters and characteristics makes the process of designing and implementing an efficient LDPC decoder a very challenging and time-consuming task.

During this process, it is often desirable for a designer to have the capability to implement design prototypes at various stages, in order to physcially measure their achievable performance in realistic conditions. This may be readily achieved through the use of a Field-Programmable Gate Array (FPGA) device, which facilitates rapid prototyping and fast parallel logic processing. These advantages are evident from the sheer number of published FPGA-based LDPC decoder designs that exist in the open literature, which will be compared later in this thesis. Furthermore, the decoding techniques and implementation-oriented research presented alongside these designs has been of significant benefit to the wider communications research community [10–15]. In particular, the implementational characteristics of these FPGA-based LDPC decoders are increasingly informing the holistic design of communication systems.

Another very practical use of FPGA-based LDPC decoders is in the research environment, where the very fast and highly-parallel logic resources available on an FPGA are very useful for measuring the Bit Error Rate (BER) performance of various codes [16]. More explicitly, simulations that would take days on a computer can be completed in only hours when using a custom FPGA implementation [17]. However, the majority of published FPGA-based LDPC decoder designs are tailored specifically to one code. This means that they would require a great deal of extra design work in order to modify the architecture every time a new LDPC code is to be investigated. This design work would then have to be re-synthesised to produce another FPGA configuration file, which would then have to be downloaded onto the FPGA before any decoding of the new code could commence.

In addition to their suitability for prototyping and research measurements, FPGAs constitute a viable alternative to Application-Specific Integrated Circuits (ASICs) for the LDPC decoders of small-production-run communication devices, while their programmability has made them attractive for software-defined radios. In these cases, a decoder having the ability to switch between different LDPC codes at run-time is necessary to fully support all of the codes within many standards [3–5], in which the coding rate and frame length may be dynamically adjusted in order to suit the prevailing channel conditions [18]. Furthermore, decoders having the ability to support LDPC codes from multiple families may be used to facilitate Centralised Radio Access Networks (C-RAN).

Here, LDPC decoding for multiple remote basestations (often serving multiple different standards) is performed by cloud-based digital function units (DFUs), allowing more efficient use of resources, and effectively combining multiple independent cells together to reduce inter-cell interference. When each LDPC decoder in these DFUs has the ability to process LDPC codes from any of the supported standards, the flexibility (and hence efficiency through pooling resources) of the associated C-RAN is increased.

It is therefore desirable to have a generalised FPGA-based LDPC decoder architecture capable of supporting multiple LDPC codes, from within a single family or from multiple families, with the ability to rapidly switch the code that is currently being decoded at run-time. Some prior research into run-time flexible FPGA-based LDPC decoder designs has been performed, however they are all either constrained to a very limited set of related codes [19–24] or achieve a very low processing throughput [25, 26]. It is also desirable that decoder instances having this architecture could be generated automatically, based on the selection of which LDPC codes they are expected to support. This enables a more comprehensive design-time parsing and analysis of the features of the target code set, facilitating greater automated optimisations to the decoder design, without requiring any prior knowledge of its operation. This design flow would hence grant automated design-time flexibility to a decoder design already having run-time flexibility, greatly extending its usefulness to a variety of parties. Again, some previous works have proposed similar automated design flows for the generation of LDPC decoders [27–31], however these do not produce designs with the run-time flexibility described above.

This work has three main aims. Firstly, we provide a comprehensive comparison of all of the implementations of FPGA-based LDPC decoders that we are aware of. In doing so, we observe, explain, and characterise the fundamental trade-offs and relationships between the associated parameters and characteristics of these decoders. Secondly, we use these findings to propose two new generalised architectures for FPGA-based decoders that are capable of supporting any set of one or more quasi-cyclic LDPC codes, with the ability to switch from one code to another in a single clock cycle. Thirdly, to improve the usefulness of these architectures in real-world situations, we present an automated design flow which generates the full specific hardware description of decoders having the proposed architecture, based on a chosen selection of codes.

1.1 Structure of the thesis

This thesis is structured as follows.

In **Chapter 2**, we present a detailed tutorial-style introduction to LDPC codes and the associated encoding and decoding processes. In doing so, we introduce many of the implementational factors that affect the design and performance of FPGA-based LDPC decoders, and provide a brief explanation of the key features of FPGAs. We also introduce the relatively recent concept of stochastic LDPC code decoding, discussing the various ways in which it differs from traditional fixed-point decoding, as well as the associated advantages and disadvantages.

Following this, **Chapter 3** presents a comprehensive survey of published FPGA-based LDPC decoders. This facilitates a detailed discussion of the selectable parameters and measurable characteristics which must be considered when designing an FPGA-based LDPC decoder, before characterising the trade-offs and relationships between them. In this chapter we also use the experience gained as a result of conducting this survey to present a set of recommendations for future authors of FPGA-based LDPC decoder implementation publications to follow, in order to simplify the process of comparing designs in future.

Chapter 4 then presents our generalised flexible fixed-point FPGA-based LDPC decoder architecture. We firstly explain our reasons for targeting quasi-cyclic LDPC codes, and present the associated advantages. We then describe each part of the decoder architecture in detail, including the degree of parallelism, the memory management scheme, the flexible datapath, the node processing units, and the control unit. In doing so we present several optimisations which improve the implementation characteristics of decoders having the proposed architecture, without making it specific to any particular code family. Implementation results presented at the end of this chapter indicate that certain parametrisations of the proposed architecture are capable of achieving throughputs that are greater than any previous flexible FPGA-based LDPC decoder. Meanwhile, instances targeting larger sets of supported codes maintain throughputs that are comparable to the majority of other decoders, but at the expense of higher hardware resource requirements.

Motivated by the findings of Chapter 4, the alternative generalised flexible FPGA-based LDPC decoder architecture presented in **Chapter 5** employs stochastic computations, in an effort to achieve a smaller hardware resource consumption than its fixed-point counterpart. In order to achieve the desired level of flexibility, this architecture employs a structure and decoding algorithm that have not been demonstrated by stochastic decoders previously. Consequently, this architecture also represents the first stochastic LDPC decoder to support run-time flexibility over multiple code sets. Chapter 5 describes the ways in which the design of the stochastic decoder architecture facilitates these aspects, whilst also briefly describing the ways in which it is similar to previous stochastic decoders and the architecture of Chapter 4. Finally, the performance of decoders having this architecture are characterised, and its strengths and weaknesses are described and explained.

Following this, an explanation of our automated design flow is presented in **Chapter 6**. This includes a description of the ways in which it parses the target set of supported LDPC codes to extract and generate the range of decoder parameters required by the architectures of Chapters 4 and 5, and also a brief description of the methods employed to automatically generate the hardware description language design files. Finally, Chapter 6 also provides a description of the ways in which some of the most highly-optimised elements of the proposed architectures are carefully automatically generated at design-time, in order to increase the achievable performance characteristics.

Finally, **Chapter 7** summarises the main findings of this thesis and provides some concluding remarks, alongside suggestions for possible avenues of future work which may improve the proposed designs further.

1.2 Novel contributions of the thesis

In summary, the novel contributions presented in this thesis are:

- an extensive survey of over 140 published FPGA-based LDPC decoders, including novel methods of measuring and comparing performance characteristics across diverse designs and architectures;
- a detailed analysis and graphical depiction of the trade-offs inherent to FPGA-based LDPC decoder design;
- a generalised architecture of a parametrisable fixed-point FPGA-based LDPC decoder, having both design-time and single-clock-cycle run-time flexibility over an arbitrary set of supported LDPC codes;
- parametrisations of the proposed fixed-point architecture exhibiting the highest throughputs achieved by flexible FPGA-based LDPC decoders;
- the first stochastic LDPC decoder to support a partially-parallel architecture, including adaptations to the traditional stochastic processing elements to make them compliant with processing multiple time-multiplexed nodes;
- a novel variation of the traditional multicast layered belief propagation decoding schedule that is compatible with stochastic processing units;
- the first stochastic LDPC decoder supporting run-time flexibility over more than one code family;
- a generalised algorithm for constructing a hardware-efficient and implementation-agnostic node processing unit, along with a novel method of adding run-time flexibility over any set of supported numbers of inputs and outputs;
- a novel design flow that automatically generates optimised hardware descriptions of decoders having either of the proposed architectures, targeted at any desired set of quasi-cyclic LDPC codes, and with a customisable degree of parallelism.

Chapter 2

Background

Before commencing the presentation of our survey results and novel designs in later chapters, this chapter presents a tutorial on FPGA-based LDPC decoders. In doing so, we introduce the concepts and notations used throughout the rest of this work. Section 2.1 commences by discussing Forward Error Correction (FEC) in general terms, before LDPC codes are introduced in Section 2.2. This is followed by a discussion of how LDPC codes may be designed and decoded in Sections 2.3 and 2.4, respectively. The practicalities of LDPC decoder implementations are then discussed in Section 2.5, which is followed by a brief introduction to FPGAs in Section 2.6. Finally, Section 2.7 provides an introduction and explanation of stochastic LDPC decoding, which is necessary to understand the design decisions motivating the novel architecture proposed in Chapter 5.

2.1 Forward error correction

Figure 2.1 shows a schematic of a simplified communications system, where the information message word $\mathbf{m} = \{m_i\}_{i=1}^{k}$ is a vector of k bits, which is FEC encoded in order to obtain the codeword $\mathbf{c} = \{c_j\}_{j=1}^{n}$, which is a vector of $n > k$ bits. The FEC encoder converts the k-bit message word $\mathbf{m}$ into the n-bit codeword $\mathbf{c}$ by adding $m = n - k$ parity bits to the message. The ratio of the message length k to the total codeword length n is referred to as the coding rate R,

$$R = \frac{k}{n} = \frac{n-m}{n}. \tag{2.1}$$

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

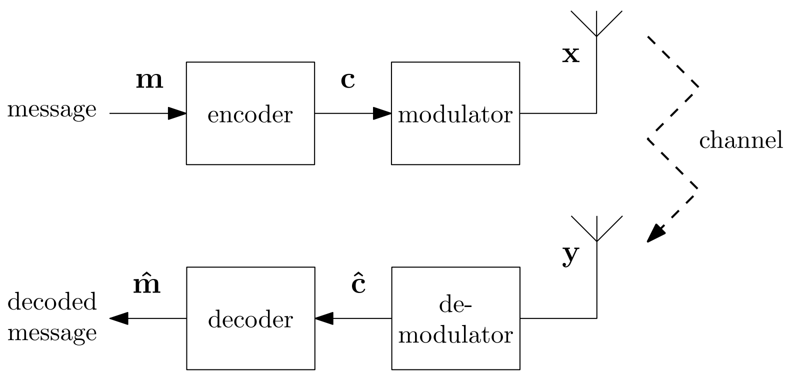


FIGURE 2.1: A communications system

Various modulation schemes can be used for modulating the codeword $\mathbf{c}$ onto the channel. As we shall show in Section 3.3.2.4, Binary Phase-Shift Keying (BPSK) modulation is assumed for nearly all FPGA-based LDPC decoder research. For this reason, we also assume the employment of BPSK modulation throughout this tutorial discussion. It is important to note however, that BPSK is a very simple modulation scheme, which is rarely employed alone in practical communication schemes. Therefore, considering BPSK modulation exclusively during the design phase could result in an LDPC decoder which does not necessarily work satisfactorily in practical systems, where higher-order modulation schemes are employed. Note that our analysis in Section 3.3 will take the specific modulation scheme that was used into consideration, when comparing the error correction performance of various FPGA-based LDPC decoders.

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

$$y_j = x_j + \mathcal{CN}(0, N_0), \tag{2.2}$$

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

$$\frac{E_b}{N_0} = \frac{1}{R} \times \frac{E_s}{N_0}. \tag{2.3}$$

The corruption imposed by the channel causes $\mathbf{y}$ to differ from $\mathbf{x}$ in an unpredictable manner, potentially resulting in the demodulation of a perturbed received codeword $\hat{\mathbf{c}}$, potentially including some transmission errors. The decoder of Figure 2.1 is employed to recover the message word $\hat{\mathbf{m}}$, and without this there would be no way of correcting (or even detecting the presence of) these errors.

The error correction capability of a FEC decoder is affected by the form of the information provided by the demodulator. Rather than using hard decisions to convert received symbols into demodulated bits, superior error correction capability can be obtained if the demodulator provides soft decisions. These may be represented in the form of probabilities, but are are far more commonly expressed using the *Logarithmic-Likelihood Ratio* (LLR) [32]. The sign of an LLR (positive or negative) expresses *what* the most likely value for the corresponding bit is (0 or 1, respectively). Meanwhile, the magnitude of an LLR expresses *how* likely this value is, where 0 represents complete uncertainty and ∞ represents absolute certainty. The value of an LLR is calculated as

$$\tilde{c}_i = \log \frac{P(c_i = 0 \mid y_i)}{P(c_i = 1 \mid y_i)}, \tag{2.4}$$

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

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

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

$$\tilde{c}_i = 4 \times R \times \frac{E_b}{N_0} \times \mathrm{Re}(y_i). \tag{2.5}$$

2.2 LDPC codes

This section provides an introduction to LDPC codes, commencing with their structure and the encoding process in Section 2.2.1. Following this, the decoder's Parity-Check Matrix (PCM) is introduced in Section 2.2.2, together with its graphical representation using factor graphs in Section 2.2.3.

2.2.1 Encoding

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

As described previously, LDPC codes permit the correction of transmission errors by supplementing each k-bit message word with m parity bits in order to produce an n-bit codeword, where $n = k + m$ [33]. Codes which include the k bits of the message word within the n bits of the codeword are referred to as systematic, while non-systematic codes have codewords which do not directly contain the original message bits. There are 2^k possible permutations of the k-bit message word, each of which is mapped by the LDPC encoder to a corresponding one of 2^k legitimate codeword permutations. The error correction capability of the LDPC code depends on the minimum Hamming distance between any pair of these 2^k legitimate codeword permutations. Naturally high minimum distances are preferred, since these make it unlikely for a legitimate codeword to be transformed into another by the distortion introduced during transmission.

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

$$\mathbf{c} = [c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}], \tag{2.6}$$

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

$$c_7 = c_4 \oplus c_6 \tag{2.7a}$$

$$c_8 = c_1 \oplus c_3 \oplus c_5 \oplus c_6 \tag{2.7b}$$

$$c_9 = c_2 \oplus c_5 \tag{2.7c}$$

$$c_{10} = c_1 \oplus c_2 \oplus c_6. \tag{2.7d}$$

The design of an LDPC code's parity check equations is subject to many complex factors, as will be briefly described in Section 2.3. Using these equations, a $(k \times n)$-element generator matrix $\mathbf{G}$ can be constructed to efficiently describe the encoding process. In a systematic code, $\mathbf{G}$ may adopt the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_k & \mathbf{A} \end{bmatrix}, \tag{2.8}$$

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

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

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

2.2.2 Parity-check matrix

In the decoder, the parity checks are used to detect the presence of transmission errors in the received codeword $\hat{\mathbf{c}}$. Since all of the codeword bits involved in a parity check (including the parity bit itself) should have a modulo-2 summation of 0, Equations (2.7a)–(2.7d) can be re-written as follows:

$$c_4 \oplus c_6 \oplus c_7 = 0 \tag{2.10a}$$

$$c_1 \oplus c_3 \oplus c_4 \oplus c_6 \oplus c_8 = 0 \tag{2.10b}$$

$$c_2 \oplus c_5 \oplus c_9 = 0 \tag{2.10c}$$

$$c_1 \oplus c_2 \oplus c_6 \oplus c_{10} = 0. \tag{2.10d}$$

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

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}^T & \mathbf{I}_m \end{bmatrix}. \tag{2.11}$$

Continuing our example from above, we have

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

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

Note however that an LDPC $\mathbf{H}$ matrix of the form shown in (2.12) is very unusual in practice. As it will be explained in Section 2.4, the decoder's error correction ability is dictated by the number of non-zero entries in each row or column, which is referred to as its *weight*. More specifically, columns with a weight of 1 can result in the decoder being unable to correct some transmission errors. This can be avoided by modifying the PCM $\mathbf{H}$ using elementary row operations (modulo-2 additions and swaps). In the case of the above example, this may lead to:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{2.13}$$

This modified $\mathbf{H}$ avoids any weight-1 columns, while still checking the same distribution of parity bits that was added to codewords by the generator matrix $\mathbf{G}$ of (2.9). Note however that this toy-example PCM is still unusual for a realistic LDPC code. Specifically, the PCM used in LDPC decoding should be sparse, containing far fewer non-zero entries than 0s. Clearly, the $\mathbf{H}$ of (2.13) does not satisfy this constraint, owing to its codeword length of $n = 10$, which is very short compared to practical LDPC codewords, which tend to be hundreds or even thousands of bits long.

Owing to its significance in the decoding process, the PCM $\mathbf{H}$ is commonly used to define a particular LDPC code design. As discussed later in Section 2.3, creating a $\mathbf{H}$ matrix that achieves a strong error correction capability is a complex task, so this is usually the first aspect of the code to be designed. Following this, the generator matrix $\mathbf{G}$ can be derived from $\mathbf{H}$, by following the reverse of the process described above.

2.2.3 Factor graphs

The PCM $\mathbf{H}$ can also be visualised graphically using a factor graph, which is also known as a *Tanner graph* [34]. This is exemplified in Figure 2.2 for the PCM of (2.13). A factor graph is comprised of two sets of connected nodes, namely n Variable Nodes (VNs) for representing the columns of $\mathbf{H}$ and m Check Nodes (CNs) for representing the rows.

The connections $\tilde{c}_j$ above each VN in Figure 2.2 pertain to the channel messages (i.e. LLRs or probabilities) associated with the n codeword bits of $\hat{\mathbf{c}}$. An edge connects the j-th VN V_j to the i-th CN C_i if there is a non-zero element in the j-th column and i-th row of $\mathbf{H}$, $H_{ij} = 1$. To illustrate this, all of the edges that are connected to the 1st CN C_1 in Figure 2.2 are shown with thicker lines. These edges are connected to the 1st, 4th,

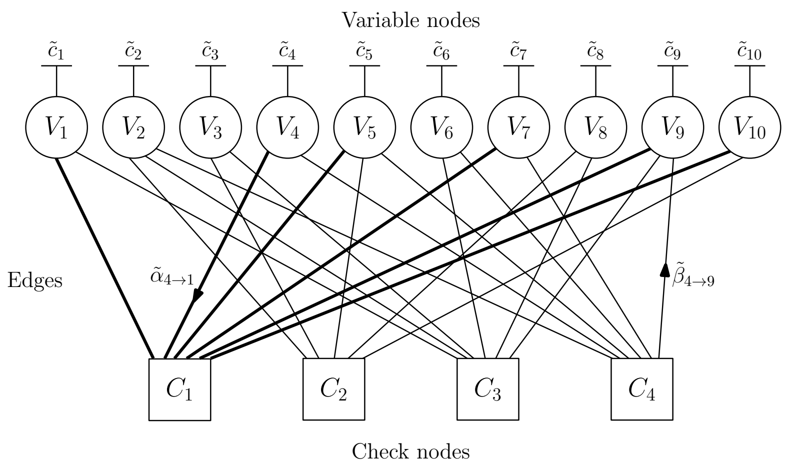


Figure 2.2: A factor graph for an example LDPC code

5th, 9th, and 10th VNs, in accordance to the position of the 1s in the top row of $\mathbf{H}$ in (2.13).

The *degree* of a node is defined as the number of other nodes that it is connected to and is equal to the corresponding row or column weight in $\mathbf{H}$. The degree of the CNs d_c and the degree of the VNs d_v are important parameters in an LDPC code. If all CNs have the same degree d_c and all VNs have the same degree d_v, the LDPC code is said to be *regular*. If either value varies from node to node, the code is said to be *irregular* and d_c and d_v can be expressed as the average degree over all nodes. For example, the factor graph of Figure 2.2 is irregular with $d_c = 5.75$ and $d_v = 2.3$. In any case, the number of 1s in the PCM $\mathbf{H}$ must be the same regardless, whether it is viewed row-by-row or column-by-column, giving $d_c \times m = d_v \times n$, with $d_v = d_c \times (1 - R)$.

2.3 LDPC code construction

In addition to the size of the factor graph and the degrees of its nodes, the position of the edges within the factor graph also has a significant impact on the associated error correction performance, as well as upon the decoding complexity. Some of the main objectives when designing the PCM $\mathbf{H}$ is to avoid creating stopping sets [35] and short cycles [36] in the corresponding factor graph, which are associated with an eroded error correction performance. A number of techniques have been proposed for placing edges within the factor graph have been proposed, as summarised in the following subsections.

2.3.1 Random codes

Unstructured randomly-designed codes potentially achieve the best LDPC error correction performance, owing to the maximised degree of freedom that is afforded when placing edges in this manner [37]. However, this is achieved at the cost of having to implement complex unstructured routing or memory lookup tables, in order to exchange messages between the VNs and CNs. A straightforward recursive algorithm for creating unstructured PCMs of this form involves placing a 1 at a random unfilled location in $\mathbf{H}$, then checking to see whether doing so has violated any design constraints, such as the maximum node degrees, stopping sets or cycle lengths. If the placement is valid, the algorithm will continue and repeat the process for the next randomly placed 1. This is repeated until the desired number of edges have been positioned. If a randomly placed 1 is not valid, then it will be rejected and a new location will be tried instead. This algorithm is conceptually very simple, but whether the process can successfully complete and how quickly is unpredictable.

2.3.2 Pseudorandom codes

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

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

2.3.3 Quasi-cyclic codes

The PCM $\mathbf{H}$ of a QC code is semi-structured, based on an upper matrix $\mathbf{H}_b$ of elements which each represent a square submatrix of size $z \times z$ [38]. If a particular element in $\mathbf{H}_b$ has a value of -1, then the corresponding submatrix of $\mathbf{H}$ is a null matrix. Otherwise, the submatrix is an identity matrix, which has been cyclically shifted a number of times according to the corresponding value in the upper matrix [39]. Adopting this structure provides a range of benefits for LDPC decoders without incurring a significant sacrifice in error correction performance, and hence QC LDPC codes are employed by a number of communications standards, including DVB-S2 [5], IEEE 802.11 (WiFi) [3] and IEEE 802.16 (Mobile WiMAX) [4]. Owing to this wide-spread adoption, the decoder architectures described in Chapters 4 and 5 target any LDPC code having a QC structure, in order to take advantage of the associated benefits. A full discussion of the nature and causes of these benefits is hence presented in Section 4.2.1.

2.3.4 Repeat-accumulate codes

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

2.3.5 Progressive edge growth algorithm

Whilst not a code structure itself, the Progressive Edge Growth (PEG) algorithm [40] is an important technique for constructing codes having an excellent error correction performance. The operation of the PEG algorithm is VN-centric, focusing on each VN in turn in order to place edges. The algorithm repeatedly constructs a set of CNs as candidates for the VN to connect to. From this set, the subset of nodes having the lowest degree is extracted and one of these is randomly selected. This approach results in LDPC codes that have approximately regular degree distributions.

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

2.4 LDPC decoding

LDPC codes are typically decoded using a belief propagation (BP) algorithm in which messages are iteratively passed in both directions along the edges between connected nodes [41]. For example, Figure 2.2 illustrates a message $\alpha_{4\to1}$ sent from the 4th VN V_4 to the 1st CN C_1, while the message $\beta_{4\to9}$ is sent from the 4th CN C_4 to the 9th VN V_9. The messages provided as inputs to a node are processed by activating that node, causing it to create new output messages that are sent back to the nodes it is connected to. Thus the processing of the LDPC decoder is delegated to the many individual calculations performed by the individual nodes, rather than being a single monolithic global equation. An important facet of the belief propagation algorithm is that any message sent to a particular node does not depend on the message received from that node. For example, CN C_2 is connected to VNs V_2, V_3, V_5, V_8, and V_{10}; however, the message $\beta_{2\to5}$ it sends to V_5 will be calculated based only on the messages it has received from V_2, V_3, V_8, and V_{10}.

Nodes are activated in an order determined by the LDPC decoder's *schedule*. This has a significant effect upon the LDPC decoder's error correction capability, as well as on its other characteristics. Many different schedules exist, and the most common options will be outlined in Section 2.4.1. Following this, variations of the specific calculations performed within CNs and VNs will be presented in Sections 2.4.2 and 2.4.3 respectively.

2.4.1 Scheduling

The schedule of the LDPC decoding process determines the order in which VNs and CNs are processed, as well as whether multiple nodes are processed in parallel. Many scheduling variations exist, but the three most common schedules are described here, namely flooding [42], Layered Belief Propagation (LBP) [43], and Informed Dynamic Scheduling (IDS) [44].

2.4.1.1 Flooding

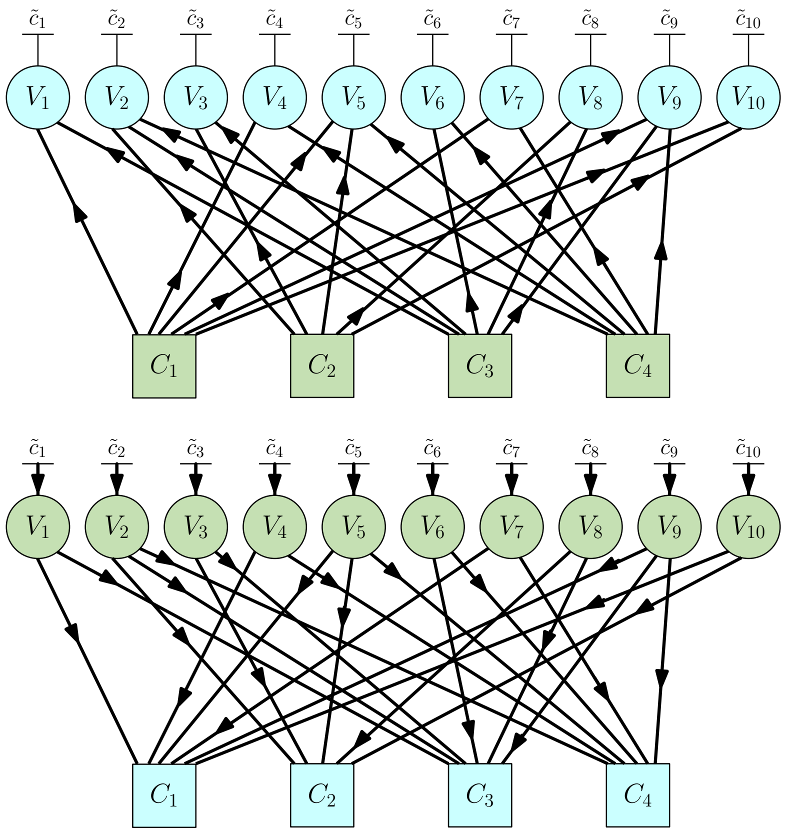


FIGURE 2.3: An example of the flooding schedule

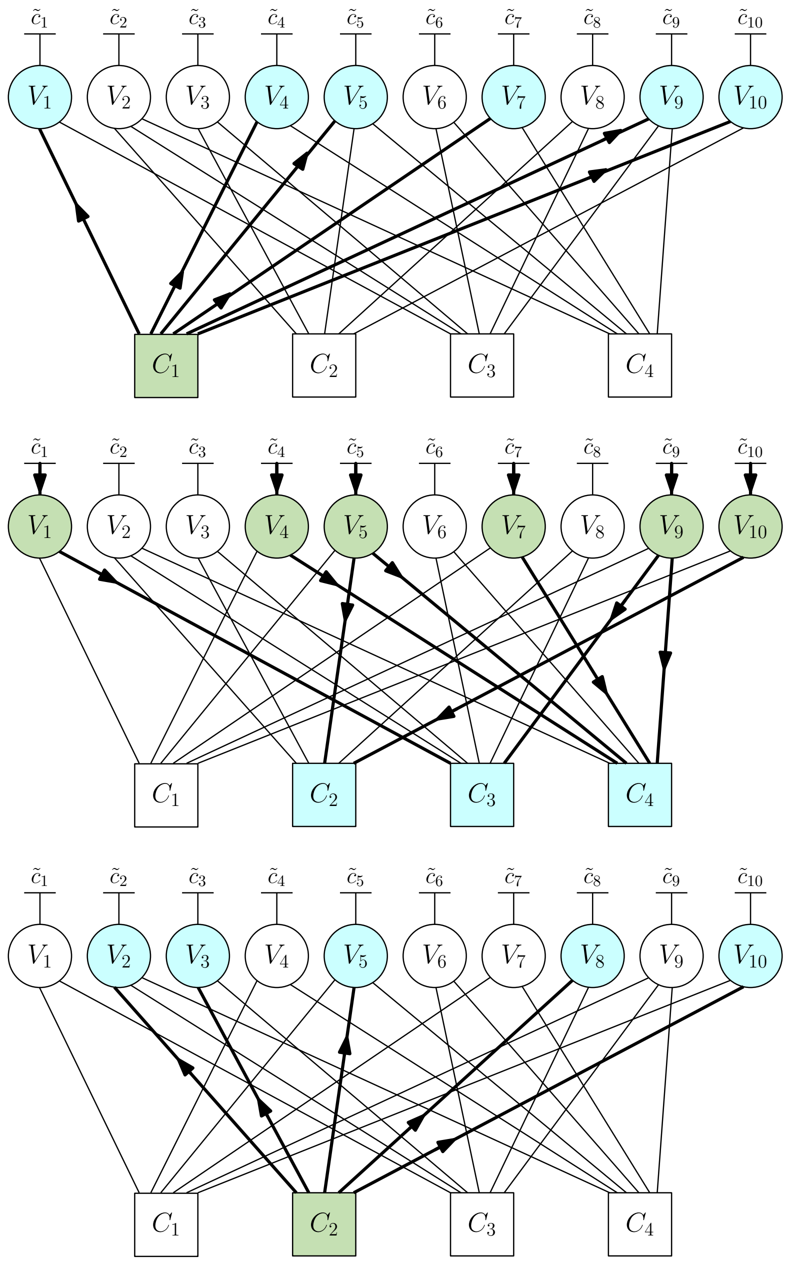
Flooding is perhaps the most conceptually simple LDPC decoding schedule. Here, the factor graph is processed in an iterative manner, where each iteration comprises the simultaneous activation of all CNs, followed by the simultaneous activation of all VNs [41]. An example of this schedule is depicted in Figure 2.3. It can be seen that at first the CNs C_1–C_4 (shown in green) calculate their messages $\beta_{j\to i}$, which are then sent along every edge (in bold) to every receiving VN (shown in blue). In the second half-iteration, the VNs are shown in green to indicate that they are performing calculations and sending messages $\alpha_{i\to j}$, while the CNs are shown in blue to indicate that they are receiving messages.

2.4.1.2 Layered belief propagation

While LBP [43] also operates in an iterative manner, within each iteration it breaks the factor graph up into a series of *layers*, which it processes sequentially [29]. Note that the following explanation focuses on a row-centric interpretation, which is the most common way of implementing LBP [45], though column-centric representations also exist [46]. Each layer comprises a number ϕ of check-nodes, where $1 \leq \phi \ll m$. The value of ϕ must be chosen such that no column within the layer has a column-weight that is greater than 1 (i.e. no VN connects to more than one CN within a single layer) [47]. The processing of each layer begins with the simultaneous activation of all of its CNs. Once the CNs have been activated, all of their connected VNs are then activated, before moving on to the next layer. Once every layer has been processed, the iteration is complete.

This process is exemplified in Figure 2.4, for a layer size of $\phi = 1$ CN. Here, the LBP decoding iteration begins with the activation of the first layer, which is comprised of CN C_1. Once it has been activated, C_1 then sends messages $\beta_{1\to j}$ to each of its connected VNs V_j, $j \in \{1, 4, 5, 7, 9, 10\}$. Each of these VNs are then activated, sending new messages $\alpha_{j\to i}$ to each of their connected CNs C_i, except C_1. Following this, the next layer (containing C_2) may be activated, allowing it to make use of the new information received from V_5 and V_{10} alongside the information previously received from its other connected VNs. This process continues until every layer has been activated, which then marks the end of one decoding iteration.

LBP has the advantage that the information obtained during an iteration is available to aid the remainder of the iteration [48]. Owing to this however, it does not have the same high level of parallelism as the flooding schedule, possibly resulting in a lower processing throughput and a higher processing latency. It can also be seen that m CN activations and $d_c \times m$ VN activations occur per iteration, resulting in a higher computational complexity per iteration, when compared to the flooding schedule. However, it will also be shown in Section 2.4.2 that CN activations can be significantly more computationally expensive than the VN activations, hence the increased cost is manageable. Additionally,



$\tilde{c}_1$ $\tilde{c}_2$ $\tilde{c}_3$ $\tilde{c}_4$ $\tilde{c}_5$ $\tilde{c}_6$ $\tilde{c}_7$ $\tilde{c}_8$ $\tilde{c}_9$ $\tilde{c}_{10}$
V_1 V_2 V_3 V_4 V_5 V_6 V_7 V_8 V_9 V_{10}
C_1 C_2 C_3 C_4

FIGURE 2.4: An example of the layered belief propagation schedule

LBP tends to converge to the correct codeword using fewer iterations and therefore with lower overall computational complexity than flooding [33, 48].

A common method for implementing LBP decoding is by also taking advantage of the *multicast*-style messages proposed in [49, 50]. Here, the VNs function simply as memories, where each VN V_j stores a single message α_j, which it broadcasts to all of its connected CNs [47]. When CN C_i is activated at iteration T, for each connected VN V_j, it first subtracts the check-to-variable message calculated during the previous iteration from the incoming broadcast VN message,

$$\alpha^{T}_{j\to i} = \alpha_j - \beta^{T-1}_{i\to j}. \tag{2.14}$$

Once the current values of $\alpha^T_{j\to i}$ have been calculated, they may then be used to activate C_i as normal,

$$\beta^T_{i\to j} = F_{CN}(\alpha^T_{j\to i}), \tag{2.15}$$

where the CN update function $F_{CN}(\cdot)$ is defined for varying algorithms and message representations in Section 2.4.2. The new values of $\beta^T_{i\to j}$ must then be stored to be used at iteration $T+1$, and are used to calculate the updated value of the broadcast message for V_j,

$$\alpha_j = \beta^T_{i\to j} + \alpha^T_{j\to i}. \tag{2.16}$$

This variation of the layered decoding schedule requires a slightly greater amount of storage at each node, but benefits from greatly reduced routing complexity for the VN messages.

2.4.1.3 Informed dynamic scheduling

Informed Dynamic Scheduling [51, 52] inspects the messages that are passed between the various nodes, selecting to activate whichever node is expected to offer the greatest improvement in belief. This requires IDS to perform additional calculations in order to determine which node to activate at each stage of the decoding process. However, IDS facilitates convergence using fewer node activations than in either flooding or LBP, which may lead to a lower complexity overall [53].

During IDS, the difference between the previous message sent over an edge and the message that is obtained using recently-updated information [44] is calculated. This difference is termed the *residual* [52], and represents the improvement in belief that is achieved by the new message. Like the LBP schedule, IDS is commonly centred on the CNs. At the start of the iterative decoding process, the residual for each output of each CN is calculated as the magnitude of the message to be sent over that edge. The message with the greatest residual is identified, and the receiving VN is then activated, sending updated messages to each of its connected CNs. These CNs then calculate new residuals for each of their edges as the difference between its new message and its previous message. All of the residuals in the graph are then compared for the sake of identifying the new maximum, before the process is repeated.

Using Figure 2.5 as before, suppose that at the start of the iterative decoding process, the message $\tilde{\beta}_{3\to 8}$ from CN C_3 is identified as having the highest magnitude of all the check-to-variable messages in the graph. Owing to this, $\tilde{\beta}_{3\to 8}$ is passed to the VN V_8, which is then activated, in order to obtain the message $\tilde{\beta}_{8\to 2}$ which is then passed to C_2. The CN C_2 can then be activated to calculate new residuals for its other four edges, as the difference between their previous messages and their new messages that have been obtained using the updated information from V_8. These new residuals are then compared with the others from the previous step, allowing a new global maximum to be

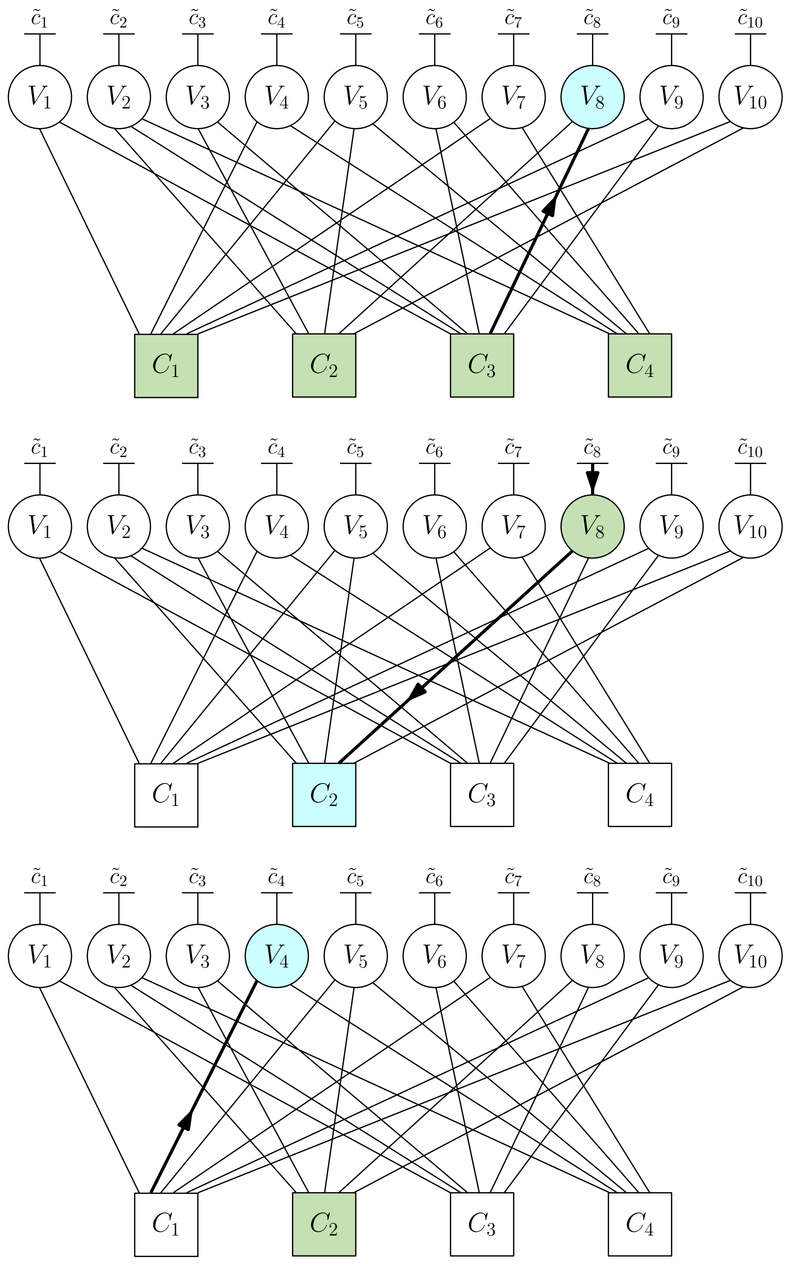


FIGURE 2.5: An example of informed dynamic scheduling

identified, to inform the next step of the decoding process. Note that the next highest residual within the factor graph does not necessarily have to originate from the most recently updated CN C_2. In the example seen in Figure 2.5, it can be seen that C_2 is activated to calculate residuals but it is $\tilde{\beta}_{1\to 4}$ from CN C_1 to VN V_4 that is sent. This implies that there is no single straightforward concept of iterations in IDS, since it is possible for a particular CN to be updated several times before another is updated once.

2.4.2 Check node calculations

The calculations performed within the CNs vary between different LDPC decoding algorithms. Of the many that exist, the two most common LDPC decoding algorithms are the Sum-Product Algorithm (SPA) [54] and the Min-Sum Algorithm (MSA) [55].

When the i-th CN C_i is activated, the message that it passes to VN V_j is a function of the messages received from all other connected VNs, except for V_j [56]. In C_i, this message will represent the likelihood that the bit at V_j should be 1, which is equivalent to the probability that parity check i has **not** already been fulfilled by the bits of the other connected VNs. This is achieved by calculating the probability that C_i is receiving an odd number of 1s from its other edges. For a degree $d_c = 3$ CN, each output depends on two soft message inputs representing the connected VN bits, which we may generally term a and b to simplify the following discussions. We also note the probability $P(a = 1)$ as P_a, and consequently $P(a = 0)$ as $(1-P_a)$. In the probability domain, the CN update function may hence be viewed as

$$F_{CN}(P_a, P_b) = P(\text{odd 1s}) = P(a \oplus b = 1) = P_a(1 - P_b) + P_b(1 - P_a). \tag{2.17}$$

By making use of (2.4), this probability may alternatively be represented as an LLR[1], according to

$$\begin{aligned}\tilde{F}_{CN}(P_a, P_b) &= \log \frac{(1 - F_{CN}(a, b))}{F_{CN}(a, b)} = \log \frac{P(\text{even 1s})}{P(\text{odd 1s})} \\ &= \log \frac{P_a P_b + (1 - P_a)(1 - P_b)}{P_a(1 - P_b) + P_b(1 - P_a)}.\end{aligned} \tag{2.18}$$

Similar equations may be derived for cases having more than two inputs by recursively substituting (2.17) or (2.18) back into themselves [57]. For example, the three-input probabilistic CN function $F_{CN}(P_a, P_b, P_c)$ may be expressed as

$$F_{CN}(P_a, P_b, P_c) = F_{CN}(P_a, P_b)(1 - P_c) + P_c(1 - F_{CN}(P_a, P_b)). \tag{2.19}$$

When the messages regarding the values of bits a and b are themselves expressed as the LLRs $\tilde{a}$ and $\tilde{b}$, we may invert (2.4) and substitute it back into (2.18), which yields:

$$\tilde{F}_{CN}(\tilde{a}, \tilde{b}) = 2\tanh^{-1}\left(\tanh\frac{\tilde{a}}{2} \times \tanh\frac{\tilde{b}}{2}\right) \tag{2.20}$$

$$\begin{aligned}&= \text{sign}(\tilde{a}) \times \text{sign}(\tilde{b}) \times \min\left(|\tilde{a}|, |\tilde{b}|\right) \\ &\quad + \log\left(1 + e^{-|\tilde{a}+\tilde{b}|}\right) - \log\left(1 + e^{-|\tilde{a}-\tilde{b}|}\right) \\ &= \tilde{a} \boxplus \tilde{b}.\end{aligned} \tag{2.21}$$

[1]The reader is reminded that in any cases where it may be ambiguous, in this thesis we use the notation $\tilde{x}$ to denote that the value x is expressed as an LLR.

Here, we introduce the notation $\tilde{a} \boxplus \tilde{b}$, referred to as the *boxplus* operator [58]. Furthermore, this operator is also associative, with the result that the function may also be extended to greater number of inputs, i.e. $\tilde{F}_{CN}(\tilde{a}, \tilde{b}, \tilde{c}) = \tilde{a} \boxplus \tilde{b} \boxplus \tilde{c}$.

The SPA uses the full version of (2.20) given above, which leads to strong error correction performance but a high computational complexity. The MSA, on the other hand, is a reduced-complexity approximation of the SPA [59], using (2.21) without the correction factor terms, according to

$$\tilde{a} \boxplus \tilde{b} \approx \mathrm{sign}\big(\tilde{a}\big) \times \mathrm{sign}\big(\tilde{b}\big) \times \min\Big(|\tilde{a}|, |\tilde{b}|\Big). \tag{2.22}$$

Note however that the complexity reduction offered by the MSA is attained at the cost of a degraded LDPC error correction capability. This degradation may be mitigated by adding a low-complexity approximation to the correction factor terms to (2.22) or by multiplying (2.22) by a scaling factor, which may be optimised during the design of the LDPC decoder [15].

2.4.3 Variable node calculations

Unlike the CN update function F_{CN}, the VN update function F_{VN} does not generally vary between algorithms. However, the VN inputs and outputs may still be expressed either in terms of probabilities or LLRs, depending on the message representation scheme employed by the decoder.

When the j-th VN V_j is activated, the message it sends to each of its connected CNs C_i is calculated in the probabilistic case as the normalised cumulative product of all of the bit value probabilities it has received from all other edges, including the estimated channel message $\tilde{c}_i$ provided on the edge from the demodulator [56]. For a $d_v = 2$ example, where the inclusion of the channel input leads to each output being the product of two probabilities P_a and P_b, the VN update function is expressed as

$$F_{VN}(P_a, P_b) = \frac{P_a P_b}{P_a P_b + (1 - P_a)(1 - P_b)}, \tag{2.23}$$

where the denominator is employed to ensure that $F_{VN}(1-P_a, 1-P_b) = 1-F_{VN}(P_a, P_b)$. Once again, the update function given by (2.23) may be trivially extended to cases having greater numbers of inputs when employed by VNs having greater degrees, according to [57]

$$\begin{aligned} F_{VN}(P_a, P_b, P_c) &= \frac{F_{VN}(P_a, P_b) \times P_c}{F_{VN}(P_a, P_b) \times P_c + (1 - F_{VN}(P_a, P_b))(1 - P_c)} \\ &= \frac{P_a P_b P_c}{P_a P_b P_c + (1 - P_a)(1 - P_b)(1 - P_c)}. \end{aligned} \tag{2.24}$$

Meanwhile, the logarithmic nature of LLRs converts these probabilistic multiplications into additions, and removes the need to normalise the result. Accordingly, the LLR representation of the VN update function is

$$\tilde{F}_{VN}(\tilde{a}, \tilde{b}) = \tilde{a} + \tilde{b}. \tag{2.25}$$

The VNs are also used for deciding the values of the reconstructed codeword bits $\hat{\mathbf{c}}$. These are calculated based on the combination (i.e. the product or sum in probabilistic or LLR cases, respectively) of the messages received on the edges from all connected CNs, as well as on the edge from the demodulator. In the probability domain, the value of the bit $\hat{c}_j$ for the j-th VN adopts the value of 1 if the result is greater than 0.5, or 0 otherwise [60]. Meanwhile, the polarity of the result is used to decide the value of $\hat{c}_j$ in the LLR domain, which becomes 0 if the result is positive, or 1 if it is negative [48]. If the reconstructed codeword has a zero-valued syndrome $\mathbf{s} = \hat{\mathbf{c}} \times \mathbf{H}^T$, then the iterative decoding process may be considered to have been a success and the process may be terminated. If not, then the iterative decoding process may be continued until a zero-valued syndrome is obtained or until an affordable complexity limit is reached. Practical LDPC decoder designs may also include other stopping criteria, as discussed later in Section 2.5.4.

2.5 LDPC decoding architectures

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

2.5.1 Parallelism

The inherent parallelism of the belief propagation algorithm facilitates the design of fully-parallel LDPC decoder architectures, in which every VN and CN in the factor graph is implemented by separate hardware Node Processing Units (NPUs) [61]. Fully-parallel decoders can achieve very high processing throughputs by performing all of the VN updates and all of the CN updates simultaneously, using the flooding schedule of Figure 2.3. However, this is achieved at the cost of excessive hardware resource consumption. For long codes comprising thousands of bits, the inter-node routing may require a greater area than the nodes themselves [62], rendering this architecture impractical for many decoder designs. Additionally, significant further hardware resources are required for implementing flexible routing, using a Beneš network [50], for example. Otherwise, fully-parallel decoders are completely inflexible, only supporting the single code that they are designed for [63].

By contrast, decoders associated with a fully-serial architecture implement just a single Check Node Processing Unit (CNPU) and Variable Node Processing Unit (VNPU). These hardware units are time-multiplexed between the various nodes of the LDPC factor graph, using memories to store interim results [26, 61]. Fully-serial decoders require few hardware resources but are restricted to very low processing throughputs, since each decoding iteration could require thousands of clock cycles. However, since all of the factor graph edges are represented by memory addresses, fully-serial decoders can be readily adapted at run-time to implement a different LDPC factor graph, by rearranging the memory accesses [49].

In order to strike a compromise between the high processing throughput of fully-parallel architectures and the more modest hardware requirement of fully-serial architectures, many LDPC decoders implement a number of time-multiplexed NPUs in a so-called partially-parallel fashion [22, 59, 64–68]. This parametrizable degree of parallelism facilitates control over the trade-off between processing throughput and hardware resource requirements. Furthermore, this approach is of particular benefit when any structure within the PCM $\mathbf{H}$ can be exploited in the configuration of the nodes implemented in hardware. For this reason, QC codes are particularly suited to partially-parallel implementations [69], as discussed further in Section 4.2.

2.5.2 Representation of messages

Another architectural consideration is the digital representation of the messages passed between nodes. For example, when implementing two's complement fixed-point LLRs, it has been shown that increasing the resolution and range of the LLR representation by using a greater bit width has a positive effect on the error correction performance [70], at the cost of increasing the hardware resources required. Equally, the algorithms described earlier can be modified to replace the LLRs with single-bit hard decisions, but this causes them to suffer from a significant error correction performance loss.

It is therefore desirable for a designer to quantify the effect of the fixed-point bit width on the performance of a chosen decoding algorithm, in order to determine the smallest number of bits that are required in order to achieve a satisfactory error correction performance. One method of doing so is through the use of Extrinsic Information Transfer (EXIT) charts [71], which have been conceived by ten Brink for characterising the operation of iterative decoding algorithms. More specifically, EXIT charts visualize the quality of the LLRs output by the VNs and CNs as functions of the quality of the LLRs provided to the corresponding inputs. By plotting these EXIT functions for LDPC decoders employing a range of fixed-point bit widths, a designer can quantify at a glance, how each representation improves or degrades the quality of the LLRs and hence the resultant error correction performance of the LDPC decoder [72]. This eliminates the requirement to run multiple time-consuming Bit Error Rate (BER) simulations.

Further to this, some designs have demonstrated that the hardware requirement can be reduced by using non-uniform quantisation schemes [73], by sending the bits of the LLR in a serial fashion rather than in parallel [67], or by utilising non-binary [74] number representations. However, these methods can also have adverse effects on the node complexity and the decoding throughput, requiring yet further investigation. Alternatively, stochastic number representations may present an attractive alternative to fixed-point LLRs [62], as discussed further in Section 2.7.

2.5.3 Pipelining

In addition to the degree of parallelism and the representation of the inter-node messages, the design of an LDPC decoder must also consider the amount of pipelining used in the datapath. In general terms, pipelining is the process by which a long operation is divided into multiple discrete stages, allowing different stages of multiple operations to be executed in parallel, hence increasing the throughput [75]. In the context of LDPC decoders, pipelining typically refers to placing registers throughout the datapath, in order to reduce the maximum critical path length and hence increase the maximum operating frequency f_{max}. However, depending on the decoding schedule and degree of parallelism employed, data dependencies that exist between the factor graph nodes may limit the effectiveness of pipelining in practice [76–79].

For example, the layered decoding example detailed in Section 2.4.1.2 assumes that the results of each node computation are available to its connected nodes immediately. If a pipelining stage was introduced between the CN outputs and the VN inputs, the new messages calculated at CN C_i would not reach its connected VNs V_j until one clock cycle later. In this case, the activation of V_j would potentially have to be delayed by a clock cycle, thereby increasing the number of clock cycles per iteration [29, 69]. Alternatively, in pipelined partially-parallel or fully-serial architectures, in which inter-node messages are stored in memories, the scheduled activation of an NPU may sometimes be permitted to occur before all updated input messages have reached the end of their respective pipelines. In this case, the cost of pipelining may be a decreased error correction performance, when compared to a non-pipelined case over an equal number of iterations. However, it may be deemed that the benefit of increased f_{max} and hence increased throughput and decreased latency is a satisfactory trade-off, subject to the desired performance characteristics of the decoder.

2.5.4 Stopping criteria

Finally, the design of an LDPC decoder also has to consider how to terminate the decoding process. Commonly, checks are carried out following each decoding iteration to determine whether the current state of the recovered codeword is a legitimate permutation or not, signalling whether or not decoding has been successful [80,81]. These checks are performed based on the output of the VNs, as mentioned previously in Section 2.4.3.

Occasionally however, a received frame is corrupted in such a way that it can never be corrected. In this case, the iterative decoding process would loop infinitely, unless other criteria for stopping it were implemented. Owing to this, a maximum iteration or complexity limit may be imposed [63]. When this limit is reached, the iterative decoding process is terminated and decoding is deemed to have failed. In implementations where a low hardware resource requirement is a greater priority than high processing throughput, the iteration limit may be the only stopping criterion imposed. Here, every received message is decoded using the same number of iterations, without early stopping. In this case, the parity checks are only used at the end of the iterative decoding process, in order to determine whether the recovered codeword is valid or not. Early stopping can also be used to detect that no error correction progress is being made with successive decoding iterations [82], allowing the decoding process to fail and terminate before the iteration limit is reached.

2.6 FPGAs

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

2.6.1 Structure

The internal structure of an FPGA typically comprises a variable number of three main programmable elements, namely logic blocks, RAM blocks, and I/O blocks [75]. The inputs and outputs of these blocks are linked by programmable routing, as shown in the sample schematic of Figure 2.6 [83].

The most fundamental design of a logic block comprises a Lookup Table (LUT) and a D-type Flip-Flop (DFF), as shown in Figure 2.6. A LUT is a digital structure that can be programmed to perform any combinatorial function of its inputs, thus mimicking any possible combination of logic gates. Typically, FPGA LUTs have 4–6 inputs, which

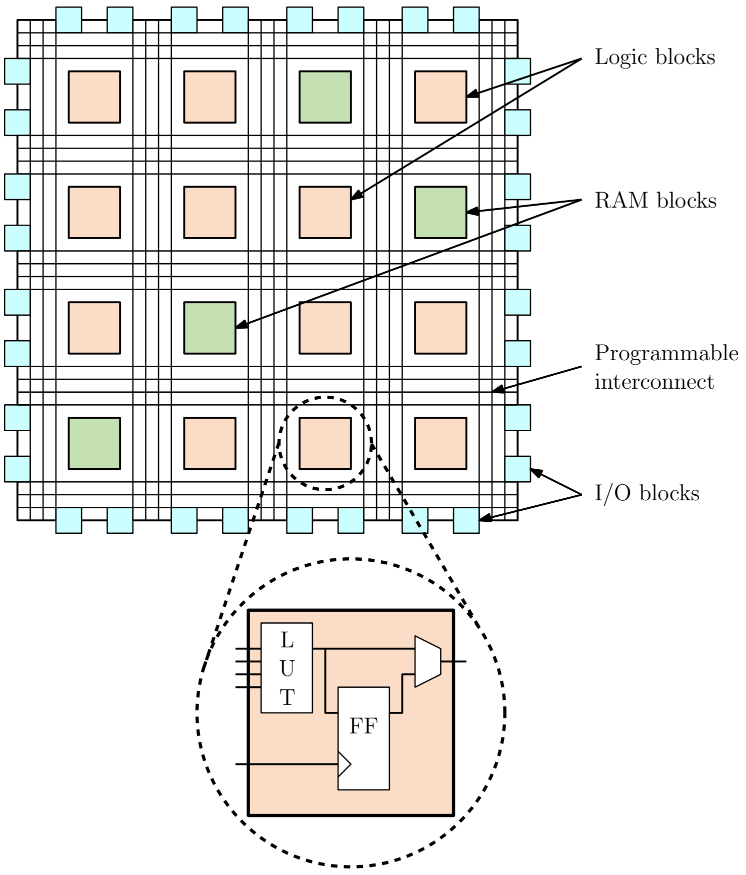


Figure 2.6: Example FPGA structure

are used to select a value for a single output bit. Increasing the number of LUT inputs typically allows the same HDL design to be implemented using fewer LUTs, therefore reducing the amount of FPGA routing required. However, the hardware resources required by a LUT increase exponentially with its number of input bits, hence very large LUTs are impractical [75]. The output of each LUT can optionally be connected to a corresponding DFF, for facilitating synchronous operation. Alternatively, the LUT output can be connected directly to the inter-block routing channels. These channels can be programmed to connect any set of logic block outputs to any set of logic block inputs, subject to the FPGA size constraints.

Instead of logic blocks, some locations within an FPGA structure may contain a RAM block for storing intermediate calculation results. The size of these RAM blocks depends on the particular FPGA being used, as does their access control. More specifically, some FPGAs provide dual-port RAMs, which allow reading and writing to two different locations simultaneously [28]. Some FPGAs may also include additional heterogeneous blocks, such as hardware multipliers and embedded processor cores, alongside non-volatile memory for storing the FPGA configuration when it is turned off [75].

2.6.2 FPGA vendor conventions

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

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

2.6.3 Comparing FPGAs

Due to the differences outlined above, comparing the hardware resources employed by contrasting designs implemented on different FPGAs is not straightforward. To this end, we propose an approximate metric based on the fundamental building blocks of FPGAs, namely the 4LUT and the DFF [84]. We refer to this metric as *equivalent logic blocks* (ELBs), since it attempts to approximate the number of simple logic blocks comprising one 4LUT and one DFF that would be required to implement each design. When calculating the number of ELBs, the LUT resources within each Intel ALM are considered to be equal to two 4LUTs, since this is one of the operating modes they offer. Similarly, to compensate for the increased size of 6LUTs compared to 4LUTs, and the additional logic that accompanies 6LUTs within Xilinx slices, each Xilinx 6LUT is considered to be approximately equal to two 4LUTs. Once this has been taken into consideration, we assume that the number of ELBs required by a design is given by the maximum of the number 4LUTs and the number of DFFs that it requires.

Table 2.1 presents an overview of the main generations and models of FPGAs available from Intel and Xilinx, along with the year of their release and the maximum number of ELBs available within the largest FPGA from each family.

Note that the proposed ELB metric is by no means perfect, since it does not consider the overhead associated with routing between logic elements or the use of additional FPGA blocks, such as memory or embedded multipliers. However, it does serve as a functional

Table 2.1: Comparison of FPGAs available from Intel and Xilinx

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

approximation of the hardware requirements associated with each design considered, if they were all implemented on the same FPGA. Measuring the usage only in terms of these fundamental building blocks permits a comparison between modern FPGA models and much older designs, which would not otherwise be possible. This metric is therefore used in the survey and comparison of published FPGA-based LDPC decoders presented in Chapter 3.

2.7 Stochastic LDPC decoding

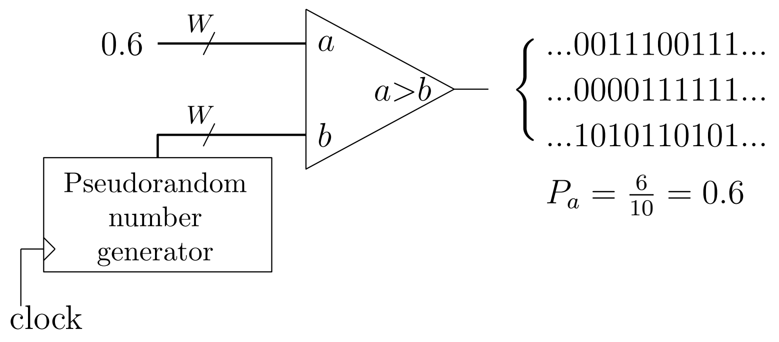
Stochastic number representations [85–87] have recently been demonstrated as an attractive alternative to fixed-point numbers for the inter-node messages sent within an LDPC decoder [62, 88]. Stochastic computing represents a probability P as a stream of bits known as a *Bernoulli sequence*, where the probability of each bit in the sequence adopting the value 1 is equal to P. These bits are generated at run-time by a random hardware process, typically a pseudorandom number generator, at a rate of one stochastic bit per clock cycle [89]. Accordingly, each stochastic bit stream requires only a single wire to transmit information between processing nodes, rather than the W wires required to transmit a fixed-point LLR [85]. Additionally, as will be shown in Sections 2.7.1 and 2.7.2, the hardware required to compute certain operations may be significantly reduced by representing the operands as stochastic bit streams, rather than fixed-point binary words.

Due to these advantages, stochastic numbers were first proposed for use in LDPC decoders in [90], and continue to motivate further research [10, 12, 14, 57, 84, 91–98]. This section presents the fundamentals of stochastic computation in Section 2.7.1, and how it may be applied to LDPC decoders in Section 2.7.2. Finally, some of the major challenges associated with stochastic decoding are explained in Section 2.7.3, alongside some of the methods that have been proposed to overcome them.

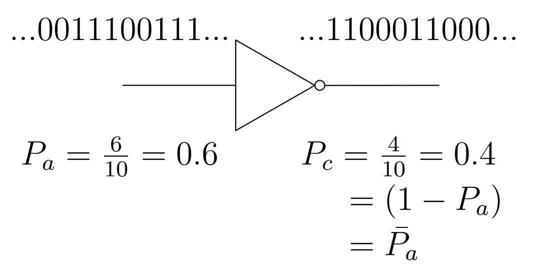
2.7.1 Stochastic arithmetic

As described previously, stochastic computing represents probabilities as streams of bits, where the fraction of 1s in each stream conveys the probability it represents. A common low-complexity method of generating these streams from the desired W-bit fixed-point probability is through the use of a pseudorandom number generator and a comparator, as depicted in Figure 2.7(a). Note that due to the random nature of this process, the positions of the individual 1s and 0s within a stochastic bit stream cannot be guaranteed, and hence many different stochastic bit streams can represent the same probability [86]. This property is also illustrated in Figure 2.7(a), where the three example stochastic streams all represent the same probability, $P_a = 0.6$.

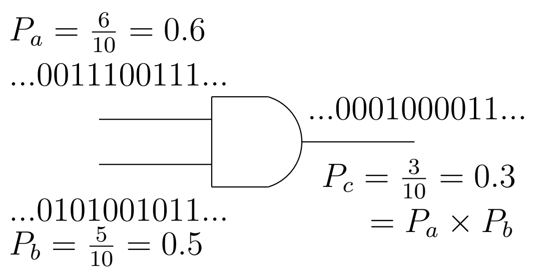
Using stochastic number representations can also simplify the hardware required to perform certain arithmetic operations. For example, Figure 2.7(b) illustrates that the complement $P_c = \bar{P_a} = (1 - P_a)$ of a probability P_a may be realised through the use of a single NOT gate. Meanwhile, the multiplication $P_c = P_a \times P_b$ of two probabilities P_a and P_b is shown in Figure 2.7(c) to be implementable using a single AND gate. Furthermore, Figure 2.7(d) illustrates that placing two stochastic probabilities P_a and P_b at the inputs of an exclusive-or (XOR) gate results in the computation of $P_c = P_a(1-P_b)+P_b(1-P_a)$. Finally, the normalised division of probability P_a by probability P_b, $P_c = \frac{P_a}{P_a+P_b}$ may be computed by connecting P_a and P_b to inputs J and K, respectively, of a JK-type Flip-Flop (JKFF), as depicted in Figure 2.7(e). The operation of a JKFF is as follows. If the two input bits J and K disagree, the JKFF output Q is set equal to J. Alternatively, if $J = 0$ and $K = 0$, the output Q holds its previous output, whilst if $J = 1$ and $K = 1$, the output Q is set to the inverse of its previous output [88].



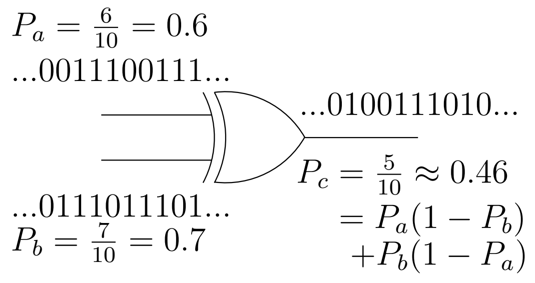
(a) Generation of stochastic bit stream for a given probability



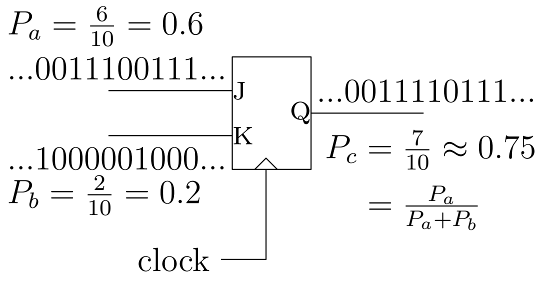
(b) Complement of a stochastic probability


$P_a = \frac{6}{10} = 0.6$
...0011100111...
...0001000011...
$P_c = \frac{3}{10} = 0.3$
$= P_a \times P_b$
...0101001011...
$P_b = \frac{5}{10} = 0.5$

(c) Multiplication of stochastic probabilities


$P_a = \frac{6}{10} = 0.6$
...0011100111...
...0100111010...
$P_c = \frac{5}{10} \approx 0.46$
$= P_a(1 - P_b)$
$+P_b(1 - P_a)$
...0111011101...
$P_b = \frac{7}{10} = 0.7$

(d) XOR of two stochastic bit streams


$P_a = \frac{6}{10} = 0.6$
...0011100111...
J
Q
K
...0011110111...
$P_c = \frac{7}{10} \approx 0.75$
$= \frac{P_a}{P_a + P_b}$
...1000001000...
$P_b = \frac{2}{10} = 0.2$
clock

(e) Division of stochastic probabilities

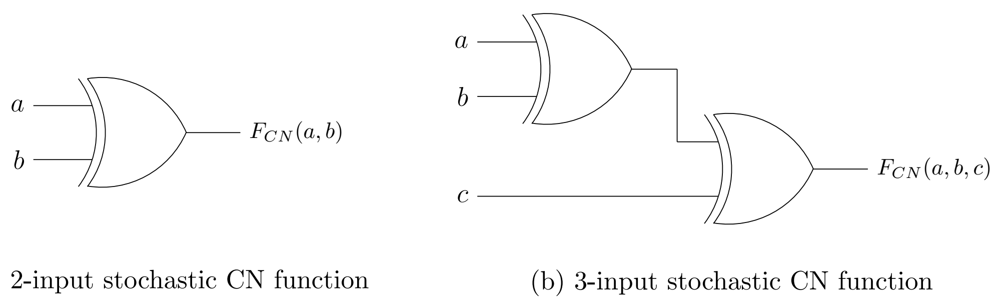
FIGURE 2.7: Examples of stochastic bit stream generation and computations

Note that throughout Figure 2.7, the example sequences often only represent an approximation of the correct probability, due to their short sequence length. Typical stochastic bit streams commonly use many tens or hundreds of bits, and can therefore maintain a much higher degree of accuracy.

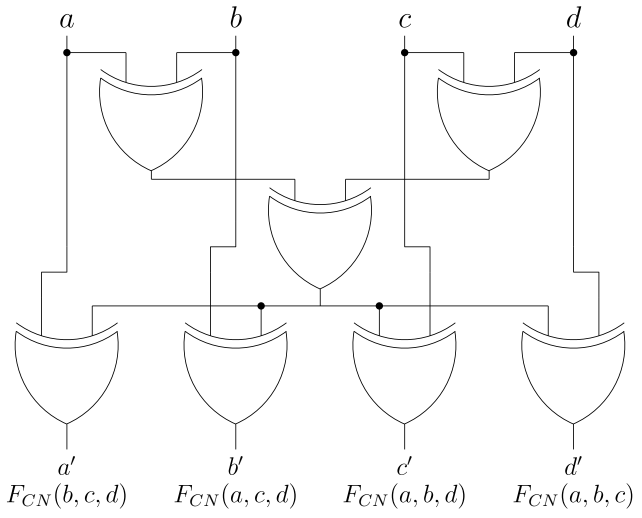
2.7.2 Stochastic LDPC computations

By combining the operations described above in Section 2.7.1, it is possible to implement the LDPC decoding equations presented in Sections 2.4.2 and 2.4.3 using stochastic numbers with only a small hardware resource requirement.

A $d_c = 3$ CN performs the 2-input CN function $F_{CN}(P_a, P_b)$ (2.17) three times, once for each unique combination of $d_c - 1$ inputs. For each pair of stochastic inputs a and b, each containing a single bit in the stochastic bitstream representing P_a and P_b respectively, this function $F_{CN}(a, b)$ may be implemented using a single XOR gate, as depicted in Figure 2.8(a) [89]. For larger numbers of inputs, required by CNs having a degree d_c



(a) 2-input stochastic CN function (b) 3-input stochastic CN function

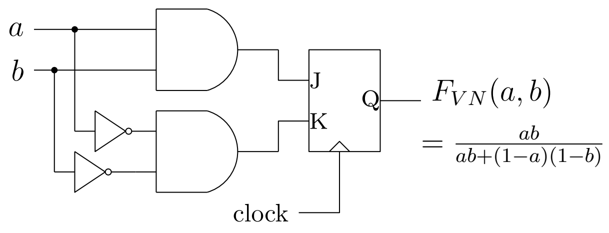


(c) 4-input 4-output stochastic CN architecture

FIGURE 2.8: Stochastic realisations of the CN function

that is greater than 3, a succession of XOR gates could be used according to (2.19), as exemplified for the 3-input case $F_{CN}(a, b, c)$ for a single output of a $d_c = 4$ CN in Figure 2.8(b). Simplistically, in $d_c = 4$ CNs, the structure of Figure 2.8(b) could be repeated four separate times, hence requiring a total of eight XOR gates. However, due to the fact that the XOR operation is its own inverse (hence $a \oplus b \oplus a = b$), a more efficient method is to calculate the cumulative XOR of all inputs, and calculate each output as the XOR of this sum with the corresponding input. This is exemplified for the complete $d_c = 4$ stochastic CN of Figure 2.8(c) having inputs a to d and outputs a' to d', which requires only seven XOR gates.

Meanwhile, the 2-input probabilistic VN function $F_{VN}(a, b)$ (2.23) may be implemented through the combination of two AND gates, two NOT gates and one JKFF, in the configuration of Figure 2.9(a) for a single output of a $d_v = 3$ VN [89]. Additionally, Figure 2.9(b) provides the truth table for the VN output shown in Figure 2.9(a), having two stochastic



(a) 2-input stochastic VN function

a	b	J	K	$F_{VN}(a,b)$
0	0	0	1	0
0	1	0	0	Q_{prev}
1	0	0	0	Q_{prev}
1	1	1	0	1

generative

conservative

(b) Truth table

FIGURE 2.9: Stochastic realisation of the VN function

input bits a and b. It may hence be observed that if a and b agree, their value is provided to the output bit; meanwhile, if a and b disagree, that output retains its previous value. These two contrasting output states may be referred to as *generative* and *conservative* respectively, according to whether the output bit has been newly generated or conserved from the previous clock cycle. For greater numbers of inputs, the stochastic VN function may be extended by adding extra terms to the two AND gates, according to (2.24). However, doing so greatly reduces the probability of the output being a generative bit, due to the reduced likelihood of either of the AND gates having an output of 1. This is one of the causes of the so-called latching problem, which is discussed in the following subsection.

2.7.3 Issues with stochastic decoding

The combination of single-bit messages and simplified computation hardware are a significant benefit of employing stochastic numbers in an LDPC decoder. However, the disadvantage of these single-bit messages is a reduction in the quantity and quality of information in each inter-node message. This results in stochastic decoders requiring a far greater number of iterations - known as Decoding Cycles (DCs) in stochastic decoders - than their fixed-point counterparts. This in turn may lead to a reduced processing throughput, and hence a higher processing latency [86].

Another potential issue with stochastic decoding is the well-known problem of latching [62]. Latching occurs when a set of nodes becomes locked into a fixed state, which is maintained by the repeated stochastic bits that are generated by that state. For example, in a DC in which every CN input is 0, every resultant CN output would also become 0, which could lead to every VN output being a generative 0, and so on. Latching may occur for a number of reasons, but manifests more frequently for factor graphs which include short cycles [99], in which node inputs and outputs become correlated. Two methods to combat the latching problem are to increase the switching activity of the VN input channel bits, and to re-randomise conservative VN output bits. These two methods are briefly described here, but more details can be found in [88, 89, 100].

2.7.3.1 Scaling the channel probabilities

One of the ways in which a group of nodes may be broken out of a latched state is by receiving a combination of channel bits which is different from those that have come previously. However, at high SNR levels, the received channel probabilities P_{ch} will tend to be very close to 0 or 1, resulting in channel bit streams which are all 0s or all 1s, respectively. Accordingly, Noise-Dependent Scaling (NDS) has been proposed [60] to increase the level of switching activity by scaling the received channel probabilities according to the channel's noise power spectral density N_0. For the case of BPSK transmission over an AWGN channel, this is defined as [89]

$$P_{ch} = \frac{1}{1 + \exp\left(\gamma \times N_0 \times 4 \times R \times \frac{E_b}{N_0} \times \mathrm{Re}(y_i)\right)}, \tag{2.26}$$

which is the probabilistic form of (2.5) with the additional factor $\gamma \times N_0$. Here, γ is an empirical factor which can be chosen to optimise the performance of the stochastic decoder [60]. This is discussed further for the proposed flexible stochastic LDPC decoder in Section 5.4.1.

2.7.3.2 Edge memories and internal memories

Edge Memories (EMs) [62] represent an alternative to the JKFF for producing the conservative bit at each output edge of a VN. The purpose of edge memories is to break any correlation between VN input bits and output bits by re-randomising the output. This is achieved by storing the most recent L_E generative bits produced at that edge in an L_E-bit shift register. Any time the VN inputs agree and produce a generative bit, this is added to the EM, as well as being sent on the output edge. Conversely, when the VN inputs disagree, the outgoing conservative bit is instead randomly selected from one of the bits stored in the EM. It may hence be seen that an EM effectively operates as a JKFF having an L_E-bit addressable memory [92].

At the same time, Internal Memories (IMs) were proposed [62] to further relieve the problems associated with high-degree VNs mentioned in Section 2.7.2. IMs have an almost identical structure to EMs, with the exception of storing a much smaller number L_I of generative bits. VNs may then be built up of a logical structure of successive 2-input 1-output IMs, only using EMs as the final subnode for each output edge [100]. This has the advantage of preventing latching at every point throughout the internal VN architecture, whilst reducing the hardware resource requirements compared to employing large EMs at every stage. We propose a novel flexible generalised structure for VNPUs employing IMs and EMs later in Section 5.3.3.2.

2.8 Conclusion

This chapter has presented an introduction to many of the key considerations regarding FPGA-based LDPC decoders. The concepts and terminology introduced here will now be used throughout the rest of this thesis in order to describe the novel contributions of this work.

Chapter 3

A survey of FPGA-based LDPC decoders

3.1 Introduction

As introduced in Chapter 1, the design of a complete LDPC decoder is subject to a complex interplay between a number of system characteristics, namely the processing throughput, processing latency, hardware resource requirements, error correction capability, processing energy efficiency, bandwidth efficiency and flexibility. These characteristics depend on a number of system parameters, namely the architecture, the LDPC code employed, the algorithm used, and the number of decoding iterations. This relationship is shown in Figure 3.1. Note that the bandwidth efficiency also depends on the modulation scheme chosen, as does the transmission energy efficiency, which furthermore depends on the coding gain and the error correction capability of the chosen LDPC code. To elaborate a little further in the context of Figure 3.1, we can improve the error correction capability in many different ways, for example by using a stronger LDPC code or more decoding iterations. Naturally, increasing the number of iterations increases the complexity and hence reduces the processing energy efficiency, but increases the transmit energy efficiency. Hence the total energy dissipation should be considered holistically, when designing an LDPC decoder. Further similar trade-offs will emerge throughout the forthcoming discussions.

Before proposing any new designs of an FPGA-based LDPC decoder, the trade-offs and relationships between these parameters and characteristics must be understood. In order to do so, we have extensively reviewed the current literature and accumulated the parameters and characteristics of all known FPGA-based LDPC decoder designs in Table 3.1, presented in this chapter. These results may then be used to visually plot the

This chapter is based on the following publications:

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

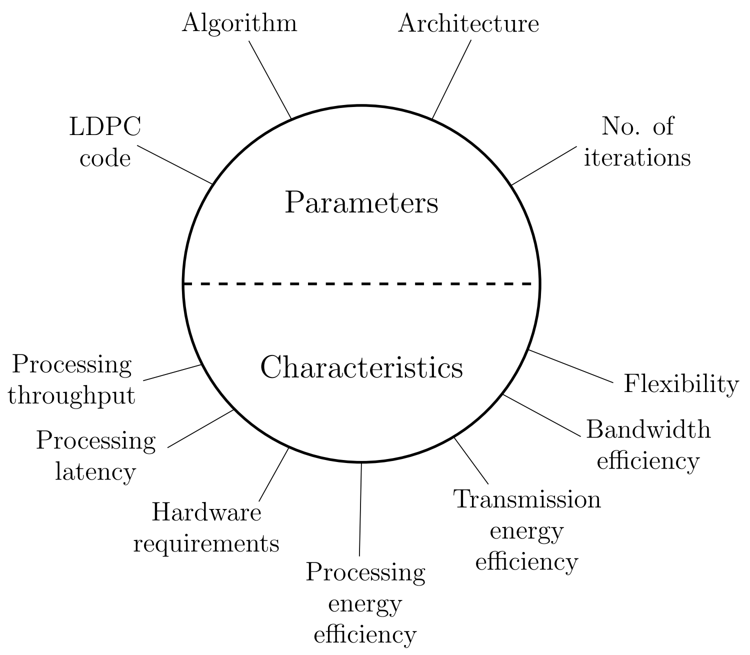


FIGURE 3.1: FPGA-based LDPC decoder system parameters and characteristics

relationships between the various parameters and characteristics, in order to investigate and characterise them.

The structure of this section is as follows. Firstly, Section 3.2 presents the condensed results from our survey of 140 published designs of FPGA-based LDPC decoders. This section also includes an in-depth discussion of each individual parameter and characteristic, including the ways in which they were measured and characterised during the compilation of Table 3.1. Following this, Section 3.3 presents our analysis of the trade-offs and relationships between the parameters and characteristics depicted in Figure 3.1. In Section 3.4, we then use the experience gained during the completion of this survey to provide recommendations and suggestions for further work within this field. Finally, concluding remarks are presented in Section 3.5, as well as a summary of the ways in which the results of this survey have influenced the rest of this thesis.

3.2 Comparison of decoders

A comprehensive review of published FPGA-based LDPC decoder designs is presented in this section. The analysis of Table 3.1 considers both the parameters that are chosen by the designers, as well as the characteristics that may be measured based on the design. Each of these is discussed and characterised in Sections 3.2.1 and 3.2.2, together with explanations and discussions of the symbols used in Table 3.1 where applicable. The entries in Table 3.1 have been sourced from both academic publications and commercially-available soft IP cores. Unfortunately, the licensers of these commercial designs were often unwilling to divulge many of the parameters and characteristics required for this analysis, resulting in several incomplete sets of results. Furthermore, none of the licensers were willing to provide pricing information for the purposes of this survey, preventing the comparison of this interesting but non-technical characteristic of their IP.

Note that we have limited this survey to FPGA implementations of LDPC decoders. This is because FPGA implementations of LDPC decoders cannot be fairly compared to Application-Specific Integrated Circuit (ASIC) implementations, which are designed at a significantly higher development cost to have particularly high performance for high-production-run applications. Indeed, ASIC implementations are even difficult to compare with each other, because some papers provide post-synthesis results, while others offer post-layout results. Meanwhile, some papers consider only the ASIC core, while others include both the memory and Input/Output (I/O) resources.

Table 3.1 presents a condensed version of our findings, showing only the most significant parameters and characteristics. In the case of references that present multiple FPGA-based LDPC decoder designs, only a representative subset has been reproduced here. A full version of our survey results may be downloaded from [101].

3.2.1 Parameters

In this section, we consider the parameters of FPGA-based LDPC decoders, which include all factors of the design that are specified by the designer. These include which LDPC Parity-Check Matrices (PCMs) to support, the decoding algorithm to employ and the number of decoding iterations used. These parameters are discussed in Sections 3.2.1.1, 3.2.1.3 and 3.2.1.4 respectively. Section 3.2.1.2 describes the architectural parameters, namely the degree of parallelism, inter-node message representation, clock frequency, flexibility, and choice of FPGA.

3.2.1.1 LDPC PCMs

One of the most fundamental features of an LDPC decoder is the selection of the PCMs that it is designed to support. Decoders may support just one PCM, be tailored to a family of related PCMs or may be designed to be completely flexible. As discussed in Section 2.2, each PCM $\mathbf{H}$ has a number of parameters, namely n, m, d_c, and d_v. However, the total number of edges in the corresponding factor graph can be considered to encompass all of these factors, representing the overall size and complexity of the code, as listed in Table 3.1.

TABLE 3.1: Comparison of FPGA-based LDPC decoders

Ref.	H dimensions	Edges (k)	PUs	LLR bitwidth	Clock (MHz)	FPGA	Algorithm	Iterations	Throughput (decoded bps)	Throughput (encoded bps)	ELBs (k)	E_b/N_0 (dB)	Bandwidth eff.	Run-time flexibility
[73]	1022 x 8176	32.7	36	N 6	193	Xilinx Virtex 2	Sum-product (modified)	15	172 M	197 M*	38.3 *	3.78	0.88	None
[102]	4608 x 9216	27.6	54	U* 5	56	Xilinx Virtex E	-	18	27 M*	54 M	15.9 *	1.85	0.50	None
[67]	125 x 480	-	605	U* 3	61	Intel Stratix	Min-sum (modified)	15	481 M*	650 M	66.6	4.69	0.74	None
[103]	432 x 1440	4.32	576	U 4	138	Intel Stratix 2	Min-sum	16 **	668 M**	954 M*	31.6 *	3.41	0.70	None
[104]	4044 x 8088	-	24	U 6	44	Xilinx Virtex 2	Log-BP	25	40 M	80 M*	72.6 *	-	0.50	None
[105]	1022 x 8176	32.7	18	-	200	Intel Cyclone 2	BP-based	18	70 M	80 M*	8.0 *	3.8	0.88	None
[105]	1022 x 8176	32.7	18	-	200	Intel Stratix 2	BP-based	18	560 M	640 M*	38.0 *	3.8	0.88	None
[19]	648 x 1296	3.89	36	U* 4	128	Xilinx Virtex 2	Min-sum	8.4 **	86.6 M**	173 M*	10.8 *	2.74	0.50	3 codes in 802.11 WiFi
[106]	1728 x 3200	-	32	U* 5	180	Xilinx Virtex 4	Turbo decoding algorithm	10	103 M*	223 M	10.7 *	2.33	0.46	None
[66]	6144 x 12288	36.9	288	U 6	96	Intel Stratix 2	Min-sum	15	149 M*	298 M	92.0 **	1.48	0.50	None
[22]	576 x 1152	3.46	32	U 2	64	Xilinx Virtex 2	Min-sum (modified)	8.5 **	38 M**	76.1 M*	5.2 *	3.27	0.50	3 codes in 802.16 WiMAX
[84]	512 x 1024	3.07	1536	NA	212	Xilinx Virtex 4	Stochastic	NA	353 M*	706 M	54.5 **	2.43	0.50	None
[59] **T**	1022 x 8176	3.46	1728	U 2	138	Xilinx Virtex 5	Min-sum (modified)	6.8	11.7 G	23.4 G*	78.0 *	-	0.50	None
[59] **E**	1022 x 8176	3.46	1728	U 2	138	Xilinx Virtex 5	Min-sum (modified)	8.5 **	9.35 G**	18.7 G*	78.0 *	3.43	0.50	None
[20]	-	-	8	U* 6	160	Xilinx Virtex 5	-	20	21.6 M	25.9 M*	38.0 *	-	0.83	Complete 802.16 WiMAX
[107]	188 x 2209	8.84	235	U* 6	50	Intel Stratix	Min-sum (modified)	20	108 M*	118 M	23.5 *	5.12 **	0.92	None
[108]	1552 x 3104	-	48	N 6	98	Intel Cyclone 2	Belief propagation	30	26 M*	52 M	33.0	1.48	0.50	None
[26]	731 x 4161	16.6	2	U 9	-	Xilinx Virtex 2P	Min-sum with correction	10 *	1.45 M		1.2 **	3.52 **	0.82	Any code
[21]	4158 x 9036	27	54	U 6	100	Xilinx Virtex 2	Min-sum with correction	60 *	15 M	30 M*	53.3 *	1.17 *	0.50	3 custom codes
[33]	768 x 1536	4.61	144	U 5	211	Xilinx Virtex 4	Min-sum with scaling	-	397 M	794 M*	18.2 *	-	0.50	None
[33]	432 x 1296	4.75	81	U 8	160	Xilinx Virtex 4	Min-sum with correction	15 *	95 M	143 M*	19.3 *	2.49 *	0.67	Complete 802.11 WiFi
[109]	1152 x 2304	-	12	U* 7	155	Intel Stratix 2	Min-sum	8	233 M	465 M*	17.3 *	1.94	0.50	None
[109]	1152 x 2304	-	12	U* 7	128	Intel Stratix 2	Min-sum	8	768 M	1.54 G*	69.1 *	1.94	0.50	None
[110]	3048 x 6096	-	72	-	64	Xilinx Virtex E	-	24	32 M*	64 M	12.3 *	-	0.50	None
[23]	3600 x 16200	45	45	-	70.8	Xilinx Virtex 2P	Min-sum with scaling	15 *	36.3 M*	46.7 M	22.0 *	-	0.78	Complete DVB-S2
[23]	3600 x 16200	45	180	-	73.2	Xilinx Virtex 2P	Min-sum with scaling	15 *	149 M*	191 M	70.6 *	-	0.78	Complete DVB-S2
[111]	519 x 1038	3.11	9	U* 4	26.3	Xilinx Virtex E	-	18	36 M*	72 M	19.4 *	-	0.50	None
[65]	4095 x 4095	262	130	U 1	191	Xilinx Virtex E	Soft majority logic	5	1.56 G*	1.9 G	21.3 **	4.36	0.82	None
[25]	-	-	32	U 8	74	Xilinx Virtex 4	Normalized BP-based	15	5 M**	10 M*	30.6 **	-	0.50	Any code
[112] **T**	324 x 648	1.94	972	U 1	188	Xilinx Virtex 5	Simplified MP	3.8	16.2 G	32.4 G*	28.5 *	-	0.50	None

TABLE 3.1: Comparison of FPGA-based LDPC decoders *(continued...)*

Ref.	**H dimensions**	**Edges (k)**	**PUs**	**LLR bitwidth**	**Clock (MHz)**	**FPGA**	**Algorithm**	**Iterations**	**Throughput (decoded bps)**	**Throughput (encoded bps)**	**ELBs (k)**	**E_b/N_0 (dB)**	**Bandwidth eff.**	**Run-time flexibility**
[112] **E**	324 x 648	1.94	972	U 1	188	Xilinx Virtex 5	Simplified MP	10	6.16 G**	12.3 G*	28.5 *	5.41	0. 50	None
[113] **T**	640 x 1920	6.4	40	U 8	150	Xilinx Virtex 4	Joint row-column	8	1.33 G*	2 G	-	-	0. 66	None
[113] **E**	640 x 1920	6.4	40	U 8	150	Xilinx Virtex 4	Joint row-column	50	211 M*	320 M**	-	2.29	0. 66	None
[113] **T**	640 x 1920	6.4	1	U 8	300	Xilinx Virtex 4	Joint row-column	8	74.6 M*	112 M	-	-	0. 66	None
[113] **E**	640 x 1920	6.4	1	U 8	300	Xilinx Virtex 4	Joint row-column	50	11.8 M*	17.9 M**	-	2.29	0. 66	None
[114]	600 x 1200	3.6	1800	U 2	123	Xilinx Virtex 4	Min-sum (modified)	8.9 **	8.3 G**	16.6 G*	58.1 *	3.38	0. 50	None
[114]	324 x 648	1.94	972	U 2	113	Xilinx Virtex 5	Min-sum (modified)	8.4 **	4.33 G**	8.67 G*	44.0 *	3.64	0. 50	None
[74]	486 x 972	2.43	18	-	131	Xilinx Virtex 4	Min-sum (nonbinary)	20	50 M		87.8 *	2.49	0. 50	None
[24]	-	-	27	U 8	100	Intel Cyclone 2	Min-sum (modified)	-	175 M*	350 M	13.5 *	-	0. 50	Within code families
[62]	528 x 1056	3.34	1584	NA	222	Xilinx Virtex 4	Stochastic	NA	348 M*	697 M**	68.2 *	2.45	0. 50	None
[115]	1022 x 8176	4.75	12	U 11	228	Xilinx Virtex 5	Sum-product (modified)	-	522 M		29.7 **	2.61	0. 66	None
[116]	384 x 2048	12.3	33	U 6	100	Xilinx Virtex 2P	Message passing	10	19.5 M*	24 M**	13.7 *	4.07 ***	0. 81	None
[28]	768 x 1536	4.61	9	U 8	162	Xilinx Virtex 2	Min-sum	3	114 M*	229 M	2.4 *	-	0. 50	None
[28]	756 x 3969	16.8	-	U 6	200	Xilinx Virtex 4	Normalized min-sum	15	82.4 M*	102 M	10.0 *	-	0. 81	None
[28]	1022 x 8176	32.7	144	U 6	212	Xilinx Virtex 4	Normalized min-sum	15	625 M*	714 M	27.2 *	3.76	0. 88	None
[117]	600 x 1200	3.6	1800	U 3	100	Xilinx Virtex 4	Min-sum	10	6 G	12 G*	69.0 *	3.76	0. 50	None
[118]	768 x 1536	4.61	9	U 8	149	Xilinx Virtex 2	Min-sum	3	49.6 M*	99.1 M	2.9 *	-	0. 50	None
[64]	768 x 1536	4.61	9	U* 8	100	Xilinx Virtex 2	Min-sum	7	5.88 M*	11.8 M**	1.8 *	3.36 ***	0. 50	None
[77]	768 x 1536	4.61	9	U 8	84 *	Xilinx Virtex 2	Sum-product (modified)	20	4.3 M*	8.59 M**	3.9 *	2.37	0. 50	None
[77]	768 x 1536	4.61	9	U 8	79.1 *	Xilinx Virtex 2	Sum-product (modified)	20	4.04 M*	8.08 M**	3.4 *	2.37	0. 50	None
[77]	768 x 1536	4.61	9	N 8	80.5 *	Xilinx Virtex 2	Sum-product (modified)	20	4.21 M*	8.42 M**	4.9 *	2.96	0. 50	None
[27]	336 x 672	2.18	96	U 5	100	Xilinx Virtex 5	Min-sum with correction	10	475 M*	950 M	71.4 *	3.02	0. 50	None
[119]	-	-	-	U* 4	27	Xilinx Virtex E	-	-	15 M		1.7 *	-	-	None
[120]	768 x 1536	4.61	144	U 5	121	Xilinx Virtex 2	Min-sum (modified)	20	63.5 M*	127 M	20.4 *	2.79	0. 50	None
[121]	298 x 980	2.83	2	U 6	136	Intel Cyclone	Improved BP	-	7 M		1.0	4.4 **	0. 70	None
[122]	-	-	-	-	140	Xilinx Virtex 5	Min-sum	-	96 M		210 *	-	0. 67	Complete DVB-S2
[122]	-	-	-	-	140	Xilinx Virtex 5	Min-sum	-	206 M		388 *	-	0. 25	Complete DVB-S2
[123]	-	-	-	U* 9	180	Xilinx Virtex 5	Min-sum with correction	5	600 M	-	64.6 **	3.84	-	Complete CCSDS-C2
[123]	-	-	-	U* 9	150	Intel Stratix 2	Min-sum with correction	12	30 M	-	8.8 **	3.71	-	Complete CCSDS-C2
[124]	1152 x 2304	7.30	-	U* 8	160	Xilinx Virtex 4	Min-sum with correction	15	71 M	142 M*	36.4 **	1.84	0. 50	Complete 802.16 WiMAX

TABLE 3.1: Comparison of FPGA-based LDPC decoders *(continued...)*

Ref.	**H dimensions**	**Edges (k)**	**PUs**	**LLR bitwidth**	**Clock (MHz)**	**FPGA**	**Algorithm**	**Iterations**	**Throughput (decoded bps)**	**Throughput (encoded bps)**	**ELBs (k)**	**E_b/N_0 (dB)**	**Bandwidth eff.**	**Run-time flexibility**
[124]	1920 x 2304	7.68	-	U* 8	160	Xilinx Virtex 4	Min-sum with correction	15	169 M	204 M*	36.4 **	3.66	0.83	Complete 802.16 WiMAX
[125]	-	-	-	-	240	-	Min-sum with correction	-	866 M		-	-	-	Complete 802.11 WiFi
[126]	-	-	-	-	-	-	-	-	1 G	-	-	-	-	Complete ITU G.hn
[127]	-	-	-	-	-	-	-	10	-	2.1 G	-	-	-	Complete 802.11ad WiGig
[128]	2048 x 32640	-	-	U* 1	126	Intel Cyclone 4	Hard decision	12	8 G	8.54 G*	40.0	-	0.94	None
[129]	-	-	-	-	-	Xilinx Spartan 6	-	-	200 M		101 *	-	-	320 custom codes
[129]	-	-	-	-	-	Xilinx Spartan 6	-	-	25 M		22.4 *	-	-	320 custom codes
[130]	972 x 1944	-	-	-	-	-	Min-sum with correction	15	-	-	-	1.66	0.50	Complete 802.11 WiFi

3.2.1.2 Architecture

Architectural decisions influence the physical implementation and hardware used by the decoder. As described in Section 2.5, the primary architectural parameter is the degree of parallelism, which may be classified as fully-parallel, partially-parallel or fully-serial. This parameter may be quantified by the total number of Processing Units (PUs) instantiated by the decoder, as listed in Table 3.1. Frequently these processors perform the function of individual Variable Nodes (VNs) and Check Nodes (CNs), although some designs use a different approach, wherein the PUs cannot be directly mapped to specific factor graph components.

The operand width of the inter-node messages - typically in the form of Logarithmic-Likelihood Ratios (LLRs) - as listed in Table 3.1, is also a measurable parameter which affects the LDPC decoder's error correction performance. Designs using a higher number of bits may be expected to have superior error correction performance than their counterparts employing fewer bits. However, this is typically achieved at the cost of a larger hardware resource requirement or a lower processing throughput.

The quantisation scheme used in the LLR representation may be either uniform or non-uniform, as denoted by a 'U' or an 'N' in Table 3.1, respectively. In uniform quantisation schemes, the entire range of representable LLR values has a constant resolution, allowing the VN and CN functions to be implemented using straightforward binary arithmetic. By contrast, non-uniform quantisation schemes typically adopt a finer resolution for lower LLR magnitudes and a lower resolution for larger magnitudes. This facilitates a more beneficial trade-off between range and resolution, but makes the associated processing significantly more complex. Many authors (e.g. [64,67,111]) mention the number of bits used in their FPGA-based LDPC decoders, but do not detail the quantisation scheme employed. Since non-uniform schemes require significantly more details than uniform representations, these cases are assumed to employ uniform quantisation and are marked with an asterisk in Table 3.1.

The maximum achievable clock frequency of an FPGA-based LDPC decoder depends largely on the capabilities of the FPGA employed, but also on some design decisions such as the critical path length and the degree of pipelining. For example, designs that process entire VNs or CNs in a single clock cycle typically have long critical paths, while designs that only perform one arithmetic or logical operation per clock cycle typically have much shorter critical paths. Based on this observation, the clock frequency is included as a parameter in this analysis. The majority of authors have explicitly stated the clock frequency at which their decoder operates. However, in some cases (e.g. [77]) we have derived the clock frequency from other data, as indicated by an asterisk in Table 3.1.

Many decoder architectures are highly optimised to the specific characteristics of the single LDPC PCM that they are designed to support (e.g. [74, 109, 117]). By contrast, some other designs instead adopt a more general architecture (e.g. [20,25,26]), sacrificing performance for the flexibility to switch between several supported PCMs at run-time. A decoder's flexibility may be considered to be both a figure of merit and an architectural decision that is made by the designer, allowing it to be regarded as a characteristic or as a parameter. However, we show in Section 3.3.2 that adding flexibility to a design can only be achieved as a trade-off against some other desirable characteristics. For this reason, we treat flexibility as a characteristic in this survey.

The selection of an FPGA for the implementation of an LDPC decoder may have a significant impact upon its performance. The selected FPGA dictates the number of logic elements, memory blocks, and I/O pins that are available for all processing and routing. Additionally, some FPGAs facilitate higher clock frequencies than others when implementing the same design, depending on the process technology employed. Unfortunately it is impossible to fairly compare the capabilities of all FPGAs numerically. For this reason, Table 3.1 simply states which FPGA has been employed for each LDPC decoders considered.

3.2.1.3 Algorithm

As discussed in Section 2.4, several variations of the LDPC decoding algorithm exist. Some algorithms vary from each other only slightly, while others may employ vastly different mathematical concepts. Furthermore, different authors may use different terms to describe the same algorithm, making this parameter difficult to compare. Table 3.1

therefore only includes the terms used by the authors to describe their algorithms and no numerical comparison between them is inferred.

3.2.1.4 Iterations

The limit placed on the maximum number of decoding iterations has a significant effect upon the processing throughput and error correction performance of decoders operating without early stopping functionality, as well as in cases where the received frame is too corrupted to be decoded successfully. Decreasing the maximum number of iterations will increase the LDPC decoder's processing throughput in terms of the maximum achievable bit rate, but runs the risk of allowing errors to remain in the recovered codeword that could have otherwise been corrected. Generally, it can be assumed that the number of iterations used in each considered design was selected by the author to offer the most desirable trade-off between error correction performance and processing throughput, subject to the influence of the other parameters outlined above. It is also worth noting that the maximum number of iterations is perhaps the easiest parameter to change at runtime. Owing to this, some designs (e.g. [59, 112, 113]) are presented with two sets of results, namely one employing a low number of iterations for maximum processing throughput (marked with a 'T' in Table 3.1), and one with a high number for maximum error correction (marked with an 'E' in Table 3.1).

Table 3.1 presents the fixed number of iterations that are employed in designs without early stopping functionality, while the average number of iterations is presented for designs employing early stopping. However, some papers proposing early stopping designs (e.g. [21,23,26]) do not present an average number of iterations, only providing the maximum limit imposed, as indicated with an asterisk in Table 3.1. Likewise, some papers (e.g. [19,103]) do not state the number of iterations employed, but this parameter can be inferred as a function of other parameters and characteristics. These cases are marked with a double-asterisk (**) in Table 3.1.

3.2.2 Characteristics

In this section, we consider all those characteristics of FPGA-based LDPC decoders, which we plan to quantify. Seven main characteristics are identified, namely processing throughput, processing latency, hardware resource requirements, transmission energy efficiency, processing energy efficiency, bandwidth efficiency and flexibility, as seen Figure 3.1. Each of these is described in turn in the following sections.

3.2.2.1 Processing throughput

Perhaps the most frequently-stated characteristic of an FPGA-based LDPC decoder is its processing throughput, which is the number of bits that it can process per second. A high processing throughput is required for high-speed data transfers and video streaming applications, amongst other uses. A base station serving many users requires the sum of the individual throughputs to be high, so that each user receives a satisfactory service.

In an LDPC decoder it is important to note the difference between encoded and decoded processing throughput. We refer to the number of codeword bits processed per second as the encoded processing throughput, while we use decoded processing throughput to quantify the number of message word bits per second. For example, half of the codeword bits generated by a $1/2$-rate LDPC code are parity bits, which carry no information of their own. Therefore if the encoded processing throughput is 2 Gbps then the corresponding decoded processing throughput would be 1 Gbps. Ultimately, it is the decoded processing throughput that matters most to the user of the decoder, so we have deemed this to be the more important characteristic in comparisons. For designs where the author has only presented encoded processing throughput, we have inferred the decoded processing throughput by multiplying by the coding rate, as denoted by an asterisk in Table 3.1. In some cases it is unclear whether the stated processing throughput is encoded or decoded. This is reflected in the Table 3.1 by allowing the stated processing throughput to span both columns. A double asterisk is used in Table 3.1 to identify designs in which the processing throughput was not explicitly stated, but has been inferred from other stated parameters and characteristics.

3.2.2.2 Processing latency

The processing latency of an FPGA-based LDPC decoder is the amount of time it requires to process a complete LDPC codeword. Low processing latency is therefore important for interactive cloud computing and safety-critical operations, where an immediate response is crucial. It may be observed that processing latency is strongly linked to processing throughput, since the processing latency can often be calculated as the message word length k divided by the decoded processing throughput. However, some decoder designs achieve a high processing throughput by decoding more than one codeword simultaneously. In these cases, the associated processing latency would be much higher than that of a decoder which achieves the same processing throughput while decoding only a single codeword at a time. For example, a decoder that decodes a single 1000-bit message word with a processing throughput of 2 Gbps would have a processing latency of 0.5 μs, while two 1 Gbps decoders operating in parallel would achieve the same processing throughput, but would have a processing latency of 1 μs. Processing latency is a key characteristic of an FPGA-based LDPC decoder, however most authors do not explicitly state it in their results, and it is therefore not included in Table 3.1.

3.2.2.3 Hardware requirements

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

The Equivalent Logic Block (ELB) metric described in Section 2.6 can be used to compare the hardware resource requirements of designs implemented on different FPGAs. However, the resource requirements stated by the various authors of LDPC-based FPGA decoder designs often do not directly translate to ELBs, hence requiring further analysis to be performed as follows:

- The conversion from 6-input Lookup Tables (6LUTs) to 4-input Lookup Tables (4LUTs) described in Section 2.6 is first employed to ensure that all measurements of Lookup Tables (LUTs) consider an approximately equivalent quantity of hardware.
- Subsequently, if the hardware requirement of a design is quantified only in terms of either 4LUTs or D-type Flip-Flops (DFFs), then we assume a numerically equal number of ELBs.
- If the hardware requirement of a design is quantified in terms of both 4LUTs and DFFs, then we assume that ELBs = max(4LUTs, FFs). These cases are identified using a single asterisk in Table 3.1.
- For designs based on Xilinx FPGAs having complex multi-element slices, we have derived a "utilisation" figure of merit, which quantifies how many LUTs/DFFs are commonly used per slice. We obtained this by calculating the average utilisation of designs for which both the number of slices and the number of LUTs/DFFs used is stated. These utilisation figures were found to be approximately 0.83 for LUTs and 0.36 for DFFs, demonstrating that the majority of slices are used for their LUTs. For designs where the hardware utilisation is presented only in terms of slices, we assume ELBs = slices $\times$ 4LUTs per slice $\times$ 0.83. These cases are indicated in Table 3.1 using a double asterisk.

3.2.2.4 Transmission energy efficiency

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

The error correction performance of a decoder is typically characterised in the form of a BER curve, showing how the BER is reduced as the channel E_b/N_0 increases. In order to convert these plots into a comparable metric, we specified a desirable target BER of 10^{-4}. For each considered design, the E_b/N_0 required by the decoder in order to achieve this BER was noted. In some publications however (e.g. [21, 33]), the error correction performance is quantified using the Frame Error Rate (FER) rather than BER. In these cases, we assumed that a BER of 10^{-4} equates to a FER of 10^{-2} [21], based on the observation that the considered designs typically have a message word length k of the order of 1000 bits, as well as a minimum Hamming distance of the order of 10 bits. These cases are indicated using a single asterisk in the E_b/N_0 column of Table 3.1.

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

$$E_b/N_0\ _{[\mathrm{dB}]} = \mathrm{SNR}\ _{[\mathrm{dB}]} - 10\log_{10}(R). \tag{3.1}$$

Entries calculated in this way are denoted in Table 3.1 using a double asterisk. Unfortunately some authors have erroneously labelled the x-axis of BER plots as SNR, when E_b/N_0 would be more appropriate. Some of these cases were clarified via private correspondence with the authors. However, in some cases there is other evidence that the presented results are in terms of E_b/N_0 rather than SNR, such as comparisons with benchmarkers or capacity bounds. In these cases, E_b/N_0 is assumed and identified using a triple asterisk (***) in Table 3.1.

3.2.2.5 Processing energy efficiency

As for any electronic system, low processing energy consumption is desirable in the design of FPGA-based LDPC decoders. However, only a few publications ([22, 28, 59]) have included energy consumption measurements, hence this characteristic cannot be considered in our comparisons.

3.2.2.6 Bandwidth efficiency

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

3.2.2.7 Flexibility

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

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

3.3 Discussions

The data presented in Section 3.2 inspires a great deal of discussions and visualisation of the relationships amongst the various parameters and characteristics of FPGA-based LDPC decoders. This section commences by characterising the fundamental trade-off between desirable characteristics in Section 3.3.1, before identifying the parameters that affect each characteristic in Section 3.3.2.

3.3.1 Trade-offs

As seen in Figure 3.1 and discussed in Section 3.2.2, the main measurable characteristics of an FPGA-based LDPC decoder are processing throughput, processing latency, hardware resource utilisation, transmission energy efficiency, processing energy efficiency, bandwidth efficiency and flexibility. Of these, it is the processing throughput, hardware resource utilisation, flexibility, and transmission energy efficiency, which provide the clearest and most fundamental trade-off, since the other characteristics are all in some way dependent on these. The relationship amongst these four characteristics is plotted in Figure 3.2.

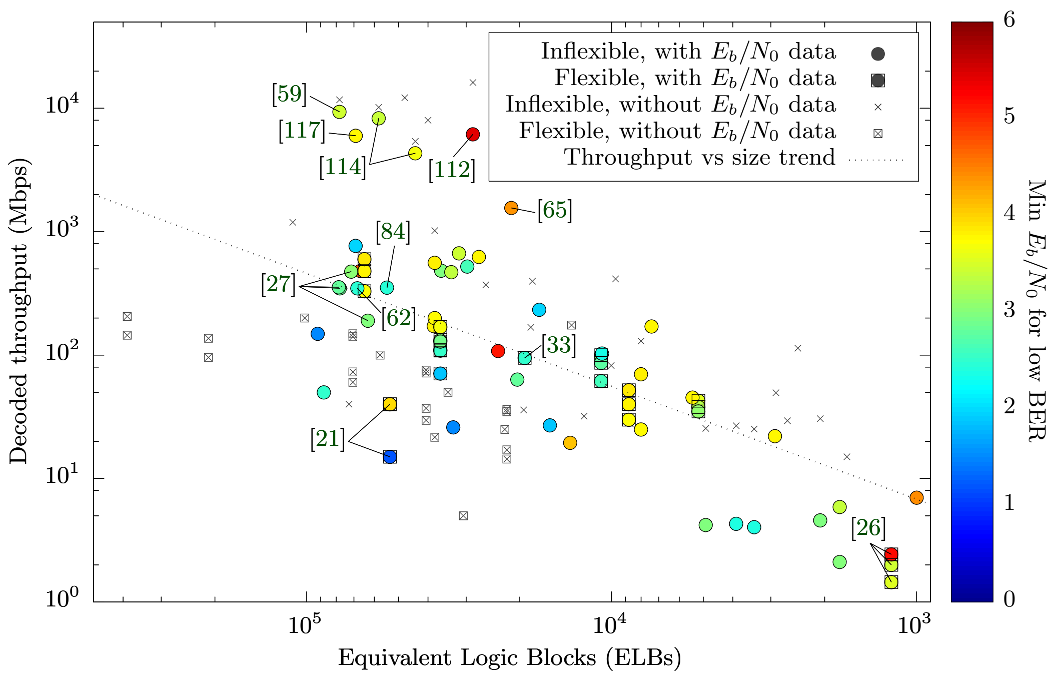


FIGURE 3.2: Processing throughput vs. hardware requirements vs. transmission energy efficiency vs. flexibility

Note that all scatter plots presented in this thesis are organised so that a decoder with desirable values for all characteristics would correspond to a data point in the top-right corner. For example, in Figure 3.2, the x-axis is plotted with the values reversed, so that decoders with smaller hardware resource requirements (preferred) are further to the right than larger ones. Meanwhile the y-axis is plotted as normal, so that decoders with the highest processing throughput are at the top. In this way, points above the trend line are superior to the average case, whilst points below it are inferior, notwithstanding the values of their other characteristics.

It can be seen in Figure 3.2 that most designs can only excel in at most three of the four characteristics presented. The trend line presents the average processing throughput vs size trade-off, and decoders that perform above this line generally tend to suffer from poor transmission energy efficiency, whilst decoders with a high energy efficiency tend to either have larger hardware resource requirements or lower processing throughput than the average case. Any decoders that perform well in all three of these characteristics tend to be totally inflexible to any PCM changes at run-time.

The five points in Figure 3.2 having the highest processing throughput are from [59], [117], [114] and [112], all of which employ fully-parallel architectures. The design of [112] has the smallest hardware resource requirement of the four, owing to its use of only one bit per LLR. By contrast, the designs of [59], [114] and [117] use two or three bits per LLR, which is reflected in their relative hardware resource requirements. None of these high-throughput decoders have any run-time flexibility, as is typical of fully-parallel architectures. The next highest processing throughput is achieved by the design of [65], which adopts a partially-parallel rather than fully-parallel architecture, but also uses only one bit per LLR. The effect of using these small numbers of bits can be seen in these decoders' poor transmission energy efficiency, since reducing the resolution of the LLRs impedes the associated error correction capability.

In addition to employing single-bit LLRs, the design of [65] achieves a high processing throughput by decoding two frames at once. The designs presented in [27] use a similar technique, processing three, four or even six frames in parallel using multiple decoder copies in the same FPGA. Owing to this, these designs have a larger processing throughput than the average case, while also having reasonable error correction performance. However, as discussed in Section 3.2.2.2, the processing latency of these decoders is much higher than their processing throughput would imply, making them less suitable for time-critical applications.

The decoders presented in [62] and [84] both achieve good transmission energy efficiency, while also having higher processing throughputs (or lower hardware requirements) than the average case. Both of these designs use stochastic bitstreams to represent the LLRs, facilitating a fully-parallel architecture having single-wire serial transmission between nodes, greatly simplifying the hardware design.

The points in the bottom-right of Figure 3.2 correspond to the designs presented in [26], which employ a fully-serial architecture and so have very low hardware resource requirements, but also low processing throughput. However, these designs also have the benefit of being truly run-time flexible for any LDPC code. By contrast, the other flexible designs shown in Figure 3.2, such as [33] and [21], are only flexible for a set of related PCMs.

In addition to the trade-offs described above, Figure 3.2 also demonstrates that it is difficult to consider all of the characteristics of an FPGA-based LDPC decoder at once. For example, Figure 3.2 does not consider the capabilities of the FPGA that each decoder is implemented using. In particular, more recent FPGAs may be able to operate identical designs at higher clock speeds than older FPGAs. This could be crudely factored into the results by dividing the processing throughput by the clock frequency, but doing so would then negate the impact of other parameters such as the critical path length. Furthermore, no consideration is given in Figure 3.2 to the processing latency of each considered design. Note, however, that by plotting the decoded processing throughput rather than the encoded processing throughput, the coding rate and the bandwidth efficiency of the LDPC code has been taken into at least partial consideration.

3.3.2 Relationships between parameters and each characteristic

Having established the fundamental trade-off that exists between the main characteristics of FPGA-based LDPC decoders, namely processing throughput, hardware requirements, transmission energy efficiency, and run-time flexibility, the following subsections present discussions of the parameters that affect each one. A discussion of bandwidth efficiency is combined with transmission energy efficiency in Section 3.3.2.4, but an in-depth quantitative discussion of flexibility and processing energy efficiency could not be made, owing to the lack of the required information in the publications considered.

3.3.2.1 Processing throughput

Figure 3.3 characterises the strong relationship between an FPGA-based LDPC decoder's degree of parallelism and its decoded processing throughput, confirming the expectation that designs having more parallel processors can decode a higher number of bits per second. Note that in Figure 3.3 the number of parallel processing units has been divided by the number of edges in the PCM $\mathbf{H}$, to remove the dependence on the LDPC code size. The colour and shapes of the markers in Figure 3.3 also indicate the influence that the number of bits per LLR and the number of decoding iterations have on the processing throughput, respectively. Points above the trend line typically employ a small number of bits per LLR or iterations, evidenced by their colour or circular point shape. By contrast, slower-than-average designs typically employ a larger number of bits per LLR or iterations, therefore having different colouration or a square shape.

Perhaps the most prominent points in Figure 3.3 are the light orange circles belonging to [113], which achieve a much higher processing throughput than the trend line, despite using 8 bits per LLR. This may be explained by this design's use of layered belief propagation with the aid of a novel joint row-column processor, which decreases the processing time of each iteration and helps to avoid memory conflicts, thereby increasing the processing throughput.

The dark orange triangles in the bottom-right represent the fully-serial decoders presented in [26], which achieve a low processing throughput owing to their low number of processors. Conversely, the dark blue points in the top-left represent the fully parallel decoders of [112] and [114], which achieve a very high processing throughput by

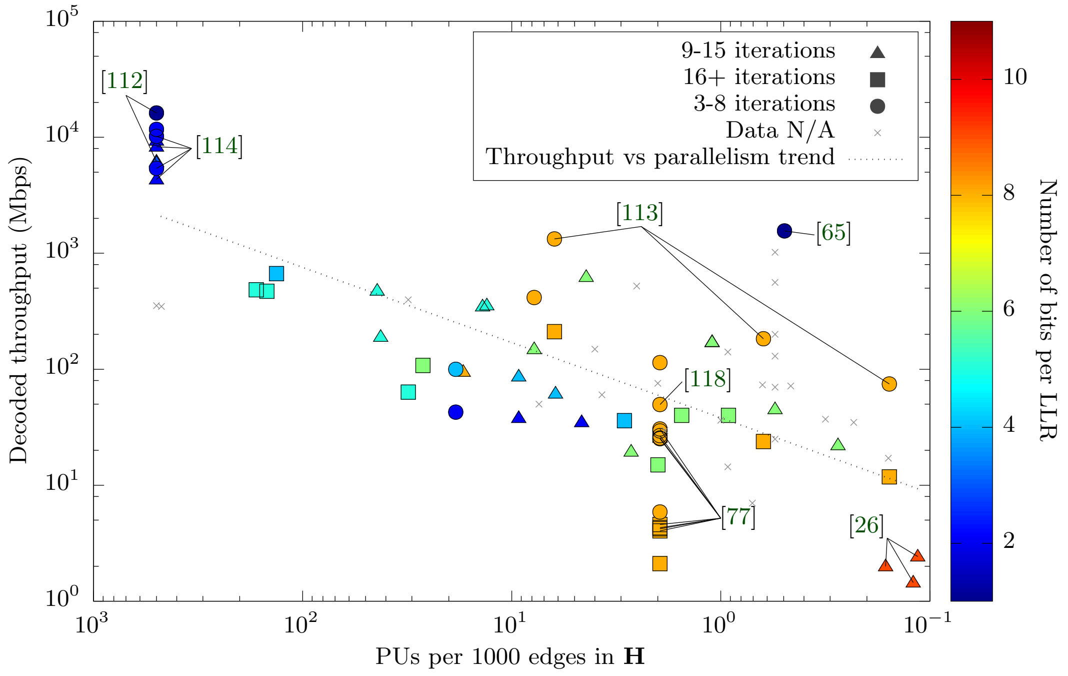


FIGURE 3.3: Factors affecting the processing throughput

using few bits, few iterations, a large degree of parallelism and operate on the basis of the Min-Sum Algorithm (MSA) [55]. The fact that the MSA can facilitate a higher processing throughput than more complicated alternatives such as the Sum-Product Algorithm (SPA) [54] is also demonstrated by comparing the results of [118] and [77], which present two very similar designs that vary in algorithm. The design in [77] suffers from a 4-5 Mbps processing throughput drop compared to [118], caused by its employment of the SPA instead of the MSA, as well as by using a non-uniform quantisation scheme for the LLRs.

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

3.3.2.2 Processing latency

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

Some of the decoders considered, such as those of [65] and [27], process multiple frames in parallel by instantiating several independent copies of the decoder on the same FPGA. In these cases the total processing throughput and resource requirement could be divided by the number of decoders, in order to produce results that correspond to the processing latency of an equivalent design that only considers one frame at a time. However, other designs, such as [67], process multiple frames by making use of spare time within the decoding schedule, with the result that the hardware cost does not increase linearly with the processing throughput. Owing to this, it is not possible to normalise the data to only consider the processing throughput and hardware resources required for decoding one frame at a time, so the processing latency cannot be fairly inferred.

3.3.2.3 Hardware requirements

Unsurprisingly, the major contributing factor to the hardware resource requirement of an FPGA-based LDPC decoder design is its degree of parallelisation, as shown in Figure 3.4. Additionally, Figure 3.4 shows that the number of bits employed per LLR and the number of edges employed in the parity check matrix also have some influence on the hardware resource requirement, though the effects of these parameters are quite varied. This may be attributed to the difficulty of accurately comparing the hardware resource requirements of different designs, as well as suggesting that other factors are involved. It is however noticeable that there is a general reduction in the number of bits per LLR employed in designs with increased parallelism. This may be explained by the explosion in routing complexity upon increasing the number of PUs, which would be exacerbated by the requirement for data buses having large operand widths.

The turquoise circles corresponding to the designs of [27] towards the bottom of Figure 3.4 seemingly have a much larger hardware resource requirement than would be expected, considering the number of processing units, the number of PCM edges and the number of bits employed per LLR. However, this is due to the fact that this publication uses multiple parallel copies of each decoder to increase the throughput, so this measurement cannot be considered as representative. Meanwhile, some of the other significant points that lie below the trend line belong to [21]. This design supports run-time flexibility over a suite of custom LDPC PCMs, which requires dynamic routing and processing structures that can adapt to different code parameters on the fly, hence requiring a higher quantity of hardware resources.

The results of [118], [64] and [77] all sit above the trend line, despite employing a large number of bits per LLR, as well as a moderate PCM size. This may be partially attributed to their implementation of quasi-cyclic LDPC codes, using partially-parallel architectures, leading to a very efficient use of hardware resources. Additionally, the smallest hardware resource requirement of these designs is achieved by one that uses

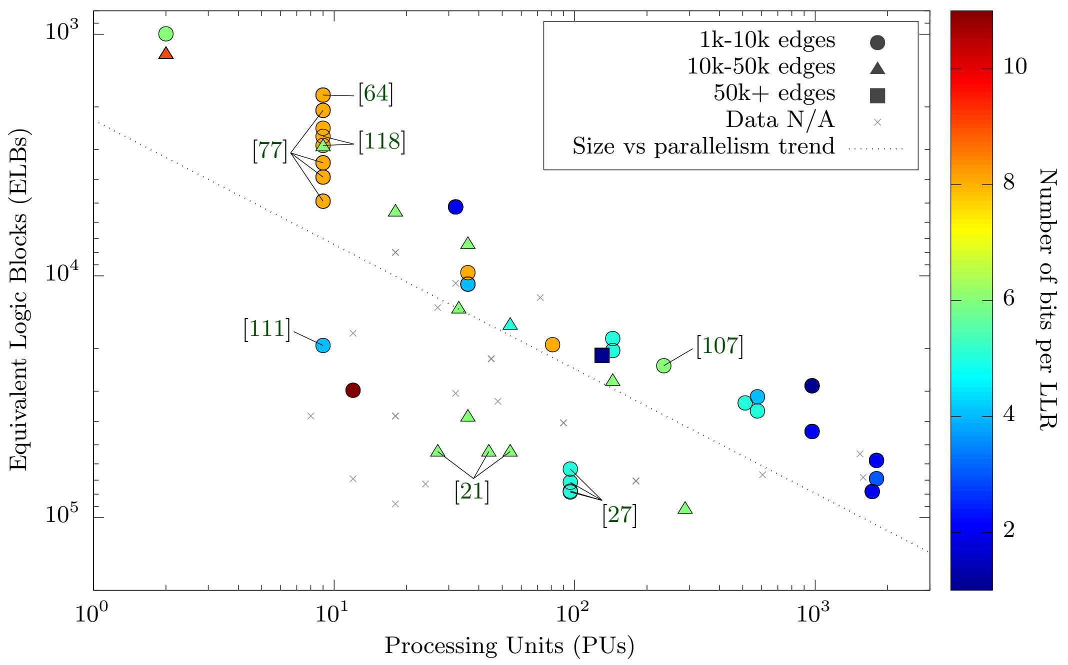


FIGURE 3.4: Factors affecting the hardware requirements

the MSA rather than the SPA, illustrating that this algorithm requires fewer hardware resources.

The design of [111] requires more FPGA resources than the trend line would suggest, which is remarkable considering its small PCM and number of bits per LLR. At first glance this may be attributed to its use of the uncommon array-based LDPC code. However, the design of [107] also uses an array-based code but sits above the trend line, despite employing a large number of bits per LLR and a large PCM. On closer inspection, it can be observed that the design of [111] employs a simple FPGA from an old generation, suggesting that its comparably large hardware resource requirement stems from inefficient FPGA synthesis.

3.3.2.4 Transmission energy efficiency and bandwidth efficiency

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

For each FPGA-based LDPC decoder considered, the theoretical Discrete-input Continuous-output Memoryless Channel (DCMC) capacity [131] was calculated, with consideration of the coding rate, modulation type, and channel model employed. This was then subtracted from the value recorded in Table 3.1 for the specific E_b/N_0 at which a low BER is achieved, in order to quantify the performance loss imposed by implementation factors. Here, a low performance loss is achieved by a decoder that can function very close to the theoretical limit, demonstrating that it is very good at correcting errors and very efficient in terms of transmission power and bandwidth.

Almost all of the publications considered characterised the error correction performance of their FPGA-based LDPC decoder designs using Binary Phase-Shift Keying (BPSK) modulation for transmission over an Additive White Gaussian Noise (AWGN) channel. This allows the BER performance of these designs to be presented together graphically, as shown in Figure 3.5. Here, the plotted line represents the DCMC capacity, while each plotted point corresponds to a different considered decoder design. The performance loss associated with each point may be obtained as its horizontal distance from the DCMC capacity curve. It can hence be seen that despite requiring drastically different E_b/N_0 levels to achieve the same BER, the designs of [21,26,73,105] can all be considered to offer a strong error correction performance, when their bandwidth efficiency is also taken into consideration. Figure 3.5 also illustrates that the designs of [22,67,112,114,117,121] are comparatively poor at correcting errors, and therefore have a low transmission energy efficiency. This is at least partly due to the fact that these designs trade off their error correction performance against other desirable characteristics, as will be explained below. Note that this analysis can be readily extended to LDPC decoders designed for other modulation schemes or channel models. This may be achieved by plotting the corresponding DCMC capacity curve and characterising the error correction performance with respect to this bound.

It is well-known that LDPC codes having longer message word lengths k are capable of performing closer to the DCMC capacity [9]. Furthermore, a higher performance loss occurs for more sparse PCMs, since these have fewer edges over which to transfer information during the decoding process. Motivated by this, Figure 3.6 plots the performance loss of each design versus the number of edges in its PCM $\mathbf{H}$, combining the message word length k with the complexity of the factor graph. As shown in Figure 3.6, the number of edges in the PCM $\mathbf{H}$ is the largest contributing factor to the error correction performance loss. As may be expected, the performance loss is also influenced by the number of iterations performed and the number of bits used per LLR, as shown in Figure 3.6. It may be observed that designs like those of [112], [65] and [117] perform poorly compared to the trend line, owing to their employment of a small number of bits per LLR or iterations. By contrast, a good performance may be observed for designs employing a large number of both, such as [113].

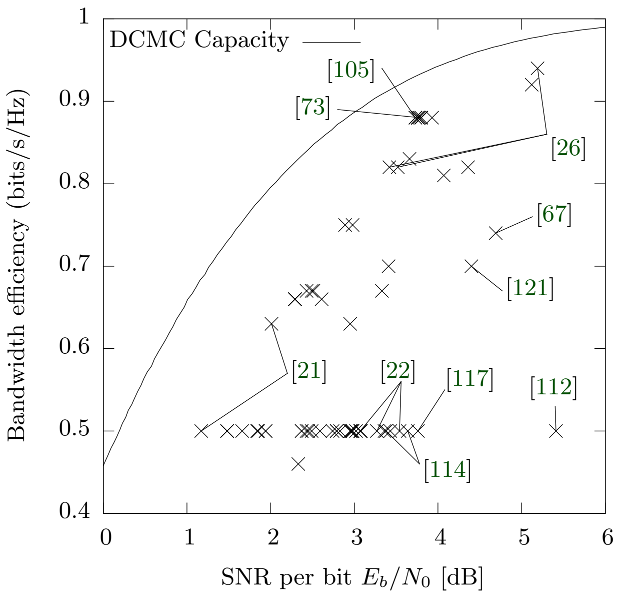


FIGURE 3.5: Decoder performance loss from capacity

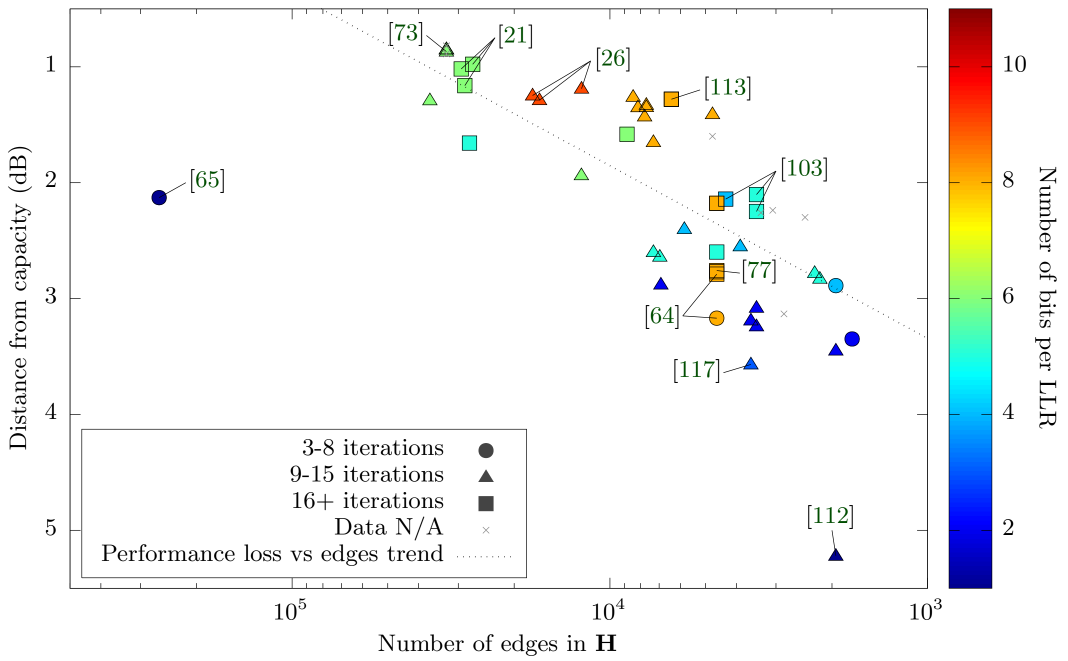


FIGURE 3.6: Factors affecting the error correction performance

The specific code construction principles used can explain some of the unexpected results seen in Figure 3.6. The design of [103] performs closer to the DCMC capacity than would be expected from a general decoder using the same small number of iterations and bits per LLR. However, the Progressive Edge Growth (PEG) algorithm explained in Section 2.3.5 was used to construct the LDPC code it uses, increasing its error correction capability at the cost of producing an unstructured factor graph, which is not optimised for hardware implementation. A “cycle elimination algorithm” was used in [21] to similar effect, while the designs of [26] achieve high performance due to the completely unstructured **H** matrix used.

The lowest error correction performance loss is achieved by the design of [73], which uses a structured quasi-cyclic code. In addition, this design also uses a sophisticated non-uniform quantisation scheme for the representation of LLRs, it employs a moderate number of bits per LLR and iterations, as well as implementing the full SPA. By contrast, the designs of [77] and [64] operate further away from capacity than may be expected, which is due to their use of the MSA.

3.4 Design recommendations

Taking advantage of the experience and lessons learned during the compilation of the previous sections, this section presents an overview of the future development effort required of designers of FPGA-based LDPC decoders. Firstly, Section 3.4.1 provides a guide to the stages involved in designing an FPGA-based LDPC decoder. Following this, Section 3.4.2 then provides a set of recommendations for future publications which will facilitate more comprehensive comparisons amongst FPGA-based LDPC decoders in the future.

3.4.1 Recommended design methodology

As discussed above, the complex relationships between the parameters and characteristics of FPGA-based LDPC decoders imply that it is not possible to identify a single design which is superior to all others in every way. Having said this, the flowchart presented in Figure 3.7 outlines a recommended series of stages for a prospective designer to complete, as a means of assisting their design process. The bullet points accompanying each stage list some of the key issues to be considered whilst completing each design element. More details about these issues can be found throughout Sections 2.3 — 2.5.

3.4.2 Recommendations for enhancing future comparisons

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

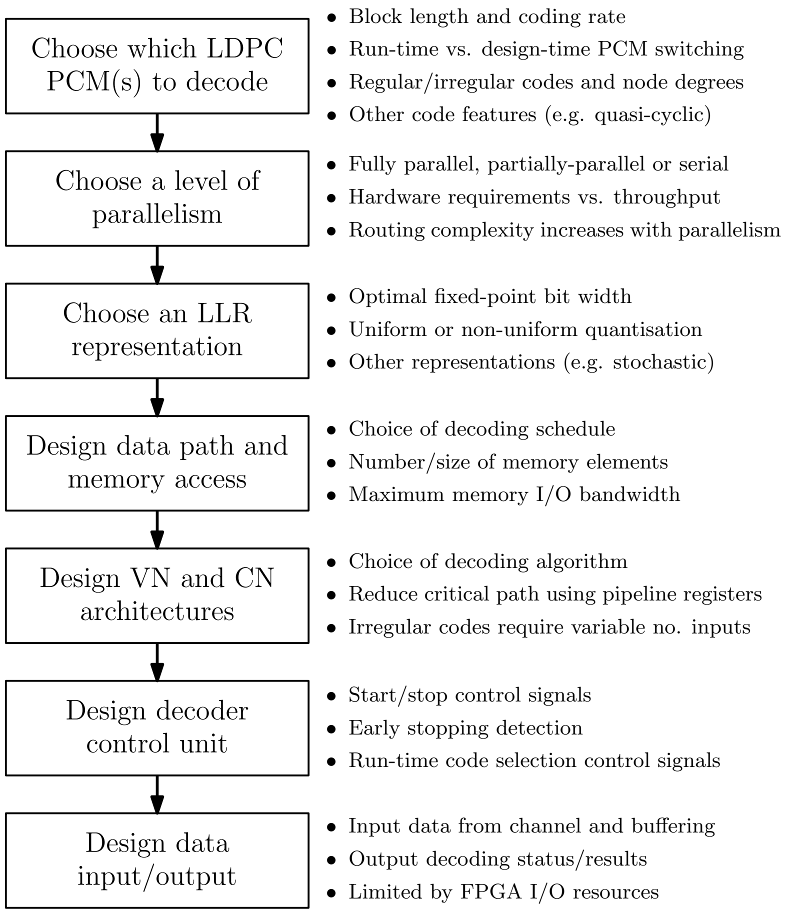

Choose which LDPC PCM(s) to decode
• Block length and coding rate
• Run-time vs. design-time PCM switching
• Regular/irregular codes and node degrees
• Other code features (e.g. quasi-cyclic)
Choose a level of parallelism
• Fully parallel, partially-parallel or serial
• Hardware requirements vs. throughput
• Routing complexity increases with parallelism
Choose an LLR representation
• Optimal fixed-point bit width
• Uniform or non-uniform quantisation
• Other representations (e.g. stochastic)
Design data path and memory access
• Choice of decoding schedule
• Number/size of memory elements
• Maximum memory I/O bandwidth
Design VN and CN architectures
• Choice of decoding algorithm
• Reduce critical path using pipeline registers
• Irregular codes require variable no. inputs
Design decoder control unit
• Start/stop control signals
• Early stopping detection
• Run-time code selection control signals
Design data input/output
• Input data from channel and buffering
• Output decoding status/results
• Limited by FPGA I/O resources

FIGURE 3.7: Stages to consider when designing an FPGA-based LDPC decoder

- provide values for every parameter and characteristic presented in Table 3.1, or at least be explicit about how they might be derived;
- ensure that all presented characteristics correspond to the same set of parameters, and if more than one parameter set is employed for demonstrating the flexibility of the design, include an equal number of full characteristic sets;
- explicitly state whether the processing throughput is encoded or decoded, and provide the formula used for calculating it;
- state the processing latency of the decoder, or signify that it can be derived simply from the processing throughput and message word length k;
- provide BER simulation curves (ideally obtained using the physical hardware), plotting the results against E_b/N_0 [dB] and explicitly stating the channel model and modulation type used, preferably BPSK modulation for transmission over an AWGN channel;
- if possible, provide multiple BER plots for different maximum numbers of iterations;

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

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

3.5 Conclusion

In this section, we have comprehensively surveyed and assessed the state of the art in FPGA-based LDPC decoder designs. In doing so, we have presented and evidenced the fundamental trade-offs between the various specified parameters and measured characteristics of these decoders, which will be of significant value to guide future design efforts. We have also been able to provide a detailed set of recommended design practices to simplify the implementation and subsequent characterisation of future FPGA-based LDPC decoders.

Furthermore, performing the analysis described above has enabled us to identify several opportunities for further research and development in the field of FPGA-based LDPC decoders, which are currently under-represented within the published literature and available commercial IP. Perhaps the biggest gap illustrated by the trade-offs described in Section 3.3.1 is for high-speed decoders having run-time flexibility and low hardware resource cost. Motivated by this, Chapter 4 presents our proposed FPGA-based LDPC decoder architecture which may be implemented to have run-time flexibility for any set of QC LDPC codes.

Additionally, the stochastic decoders presented in this report, [62] and [84], performed well in terms of processing throughput, BER performance, and hardware requirements, but without any run-time flexibility. Stochastic designs are associated with their own set of advantages and challenges, whereby the serial transmission of messages between processing nodes facilitates a lower hardware resource requirement, but requires a higher number of decoding iterations to achieve convergence. Several ASIC implementations of stochastic LDPC decoders exist [91, 93, 95, 97], however the lack of published stochastic FPGA-based LDPC decoders marks this as a possible area for future investigation. Motivated by this, in Chapter 5 we present a novel stochastic FPGA-based LDPC decoder architecture, which is the first of its kind to exhibit multi-standard run-time flexibility.

The proposed decoder architectures have been designed in such a way that specific parametrised decoder instances may be automatically generated by the offline design flow presented in Chapter 6. This allows the supported set of QC LDPC codes and the target level of parallelism to be considered as user-input variables, which can be specified at design-time. This design flow hence additionally grants the proposed architectures the property of automated design-time flexibility over the trade-offs involving throughput, hardware resources, and flexibility. The error correction performance of the implemented decoders may then be controlled through the trade-offs with processing throughput and hardware resources, via the number of iterations and the choice of LDPC code respectively, as discussed in Section 3.3.2.4.

Chapter 4

A flexible fixed-point Quasi-Cyclic LDPC decoder architecture

4.1 Introduction

The analysis presented in Chapter 3 reveals a clear gap in the state of the art in FPGA-based LDPC decoder designs, namely for architectures which are capable of switching between the decoding of messages associated with multiple different LDPC Parity-Check Matrices (PCMs) at run-time. Run-time flexibility has huge advantages for commercial applications, since it is required for a decoder to dynamically support the variety of different PCMs within a particular communications standard, such as those used by WiFi [3], WiMAX [4], and DVB-S2 [5], without requiring the extra time and technical intervention that is involved in re-designing and re-programming an FPGA. Further to this, flexible decoders can adapt automatically depending on the channel conditions [132], allowing reliable communications to be maintained by dynamically adapting the spectral efficiency and frame length. Run-time flexibility can also be useful for research purposes [26], eliminating the requirement for an FPGA to be re-synthesised when testing multiple different PCMs. It is therefore desirable to have a decoder design which may exhibit run-time flexibility over a selection of LDPC PCMs, within one code family or even among several different families, for example from different communication standards.

As discussed in Section 3.3.1, decoders having a fully-serial architecture [26, 121] naturally offer flexibility with little additional hardware overhead, but suffer from low processing throughputs, which typically do not meet the requirements of communication

This chapter is based on the following publications:

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “A Flexible FPGA-Based Quasi-Cyclic LDPC Decoder,” in *IEEE Access*, vol. 5, pp. 20965–20984, mar 2017.

standards. Meanwhile, the additional hardware required to implement a fully-parallel decoder [59, 117] with a high degree of run-time flexibility renders this approach impractical, regardless of their capacity for high processing throughputs. This suggests that partially-parallel decoder architectures [61] have the greatest potential for run-time flexibility [50], while achieving low hardware resource requirements and high processing throughputs. In particular, partially-parallel LDPC decoder architectures are particularly suited to the Quasi-Cyclic (QC) LDPC PCMs that are employed in many modern communication standards [27].

Against this background, in this chapter we present a novel FPGA-based LDPC decoder architecture having run-time flexibility for any set of QC PCMs. Furthermore, the design of this architecture is presented in generalised terms, which does not depend on any characteristic of any specific PCM set. Accordingly, the chosen PCMs to support may originate from one family or several different families, and may vary in any of their parameters, including the code length n, the QC expansion factor z, the coding rate R, the row/column degrees d_c and d_v, and the structure of the QC base matrix $\mathbf{H}_b$. The generalised format of the proposed architecture then facilitates the rapid development and synthesis of specific decoder instances having the proposed architecture. The run-time flexibility of these decoders is sufficiently high that they may switch between the use of any of their supported PCMs within a single clock cycle.

The structure of this chapter is as follows. Firstly, Section 4.2 expands on the background information regarding the properties of QC codes given in Chapter 2, as well as describing how these properties can be exploited to generate an efficient partially-parallel decoder design. Section 4.3 then describes the overall structure of the proposed fixed-point LDPC decoder architecture, paying particular attention to the features that enable run-time switching of PCMs. Following this, Section 4.4 presents the characteristics of several implementations of the proposed architecture, comparing its performance alongside several benchmarkers according to the guidelines set out in Chapter 3. Finally, Section 4.5 provides some concluding remarks and recommendations for further work.

4.2 Partially-parallel decoding of quasi-cyclic codes

The majority of communications standards employing LDPC codes for error correction utilise codes having a QC construction [45], which have several properties that may be exploited by a partially-parallel decoding architecture. This background section expands on the brief introduction given to these topics in Chapter 2. Firstly, Section 4.2.1 provides a description of the structure of QC codes, and the resultant properties of their PCMs. Section 4.2.2 then reviews the nature of partially-parallel decoding architectures, discussing various ways in which these properties may be utilised in an efficient hardware design.

4.2.1 Quasi-cyclic LDPC codes

As discussed in Section 2.3.3, the PCM of a QC LDPC code is a specially structured realisation of the generalised unstructured PCM **H**, which is used to completely define any LDPC code. The number of rows in an LDPC PCM **H** 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 then defined as $R = 1 - {}^{m}\!/_{n}$, where $0 < R < 1$. Each row and column contains a number of non-zero elements, which 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 degrees, respectively. Since the number of non-zero elements must be the same when viewed as columns or rows, $m \times d_c = n \times d_v$, giving $d_c > d_v$. To illustrate the above, an example of a PCM **H** with $n = 18$ and $m = 9$ would be

$$\mathbf{H} = \begin{bmatrix} 0&0&0 & 0&1&0 & 0&0&0 & 1&0&0 & 0&0&1 & 0&1&0 \\ 0&0&0 & 0&0&1 & 0&0&0 & 0&1&0 & 1&0&0 & 0&0&1 \\ 0&0&0 & 1&0&0 & 0&0&0 & 0&0&1 & 0&1&0 & 1&0&0 \\ 0&1&0 & 0&0&1 & 1&0&0 & 1&0&0 & 0&0&0 & 1&0&0 \\ 0&0&1 & 1&0&0 & 0&1&0 & 0&1&0 & 0&0&0 & 0&1&0 \\ 1&0&0 & 0&1&0 & 0&0&1 & 0&0&1 & 0&0&0 & 0&0&1 \\ 0&0&1 & 0&0&0 & 0&1&0 & 0&0&0 & 0&0&1 & 1&0&0 \\ 1&0&0 & 0&0&0 & 0&0&1 & 0&0&0 & 1&0&0 & 0&1&0 \\ 0&1&0 & 0&0&0 & 1&0&0 & 0&0&0 & 0&1&0 & 0&0&1 \end{bmatrix}. \tag{4.1}$$

Note that **H** in (4.1) is irregular, with average row degree $d_c = 4.33$ and average column degree $d_v = 2.17$.

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 are presented in Section 2.4.1. The results of the row and column calculations are extrinsic messages, commonly represented in the form of Logarithmic-Likelihood Ratios (LLRs), which provide soft information regarding the likelihood of the corresponding bit being a 0 or 1, as defined in Section 2.1. When using the factor graph representation of a PCM introduced in Section 2.2.3, 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 between VNs and CNs along these edges, where new extrinsic LLRs are calculated within the VNs (which perform the PCM column calculations) and the CNs (which perform the PCM row calculations).

The PCM $\mathbf{H}$ of a QC LDPC code is completely defined by a base matrix $\mathbf{H}_b$, where each element of $\mathbf{H}_b$ represents a square submatrix in $\mathbf{H}$ of dimensions $z \times z$. 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 (4.1) 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. (4.1) may therefore be re-written as a QC base matrix $\mathbf{H}_b$ with $z = 3$, $n_b = 6$, and $m_b = 3$, according to

$$\mathbf{H_b} = \begin{bmatrix} -1 & 1 & -1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 0 & -1 & 0 \\ 2 & -1 & 1 & -1 & 2 & 0 \end{bmatrix}. \tag{4.2}$$

Note that a value of -1 in $\mathbf{H}_b$ corresponds to a null submatrix, whereas a non-negative value represents the shift value s of the identity matrix in the corresponding submatrix. As an example, $H_{b11} = -1$, indicating that the 3×3 submatrix in the top-left corner of (4.1) is a null matrix. Meanwhile, $H_{b16} = 1$, indicating that the submatrix in the top-right corner of (4.1) is an identity matrix that has been circularly-shifted to the right by $s = 1$ place. Groups of z rows or 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.

This semi-structured nature causes the expanded PCM $\mathbf{H}$ of a QC LDPC code to possess some properties that would not normally exist in generalised unstructured PCMs generated using other methods. Firstly, all of the columns or rows within the I-th block-row or J-th block-column will have the same degree, d_{cI} or d_{vJ}. Furthermore, it can be seen that no two rows or columns within a block-row or block-column will have non-zero elements in the same column or row, respectively. This is due to the fact that every non-zero element within a submatrix is positioned on its own row and column. As an example, the first row in (4.1) has non-zero elements in the 5th, 10th, 15th, and 17th columns. However, these columns are guaranteed to contain zero elements in the other rows within the first block-row, namely rows two and three.

In addition, it can be seen that the shift values of the submatrices can be used to cyclic-shift the LLRs it represents to be read as either rows or columns. For example, Figure 4.1 shows a submatrix of size $z = 3$ with a shift value of $s = 1$. The LLRs represented by the non-zero elements in this submatrix are displayed in-place as a, b, and c. Clearly, when this submatrix is viewed as rows, message a is in row 1, message b is in row 2,

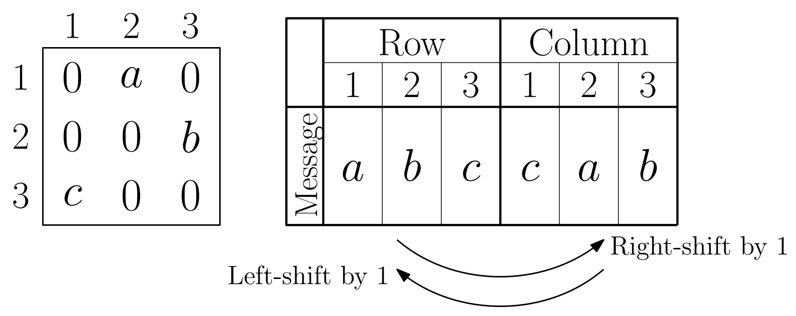


FIGURE 4.1: Demonstration of using a submatrix shift value s to alternate between row-centric and column-centric representations

and message c is in row 3. In a row-centric representation, therefore, the messages are ordered a, b, c. However, when the submatrix is viewed as columns, message a is in column 2, message b is in column 3, and message c is in column 1. The column-centric representation of the messages would therefore be c, a, b. For any shift value s, it is possible to re-order the messages from their row-centric representation to their column-centric representation by cyclic-shifting the values to the right by s, and then left-shift back by s to return again. This property may be exploited by a partially-parallel LDPC decoder architecture, as described in Section 4.2.2.

4.2.2 Partially-parallel decoder architectures

As mentioned in Section 2.5.1, partially-parallel architectures implement a number ψ of parallel Node Processing Units (NPUs), where $1 < \psi < n$. Each decoding iteration is then split into several stages, wherein ψ rows or columns of the PCM are processed simultaneously. This permits a much higher processing throughput than fully-serial architectures, which process one row or column at a time. Partially-parallel architectures also avoid the impractical hardware resource usage required by fully-parallel architectures, which process all rows or columns at the same time.

However, unlike fully-parallel architectures, it is necessary for partially-parallel LDPC decoders to temporarily store intermediate calculation results, namely the extrinsic LLRs for each row and column calculated by the NPUs over the course of the iterative decoding process, described in Section 2.4. During the decoding process, these messages may be read and written iteratively in both row-centric and column-centric manners, as required. In an FPGA-based decoder implementation, it is common to use the FPGA's built-in Block RAMs (BRAMs) to store these messages [28], with addresses dictated by the positions of the non-zero entries in the PCM $\mathbf{H}$ and stored in ROM. For unstructured codes, however, this memory addressing logic can create a prohibitively large hardware resource usage overhead [26]. Additionally, only one location in a BRAM can be addressed at a time, requiring careful organisation. The distribution of values into BRAMs must therefore be chosen such that when processing ψ rows (or columns) simultaneously, no more than one location within each column (or row) BRAM is required [133]. For unstructured codes there is no way of guaranteeing that such an arrangement will be possible.

An additional factor that must be taken into consideration when designing a partially-parallel decoder is its ability to handle irregular LDPC codes, wherein each row or column can have a different degree. One method of doing so would be to implement ψ NPUs of the maximum degree within the PCM, with the ability to "turn off" any unused inputs and outputs when processing a lower-degree row or column. For unstructured irregular codes, however, this would require a large amount of extra ROM and routing networks to store m row degrees and n column degrees, of which ψ would be read simultaneously.

The semi-structured nature of QC codes described in Section 4.2.1 above simplifies these problems in a number of ways, particularly when the parallelism ψ of the decoder is chosen to match the width z of the block-rows and block-columns of the PCM, as discussed in Section 4.3.1. Firstly, the in-memory representation of each PCM is greatly 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 knowledge of the locations and values of the non-zero entries in the reduced PCM $\mathbf{H}_b$ [134]. Secondly, by processing block-rows or block-columns in parallel, every row or column that is processed simultaneously will always have the same degree as discussed above. For irregular codes, this reduces the total number of degrees that must be stored by a factor of z, and permits the sharing of control signals for each NPU. Additionally, as 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 required at the same time. This fact is utilised further in the memory arrangement of the proposed decoder design, explained in Section 4.3.2. Finally, cyclic-shifting the messages represented by each submatrix by s to alternate between their row-centric and column-centric representations simplifies the permutation operations required between the NPUs, as described in Section 4.3.3.

4.3 The proposed 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 codes, as outlined in Section 4.1. Owing to this, the discussion of this section is presented in generalised terms, with the features of the supported QC PCMs such as 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, considered as variable parameters. Chapter 6 later presents a design flow that takes this general architecture and a set of one or more PCMs as inputs, and generates a specific LDPC decoder design by selecting desirable values for the parameters described in this section.

For the reasons stated in Sections 4.1 and 4.2, 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 Figure 4.2. The

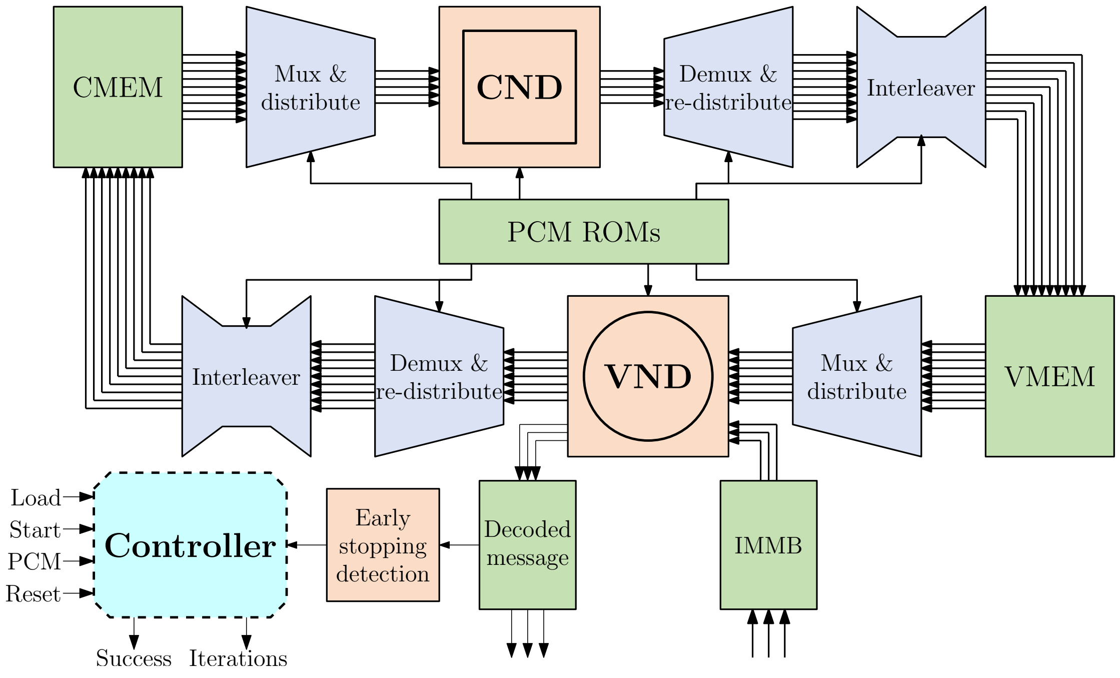


FIGURE 4.2: Block diagram of the proposed fixed-point flexible LDPC decoder architecture

decisions motivating the various aspects of this design are discussed in the following subsections. More specifically, Section 4.3.1 presents the degree of parallelism, the decoding schedule, and the PCM ROMs. Section 4.3.2 then presents the composition and access arrangements for the two main BRAM memory banks in Figure 4.2, namely VMEM and CMEM. Following this, Section 4.3.3 discusses the datapath components linking these memory banks to the VND and CND respectively, before Section 4.3.4 presents the structure of the individual node processing units, namely the VNPU and CNPU. Finally, Section 4.3.5 details the controller module, which oversees the decoding process according to the input control signals.

4.3.1 Overview of parallelised decoding in the proposed architecture

This section presents the general structure and schedule of the proposed architecture, before the specific details are described fully in the following sections. Firstly, the level of parallelism is defined and characterised mathematically. Following this, the decoding schedule is explained. Finally, the PCM ROMs depicted in Figure 4.2 are briefly introduced, allowing their function to be explained further in the relevant places throughout the rest of this section.

4.3.1.1 Level of parallelism

In order to produce 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. For this reason, as shown in Figure 4.2, the decoder has a separate Variable Node Decoder (VND) and Check Node Decoder (CND), which are designed to be operated in parallel. Doing so facilitates the addition of pipelining registers in the datapaths between the NPUs and the memories, as the updated outputs from each node will then be used as inputs by its connected nodes during their subsequent activations.

The VND contains ψ Variable Node Processing Units (VNPUs), each of which processes one column of the PCM in $t_v = 1$ clock cycle, with the result that ψ VNPUs can process ψ 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, and the additional input and output is used for intrinsic messages from the channel and the estimation of the decoded bit respectively, as described in Section 2.4.3. Implementing this maximum number of inputs and outputs is essential to ensure each VNPU can be used to decode any column in the set of supported PCMs.

Similarly, the CND contains ψ_c Check Node Processing Units (CNPUs), where $\psi_c \leq \psi$ 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. As shown in Figure 4.2, the VND and CND both have separate datapaths, which are described further in Section 4.3.3. They also have separate memories, namely VMEM and CMEM respectively, which are described further in Section 4.3.2.

Motivated by the discussion in Section 4.2, the parallelism of the proposed partially-parallel architecture reflects the dimensions z of the submatrices within the target QC PCMs. Accordingly, the parallelism factor ψ is given by

$$\psi = \left\lceil \frac{Z}{Q} \right\rceil , \tag{4.3}$$

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, as demonstrated in Sections 4.4 and 5.4. When $Q = 1$, we have $\psi = Z$, allowing the decoder to process an entire block-column in $t_{vb} = t_{vp} = 1$ clock cycle. Larger values of Q lead to a parallelism of $\psi < Z$, such that the block-columns of each PCM p having a submatrix size of z_p are processed in $t_{vb} = q_p$ clock cycles, where

$$q_p = \left\lceil \frac{z_p}{\psi} \right\rceil . \tag{4.4}$$

More explicitly, q is a parameter that varies between PCMs within the supported set, and represents how many layers of size ψ are required for each QC block-column, where $1 \leq q \leq Q$. Note also that selecting a value of $Q = Z$ leads to a parallelism factor of $\psi = 1$, which results in a fully-serial decoder architecture where each PCM p subdivides each block-column into $q_p = z_p$ layers.

The value of ψ_c is calculated as a factor of ψ, according to the relative numbers of rows and columns in the PCM having the lowest ratio of $^{n_b}/_{m_b}$, as follows. Note that the number of block-columns n_b is larger than the number of block-rows m_b, and that one decoding iteration comprises the processing of all n_b block-columns in parallel with all m_b block-rows. It can hence be seen that the number of clock cycles required to process the block-columns represents the minimum number of clock cycles required per iteration, t_i. Note also that each block-column is processed using $t_{vb} = q$ clock cycles as described above, hence $t_i = n_b \times q$ clock cycles. Next, let G_{min} represent the minimum value of G within the set of supported PCMs p, where

$$G_p = \left\lfloor \frac{n_{b,p}}{m_{b,p}} \right\rfloor . \tag{4.5}$$

The number of clock cycles t_{cb} required to process each block-row can hence be larger than t_{vb} by a factor of G_{min}, without increasing t_i. In cases where $G_{min} \geq 2$, the required number ψ_c of CNPUs within the CND may then be reduced according to

$$\psi_c = \left\lceil \frac{\psi}{G_{min}} \right\rceil . \tag{4.6}$$

More explicitly, using ψ_c CNPUs, a group of ψ 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 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 consumption of a CNPU is much larger than that of a VNPU. Due to this, a reduction in ψ_c can lead to a significant reduction in a decoder's overall hardware resource usage.

4.3.1.2 Decoding schedule

As described previously, the proposed decoder operates ψ_c CNPUs and ψ VNPUs in parallel, with each decoding iteration comprising the activation of all CNs and VNs within the factor graph. Through extensive simulations, it was determined that processing the CND and VND simultaneously in this way does not have a noticeable detrimental impact on the decoder's BER performance, when compared to more serial decoding schedules. More explicitly, when utilising the same number of decoding iterations, the simultaneous decoding schedule proposed here produces BER results equivalent to the performance of the flooding schedule [42], but worse than the LBP schedule [43]. However, as described in Section 2.5.3, adopting the LBP schedule places limitations on the potential degree of pipelining, limiting the attainable throughput.

Furthermore, our simulations revealed that the order in which block-rows and block-columns are processed makes little difference to the decoder's BER performance. For simplicity, the proposed decoder is therefore designed to process adjacent block-rows and block-columns sequentially within each decoding iteration. More explicitly, the VND spends the first t_{vb} clock cycles in each iteration processing the columns within the first block-column, then moves on to the second block-column for the next t_{vb} clock cycles, and so on. Simultaneously, the CND spends the first t_{cb} clock cycles processing the rows within the first block-row, then moves on to the second block-row for the next t_{cb} clock cycles, and so on.

The discussion provided above is exemplified in Table 4.1, by demonstrating the decoding schedule for one iteration of the PCM presented previously in (4.1), where $n_b = 6$, $m_b = 3$, $z = 3$, $n = 18$, and $m = 9$. In this example, Q is chosen to be equal to 1, giving

TABLE 4.1: Decoding schedule for one iteration of an example QC PCM

Clock cycle	**Block-column**	**Columns**	**Block-row**	**Rows**
1	V_{B1}	V_1, V_2, V_3	C_{B1}	C_1, C_2
2	V_{B2}	V_4, V_5, V_6	C_{B1}	C_3
3	V_{B3}	V_7, V_8, V_9	C_{B2}	C_4, C_5
4	V_{B4}	V_{10}, V_{11}, V_{12}	C_{B2}	C_6
5	V_{B5}	V_{13}, V_{14}, V_{15}	C_{B3}	C_7, C_8
6	V_{B6}	V_{16}, V_{17}, V_{18}	C_{B3}	C_9

$\psi = 3$, and allowing the VND to decode a complete block-column of $z = 3$ columns within $t_{vb} = Q = 1$ clock cycle. Therefore, the number of clock cycles required per iteration is given by $t_i = t_{vb} \times n_b = 6$. Furthermore, we have $G_{min} = G = \lfloor {}^{n_b}/_{m_b} \rfloor = 2$, allowing the CND to utilise $t_{cb} = Q \times G = 2$ clock cycles per block-row, requiring only $\psi_c = 2$ CNPUs. Table 4.1 presents the schedule of which parts of the PCM are processed during each clock cycle. Here, V_{BJ} represents the J-th block-column, where $1 \leq J \leq n_b$, while V_j represents the j-th column, where $1 \leq j \leq n$. Likewise, C_{BI} represents the I-th block-row, where $1 \leq I \leq m_b$, while C_i represents the i-th row, where $1 \leq i \leq m$. Note that this selection of parameters results in only one of the $\psi_c = 2$ CNPUs activating in clock cycles 2, 4, and 6. Whilst seemingly inefficient, this is necessary during the decoding of any PCM in which the number of VNPUs ψ or CNPUs ψ_c in the decoder are not exact factors of the QC submatrix size z. This is to ensure that only rows from within one block-row, or columns from within one block-column, are processed at the same time, for the reasons stated in Section 4.2.

4.3.1.3 PCM ROMs

The run-time flexibility of the proposed architecture is enabled through the use of several ROM blocks, which store the values of each QC PCM in a hardware-optimised form. These ROMs are used extensively within the datapaths for the VND and CND to control the routing and shifting of *a priori* and extrinsic LLRs between memory banks and NPUs.

For each PCM, there are two 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, and one for the CND which views the base PCM as m_b rows of d_c values. Within each set, there are three ROMs. The first is referred to as **SELECT**, which stores the locations of the non-null submatrices within the PCM. The shift values of each of these non-null submatrices are stored in the **SHIFT** ROMs. Finally, the binary values stored in the **NONZERO** ROMs are 1 when the corresponding values in the SELECT and SHIFT ROMs correspond to a non-null submatrix, or 0 otherwise.

4.3.2 BRAM arrangement

As discussed in Section 4.2.2, FPGA-based partially-parallel LDPC decoders may take advantage of the FPGA's built-in BRAMs[2], in order to store the extrinsic LLRs calculated by the VND until they are needed by the CND, and vice versa. The proposed architecture represents all LLRs using $W = 4$-bit two's complement integers, as recommended in [19]. The diagram in Figure 4.2 shows that the proposed architecture splits

[2]Note that the term "BRAM" can mean different things within the field of FPGAs, namely physical memory blocks of varying sizes and configurations. However, both Intel's and Xilinx's synthesis tools permit the transparent joining of two small BRAMs to function as one memory block of double the size [28]. For this reason, the following discussion uses the term BRAM to refer to a discrete memory block of variable size, without making any distinction regarding its possible composition of several smaller physical BRAMs.

this extrinsic memory into two sections, named VMEM and CMEM. During each clock cycle, the VND will read up to $\psi \times D_V$ 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 $\psi_c \times D_C$ 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 memory management described below.

In the proposed architecture, let the maximum value of n_b within the set of supported PCMs be denoted as N_B, and the maximum value of m_b within the set of supported PCMs be 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 $\psi \times W$. 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 to be stored in one BRAM word. In the simplified case where $\psi_c = \psi$, it may be observed that the ψ 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 ψ 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 both 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, which can be found on most modern FPGAs [75].

When decoding a set of PCMs in which $D_C < N_B$, the number of required BRAMs could in theory be reduced from N_B to D_C by implementing further switchable routing networks. However, these networks would further increase both the logic resource consumption of the design and the lengths of the paths between the NPUs and the memories, reducing the maximum clock frequency and thus the processing throughput. It was therefore decided to use N_B BRAMs in each memory for all cases, reducing the decoder's hardware resource consumption and critical path length at the expense of possibly increasing the number of BRAMs required. It will be shown in Section 4.4 that the critical path length and the consumption of logic resources are the key factors that limit the efficacy of decoders having the proposed architecture, supporting this decision.

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. More specifically, despite the use of N_B BRAMs in both the VMEM and the CMEM, each BRAM cannot simply hold all of the submatrices in one block-column, since this would require multiple addresses to be read simultaneously by the VND. In order to address this, we propose the layout of Figure 4.3, which exemplifies one of the memories for a decoder designed to support the example PCM of (4.2). 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-columns					
		1	**2**	**3**	**4**	**5**	**6**
Block-rows	**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

FIGURE 4.3: VMEM and CMEM locations of the submatrices of an example QC PCM

In the arrangement of Figure 4.3, it may be observed that the LLRs required for block-row C_{BI} are stored in address I of all N_B BRAMS, while the LLRs required for block-column V_{BJ} are stored in addresses 1 to M_B of BRAMs $(J, J+1, \ldots, J+(M_B-1))$-modulo N_B. Note, however, that every time LLRs are read from the VMEM, the output still produces ψ LLRs from all N_B BRAMs, despite the fact that only M_B BRAMs store the LLRs associated with the active block-column. The selection of which BRAM outputs to use for the active block-column is performed in the VND datapath, detailed in Section 4.3.3.1.

4.3.3 Datapath

The following discussions detail the VND and CND datapaths, in Sections 4.3.3.1 and 4.3.3.2 respectively. Following this, Section 4.3.3.3 then provides more detail to the programmable Barrel Shifters (BSs), which facilitate the flexible routing between the NPUs and BRAMs.

4.3.3.1 VND datapath

The input to the VND datapath is provided by the N_B groups of ψ W-bit LLRs from the VMEM, as described in Section 4.3.2. These must be routed into the ψ VNPUs, which each have D_V inputs. The datapath must then take the ψ groups of D_V outputs from the VND and route them back into N_B groups of ψ messages for writing into the CMEM. An example of the proposed VND datapath is provided in Figure 4.4, for the case of a decoder designed to support a single example code[3], having parameters $\psi = 8$, $N_B = 7$, 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 $\psi = 8$ 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 Figure 4.4. Note that the intrinsic LLR inputs and the decoded bit outputs for each VNPU have been omitted for simplicity. These will be detailed with the discussion of the VNPU architecture in Section 4.3.4.

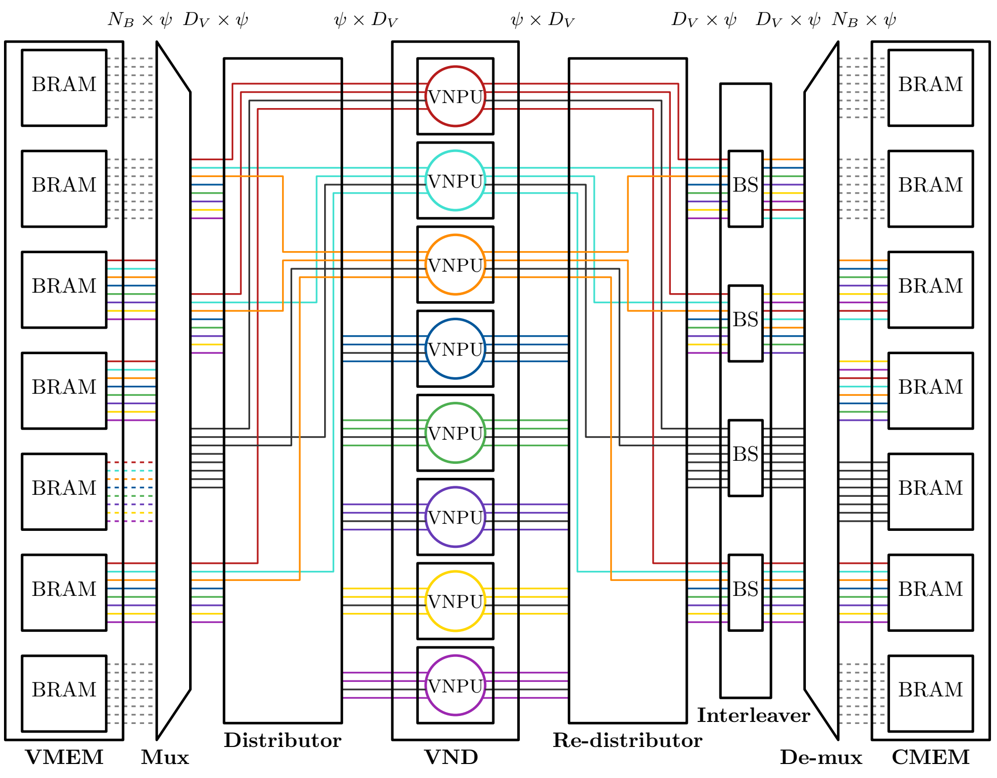


FIGURE 4.4: Example of a VND datapath

Using the memory organisation system of Section 4.3.2, it can be seen that the 3rd block-column within the PCM is being processed in Figure 4.4, since BRAMs 3, 4, 5, and 6 of the VMEM and CMEM contain the active submatrices. The programmable multiplexer, labelled **Mux** in Figure 4.4, uses the values stored in the SELECT ROM to

[3]The example PCM of (4.1) could not be used again here, as its small values of the QC submatrix dimension z and the column degrees d_v would obfuscate many of the salient points being illustrated in Figure 4.4.

ensure 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 $d_v = 3$, since a null submatrix is stored in BRAM 5, as represented by the dashed lines at its output. It is therefore also the role of the multiplexer to ignore the BRAM contents that correspond to null submatrices, which are identified by the 0s stored in the NONZERO ROM. Instead of forwarding the unwanted BRAM contents to the VND, the multiplexer instead provides a value on the connected wires that represents the inputs being "turned off". For a VNPU, this "off" value is equal to 0, as will be explained further in Section 4.3.4.1. In Figure 4.4, this is represented by the black lines emanating from the 3rd output of the multiplexer.

Subsequently, the D_V groups of ψ LLRs output from the multiplexer are distributed into ψ groups of D_V LLRs on a per-VNPU basis by the **Distributor**. Note that the action of the distributor is only shown in Figure 4.4 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 4.3.4.

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. However, before the messages can be written into the CMEM, they must be cyclic left-shifted by the D_V programmable BSs in the **Interleaver**, as will be described in Section 4.3.3.3. The values of these shifts are stored in the SHIFTS ROM. It can be seen from analysis of Figure 4.4 that the top BS left-shifts the LLRs by the shift factor $s_1 = 2$, whilst BSs 2 and 4 have shift values $s_2 = 6$ and $s_4 = 0$, respectively. A programmable de-multiplexer (labelled **De-mux**) then performs the inverse operation to the multiplexer at the start of the datapath, again using the indices stored in the SELECT ROM, to place the output values at the input ports for the correct D_V BRAMs within the CMEM.

4.3.3.2 CND datapath

The datapath for the CND performs largely the same functions as the VND datapath, albeit with some additional complexities. As described in Section 4.3.1, ψ_c CNPUs are employed to process ψ rows over $t_{cp} = G_{min}$ clock cycles, where G_{min} and ψ_c are defined in (4.5) and (4.6), respectively. The spread of the operation over $t_{cp} = G_{min}$ clock cycles is achieved through the use of D-type Flip-Flops (DFFs) and additional multiplexers embedded in the datapath between the CMEM and the CND, in order to ensure the CND is only presented with ψ_c inputs at a time. De-multiplexers and additional DFFs are then employed between the CND and the VMEM so that all ψ 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 [3], WiMAX [4], and WiGig [135]) 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 typically have larger row degrees d_c. Owing to this, high-rate codes tend to have a low number of rows, each having high degrees; meanwhile, 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, though this cost is offset by the smaller number m_b of block-rows present in these high-rate PCMs. This optimisation halves the number D_C of inputs and outputs required by each CNPU within the CND. The internal operation of two low-degree CNPUs that have been linked in order to provide the functionality of one high-degree CNPU is explained in Section 4.3.4.

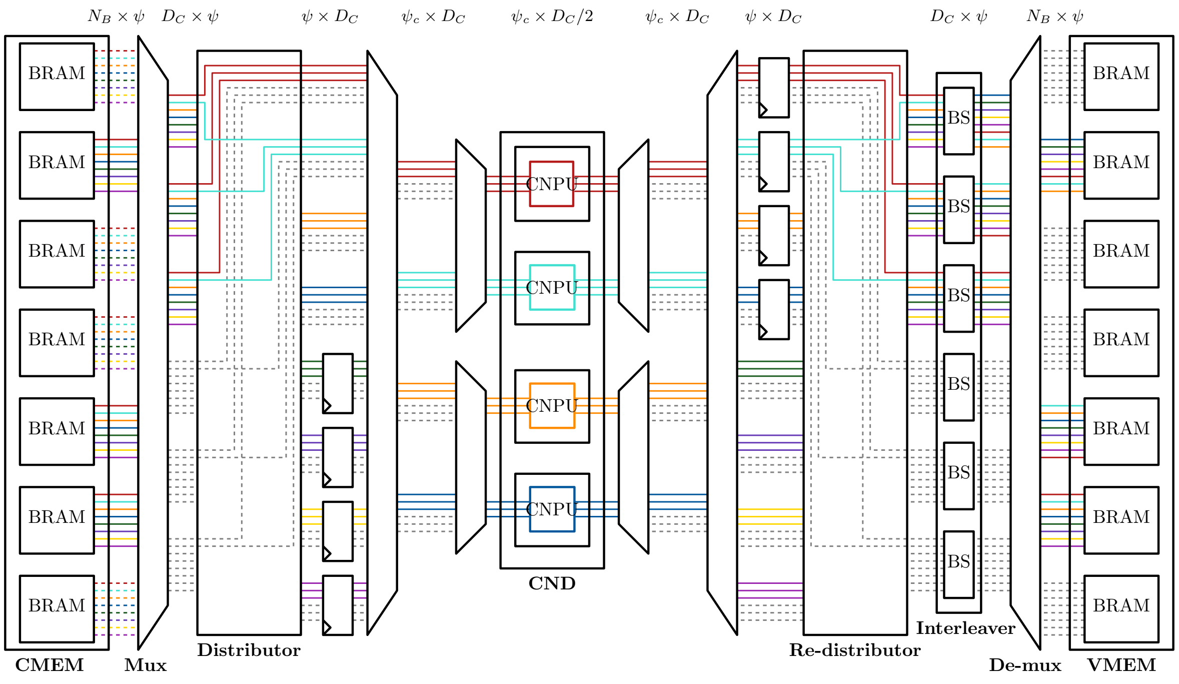
In order to support this functionality in the CND datapath, an additional boolean parameter L is introduced for each supported PCM, as well as an additional boolean decoder parameter F. For each PCM p, the parameter $L_p = 1$ indicates that each row should be processed using two linked CNPUs. It is calculated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

$$L_p = \begin{cases} 1, & \text{if } G_p \geq G_{min} \times 2 \\ 0, & \text{otherwise} \end{cases}, \tag{4.7}$$

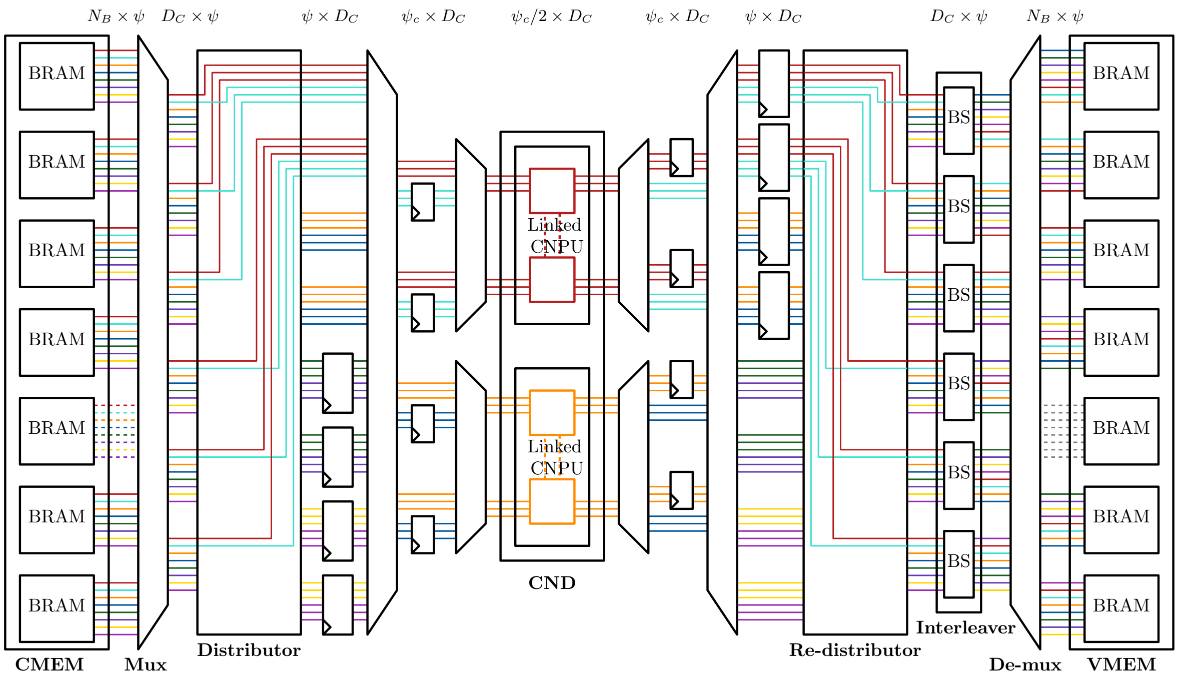
where G_{min} is defined in Section 4.3.1 as the minimum value of G_p for all PCMs p within the chosen set. However, as will be shown in Section 4.3.4, the hardware required to facilitate the linking of two CNPUs slightly increases the hardware resource consumption and the critical path length of the 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 $L_p = 1$, in which case we set the boolean decoder parameter $F = 1$.

An example of these optimisations is presented in the flexible CND datapath of Figure 4.5 for a decoder with $F = 1$, when used to implement 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 $\psi = Z = 8$, and thus $\psi_c = 4$, according to (4.6). Figure 4.5(a) represents the lower $R = {}^1/_2$-rate code where $L_p = 0$, meaning that $\psi_c = 4$ CNPUs of degree $D_C = 3$ are employed to process the $\psi = 8$ rows within $t_{cb} = t_{cp} = G = 2$ clock cycles.

Conversely, Figure 4.5(b) represents the higher $R = {}^3/_4$-rate code, where $L_p = 1$, meaning that the $\psi_c = 4$ CNPUs are combined to form ${}^{\psi_c}/_2 = 2$ linked CNPUs of degree $2D_C = 6$, 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-rate code, this does not produce any change in the number of clock cycles t_i required per decoding iteration. Similarly to Figure 4.4, the eight different colours each represent the $\psi = 8$ rows of the active block-row, with each line representing one LLR. The flow of data is from the CMEM on the left, through the CND in the middle, to the VMEM on the right. The total number of LLRs passed by each stage is displayed at the top of Figure 4.5.



(a) Low-rate codes with $L = 0$



(b) High-rate codes with $L = 1$

FIGURE 4.5: Example of a CND datapath with conditional CNPU sharing

Looking firstly at the low-rate configuration in Figure 4.5(a), it may be observed that only $D_C = 3$ 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 Figure 4.5, must use the values stored in the SELECT ROM to ensure that these inputs form the top half of its $2D_C = 6$ outputs, while its bottom $D_C = 3$ outputs are effectively "null" or "don't care" values, represented by dashed grey lines in Figure 4.5(a). Similarly to the VND datapath explained in Section 4.3.3.1, when the active block-row has a degree $d_c < D_C$, this multiplexer would also "turn off" the outputs from the extra null submatrices, which may be identified using the NONZERO ROM. For a CNPU, this "off" value corresponds to an output value equal to $(2^{W-1} - 1)$, as will be explained further in Section 4.3.4.

In this configuration, the operation of the **Distributor** is the same as that in the VND datapath, with each of its ψ output groups of $2D_C = 6$ LLRs comprised of $D_C = 3$ extrinsic messages (or "off" values) followed by D_C "null" values. Note that the action of the distributor is only shown in Figure 4.5 for two CNPUs, for the sake of avoiding obfuscation. The first DFFs and multiplexer after the distributor perform the function of reducing the $\psi = 8$ groups by a factor of $G = 2$ to form $\psi_c = {}^{\psi}/_{G} = 4$ groups. It can be seen that in the first clock cycle, the multiplexer will select the top $\psi_c = 4$ rows (red, turquoise, orange, and blue), which will be processed and then latched by the DFFs before the re-distributor. In the second clock cycle, the multiplexer will select the latched values of the following $\psi_c = 4$ rows (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 lines from each ψ_c group, presenting ψ_c groups of D_C values to each CNPU as required. The rest of the datapath functions identically to the VND datapath described in Section 4.3.3.1, including the $2D_C = 6$ programmable BSs that use the values in the SHIFT ROM to rotate the messages, and the de-multiplexer to provide the VMEM inputs.

The high-rate configuration shown in Figure 4.5(b) behaves slightly differently in several key ways. Firstly, due to the fact that up to $2D_C = 6$ input submatrices are non-null, the initial multiplexer no longer provides "null" data on any of its outputs (although it may still provide "off" values when $d_c < 2D_C$). The action of the distributor also changes to divide each group of $2D_C = 6$ messages across two outputs, in order to simplify the routing and control later in the datapath as will be shown. The first set of DFFs and multiplexer after the distributor perform the equivalent function to their counterparts in Figure 4.5(a), namely reducing the $\psi = 8$ groups into $\psi_c = 4$ groups. The following DFFs and multiplexer perform in a similar way to route the top $D_C = 3$ messages to the top half of the merged CNPU, while routing the bottom $D_C = 3$ messages of the same row to the bottom half of the merged CNPU. It can therefore be seen that the 1st and 3rd rows (red and orange) will be processed in the first clock cycle, followed by the 2nd and 4th rows (turquoise and blue) in the second clock cycle, and so on. These DFFs are avoided in the low-rate configuration using extra multiplexers, which for simplicity are not shown in Figure 4.5.

4.3.3.3 Programmable barrel shifters

Once a group of ψ 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 corresponding shift value s in $\mathbf{H}_b$, which is performed by the programmable BSs in the interleaver, as mentioned previously.

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 $\psi < Z$ of NPUs, additional DFFs must be used at the BS inputs to hold each group of ψ NPU outputs until all z outputs have arrived and the shifting can be completed. Similarly, additional DFFs are employed to hold the Z BS outputs after they have been shifted, so that groups of ψ 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 [47]. Accordingly, several competing designs have been proposed in the literature [18, 68, 78, 136–138], each with their own strengths and weaknesses. The design proposed in [18] 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 [138], 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 6.4.3.

4.3.4 NPU architectures

The design of the VNPUs and CNPUs can have a significant effect on the overall hardware resource usage of an LDPC decoder, as well as its critical path, and hence maximum clock frequency (f_{max}), and hence processing throughput and latency. This problem may be exacerbated depending on how 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 simulated and compared, as presented in the following discussions.

4.3.4.1 VNPU architecture

The VNPU architecture 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 Figure 4.6. 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 mentioned previously in Section 4.3.3.1, when a VNPU is used to calculate a PCM column with $d_v < D_V$, unused inputs can be "turned off" by supplying an input value of 0 since this does not affect the above-mentioned summations.

The hardware requirements and maximum operating frequency of five different VNPU designs are presented in Table 4.2 for the case of $D_V = 12$, which is the maximum column degree of the PCMs within IEEE 802.11n [3]. From these results, the **Tree SM** architecture was selected for inclusion in the proposed design due to its small hardware resource consumption and high f_{max}, which results from its short critical path. A description of its key features is presented here, together with brief descriptions of the rejected designs.

TABLE 4.2: Comparison of VNPU architectures

Design	ALUTs	FFs	BRAMs	f_{max}
Tree SM	267	49	0	210.8
Sum-Minus	234	49	0	206.4
Fwd-Bwd	468	49	0	98.66
Lookups	648	13	13	207.38
Loops	467	49	0	100.24

Note that the operation performed between relevant operands within a variable node, namely addition, has an easily-definable inverse operation, namely subtraction. Accordingly, as shown in Figure 4.6, the **Tree SM** 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 6.4.1. 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.

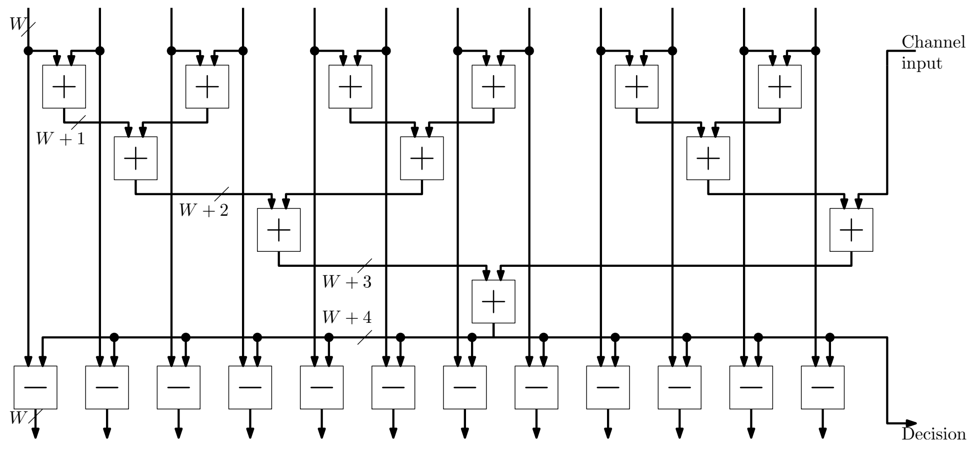


FIGURE 4.6: Tree SM VNPU structure with $D_V = 12$

Similarly, the **Sum-minus** architecture also calculates the total sum then uses the inverse operation to calculate each output. However, in this architecture the HDL to calculate the total sum is written without specifying how this should be synthesised. It can be seen from Table 4.2 that this results in a VNPU with a slightly smaller hardware resource requirement, but also a slightly longer critical path, than the Tree SM architecture.

By contrast, the **Fwd-Bwd**, **Loops**, and **Lookups** architectures calculate each output individually, eliminating the requirement for the inverse operation. The Fwd-Bwd architecture utilises the Forwards-Backwards algorithm [54], which is known to maximally reuse subnode outputs, hence requiring a minimal number of $3 \times (D_V - 1)$ additions to produce $D_V + 1$ separate outputs [139]. However, due to its long chain of $D_V - 1$ additions, this architecture results in a very long critical path, and still has a hardware resource requirement higher than that of the architectures employing the inverse operation. Table 4.2 shows that the Loops architecture fares similarly poorly in terms of high hardware resource usage and low operating frequency. Similarly to the Sum-minus architecture, the HDL for this VNPU design is written using high-level constructs such as nested loops without explicitly stating how it should be synthesised. By contrast, the Lookups architecture represents all two-input functions as explicitly-stated lookup tables, aiming to mimic an FPGA's core structure. While this structure benefits from a short critical path length, it requires far more logic resources than other designs, including $D_V + 1$ BRAMs, which renders it impractical for FPGA use. The Lookups architecture also suffers from very poor decoding performance, owing to its saturation of all intermediate results to W bits.

4.3.4.2 CNPU architecture

The CNPU architecture 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 [55] is employed in the proposed decoder. This algorithm calculates each of its 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 with $d_c < D_C$, the extra $D_C - d_c$ LLR inputs can be "turned off" by supplying a 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.

Table 4.3 compares the hardware resource requirements and maximum operating frequencies of five candidate CNPU architectures with $D_C = 24$, which is the maximum row degree of the PCMs within IEEE 802.11n [3]. Based on these results, the **Min Tree** architecture was selected for inclusion in the proposed design due to its combination of short critical path and small hardware resource consumption. A description of its key features is presented here, together with brief descriptions of the rejected designs.

TABLE 4.3: Comparison of CNPU architectures

Design	**ALUTs**	**FFs**	**BRAMs**	f_{max}
Min Tree	2005	96	0	137.8
Min-Sum Tree	2266	96	0	113.87
Fwd-Bwd	939	96	0	34.56
Lookups	2920	24	24	140.73
Sort Tree	923	96	0	85.84

The function of a CNPU is more complex than that of a VNPU since the min operation cannot be inverted. Owing to this, most of the node designs presented here calculate each output separately, albeit with some re-use of intermediate calculation results as in [57]. The chosen **Min Tree** architecture of Figure 4.7 exemplifies this approach, in which D_C tree structures are created in order to find the minimum of the magnitudes of each combination of $D_C - 1$ inputs, while making the maximum possible re-use of calculated minimums between tree structures. 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 by the total sign and the sign of its original input. Note that in Figure 4.7 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 output is saturated in the range (-2^{W-1}) to $(2^{W-1} - 1)$ as normal.

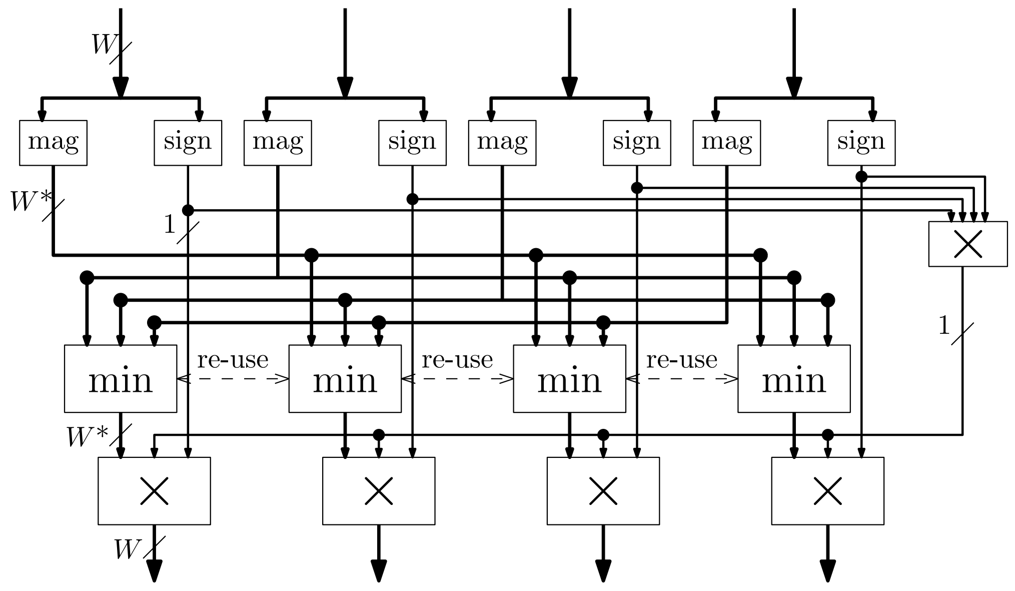


FIGURE 4.7: Min Tree CNPU structure with $D_C = 4$

Each of the Min-Sum Tree, Fwd-Bwd, and Lookups architectures also calculate each output separately, albeit in different ways. The **Min-Sum Tree** approach uses a similar set of overlapping trees as is used in the Min Tree architecture. However, where the Min Tree approach calculates the minimum for each output separately from the cumulative sign, the Min-Sum Tree architecture performs the minimum and cumulative sign jointly at each intermediary stage. It can be seen from Table 4.3 that this results in a design that is both slower and fractionally larger than the Min Tree. This approach also suffers from a slight performance loss due its need to use signed logic at each intermediary stage, which causes saturation of any output result equal to 2^{W-1}. The **Fwd-Bwd** architecture and **Lookups** architecture, meanwhile, perform in equivalence to their VNPU counterparts described previously. Table 4.3 demonstrates that these approaches have the same disadvantages for CNPUs as they did for VNPUs, namely very slow operation for the Fwd-Bwd and very large hardware resource requirements for the Lookups.

In contrast to the other four possible architectures, the **Sort Tree** CNPU design considers all D_C inputs simultaneously, by sorting the magnitudes of all inputs in order to find the minimum and second-minimum. Simultaneously, the total cumulative sign is calculated as in the Min Tree. Each output can thereby be calculated as the minimum of all inputs with appropriate sign, except where that minimum originates from the corresponding input, in which case the second-minimum is used. Table 4.3 shows that this design has a very low hardware resource requirement, but its critical path is significantly longer than the alternatives.

Note that while the Min Tree architecture may be judged to be preferable to the alternatives listed here, a superior architecture will be presented in Section 5.3.3.2. This so-called *dual-tree* architecture is applicable to any number representation and node function, and supports run-time flexibility without the need for a "null" value to be supplied on unused inputs. The impact of implementing dual-tree CNPUs into the proposed fixed-point decoder architecture may be considered for future work. Meanwhile, however, the Min Tree architecture discussed here retains usefulness in illustrating the potential of a fully run-time flexible decoder architecture.

4.3.4.3 Flexible CNPUs

Section 4.3.3.2 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 Min Tree 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 Figure 4.8, which is similar to Figure 4.7 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, so are only synthesised in the proposed architecture if the decoder parameter $F = 1$ to indicate that they are needed by at least one of the supported PCMs, as explained in Section 4.3.3.2.

4.3.5 Controller

In addition to the data processing units and memories, an LDPC decoder must also include hardware to control 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 must also determine whether the decoding process has been a success. These aspects are described in turn in the following discussions.

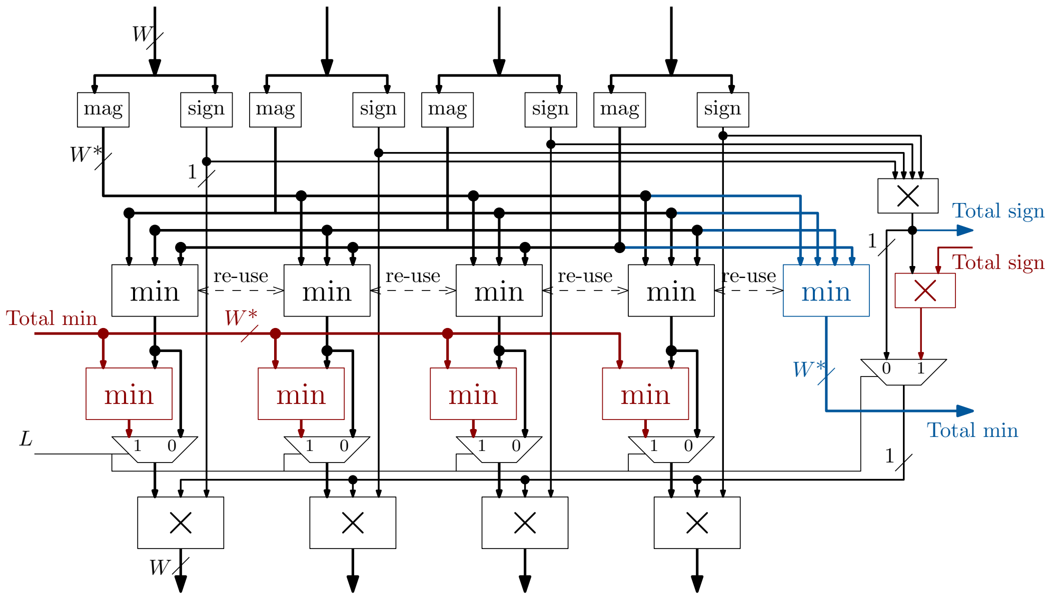


FIGURE 4.8: Flexible Min Tree CNPU structure with $D_C = 4$. Inputs from paired CNPU are shown in red, outputs to paired CNPU are shown in blue

4.3.5.1 Control signals

The input control signals are depicted in the bottom-left of Figure 4.2. The **load** signal may be used to indicate that a new set of input intrinsic LLRs should be loaded into the Intrinsic Message Memory Bank (IMMB). Storing the intrinsic message LLRs within memory increases the maximum operating frequency of the decoder, since high-speed memory operations may be used during the decoding process, rather than relying on high-latency external data operations. However, this approach requires the intrinsic LLRs to be loaded into the IMMB before they can be used. Owing to this, the IMMB of the proposed architecture contains two separate locations, allowing the next frame of LLRs to be loaded during the current decoding process. This facilitates the high throughput processing of several successive messages, since each new decoding process may be started in a single clock cycle.

The **reset** signal may be used to indicate that a new frame is to be decoded, resetting all internal components to their initial values. A rising edge on this signal additionally switches the active IMMB location, causing the recently-loaded new message to instantly become available for decoding. Upon a system reset, the values in the VMEM and CMEM become invalid for the new coded message. However, erasing all of the locations would require at least $M_B \times Q$ clock cycles. To avoid this, each location in each BRAM is accompanied by a single register to indicate that the data inside is invalid. This register is set up on a system reset and is unset when the memory is written to. Each time the memory location is read, this register is checked; if it is set, null data is read instead of the memory's contents. This facilitates the decoder's ability to switch frames immediately, at the cost of requiring $N_B \times M_B \times Q$ registers and multiplexers (one for each BRAM location).

The **PCM** input is a multi-bit signal, which allows the selection of the currently active PCM. One of the key features of the proposed decoder design is its ability to switch the 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 ROMs at compile-time, as discussed in Section 4.3.1. The current value of the PCM input is used throughout the decoder to address these memories.

Finally, the **start** signal commences and maintains the decoding process. The decoder will continue to perform iterative decoding until this signal stops being asserted, whereupon the decoding process is frozen. When the signal is re-asserted again, the iterative decoding process resumes precisely from where it left off, unless the decoder has been reset or the PCM has been changed in the meantime.

4.3.5.2 Early stopping detection

As discussed in Section 2.5.4, the controller is also required to determine when to terminate the iterative decoding process. In order to increase the decoding throughput, the proposed architecture includes an early stopping detection unit. This module views the decoded outputs of the ψ VNPUs in each clock cycle and maintains the cumulative output of every parity check within the current PCM. More specifically, the early stopping detection unit 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 a 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. This allows the operator of the decoder to determine at run-time when to give up on a decoding process, using the start and reset control signals described previously.

4.4 Implementation results

In this section, we present the measured characteristics of several instances of the proposed decoder, for a variety of configurations. Motivated by the recommendations made in Section 3.4.2, comparisons with relevant benchmarkers are made wherever possible. Furthermore, in order to highlight the practicality of the proposed decoder, the following comparisons focus on QC PCMs published in four major state-of-the-art wireless communications standards, namely IEEE 802.11n/ac (WiFi) [3], IEEE 802.16e (WiMAX) [4], IEEE 802.15.3c (Wireless Personal Area Network, henceforth referred to as WPAN) [140], and IEEE 802.11ad (WiGig) [135].

The structure of this section is as follows. Firstly, Section 4.4.1 discusses the methods employed to measure the characteristics, in line with the analysis presented in Chapter 3. Following this, parametrisations of the proposed decoder that are targeted at PCMs from the WiFi, WiMAX, WPAN, and WiGig families are characterised and discussed in Sections 4.4.2, 4.4.3, 4.4.4 and 4.4.5, respectively. Finally, Section 4.4.6 presents the characteristics of decoders targeted at PCMs from more than one of these families simultaneously. Throughout this section, each specific decoder parametrisation is identified using an abbreviated name, comprised of two letters and an identifying number. The first letter is always 'F' in this section regarding the **F**ixed-point architecture, which will be compared to the **S**tochastic architecture in Section 5.4. The second letter identifies the code family targeted by that decoder instance, namely 'F' for Wi**F**i, 'M' for Wi**M**ax, 'P' for W**P**AN, 'G' for Wi**G**ig, or 'S' to represent support for **S**everal of these families simultaneously. The specific parameters of each decoder instance will be discussed in the relevant sections.

It should be noted that, due to the huge degrees of flexibility involved, it would be impractical to characterise every combination of PCMs that the proposed architecture can be configured to support. Owing to this, the results here are selected as representative samples of possible combinations, to highlight key features and issues related to the proposed architecture's performance in practical applications.

4.4.1 Method

For each chosen set of QC PCMs, combined with a value for the parallelism reduction factor Q, the offline design process presented in Chapter 6 was utilised to produce a SystemVerilog description of a decoder having the architecture presented in Section 4.3. This description was then synthesised using Intel Quartus II with the target FPGA selected as an Intel Stratix IV EP4SGX530. The results of this synthesis were a measurement of the hardware resource usage and the maximum operating frequency f_{max}. The Equivalent Logic Blocks (ELBs) metric proposed in Section 2.6.3 was used to characterise the hardware resource requirements of each synthesised decoder, so that they can be compared to the published decoders listed in Table 3.1.

At the same time, bit-accurate C++ Bit Error Rate (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 surveyed in Chapter 3. The 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 is 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 Section 3.2.2. Finally, the average number of decoding iterations I_a required to achieve this BER performance at this E_b/N_0 were measured and hence used to calculate the decoded processing throughput T, according to

$$T = \frac{f_{max} \times n \times R}{t_i \times I_a}, \tag{4.8}$$

where n is the 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, as discussed in Section 3.2.2.

4.4.2 Decoders targeted at WiFi LDPC PCMs

The standard on which WiFi is based, IEEE 802.11n/ac [3], 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 the same number of block-columns $n_b = 24$, resulting in three different values of z corresponding to the three frame lengths, according to $z = {}^n\!/_{n_b}$. Furthermore, the number of block-rows m_b employed by each PCM may be calculated according to $m_b = n_b \times (1 - R)$ for the four different coding rates.

The FPGA-based LDPC decoder presented in [27] employs a parameterised architecture and an offline design flow similar to those proposed in Section 4.3 and Chapter 6. The authors of [27] state that the decoder parameters are "limited run-time reconfigurable by over allocation and selective enabling", however this is not then demonstrated or discussed further. The only characteristics presented in [27] 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 $R = R_1 = {}^1\!/_2$. The work of [33] is the only comparable FPGA-based LDPC decoder found during the survey presented in Chapter 3 that supports run-time flexibility across all 12 of the WiFi PCMs mentioned above. However, measurements of the processing throughput, hardware resource requirements and transmission energy efficiency are only available 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 [27] and [33], four separate instances of the decoder proposed in this work were implemented and characterised for WiFi PCMs. The first, referred to as decoder FF1, 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 [27] (highlighted in blue in Table 4.4). The second decoder, FF2, targets the three PCMs with $R = R_1 = 1/2$, having frame lengths $n_1 = 648$ to $n_3 = 1944$. Thirdly, decoder FF3 targets the four PCMs with frame length $n_1 = 648$, having rates $R_1 = 1/2$ to $R_4 = 5/6$. Finally, decoder FF4 targets all 12 PCMs in the WiFi family, facilitating a direct comparison to the results from [33] (highlighted in orange in Table 4.4). Note that decoders FF2 and FF3 do not fairly compare with the benchmarkers [27] and [33] in terms of the number of supported PCMs. However, they have been implemented and characterised in order to demonstrate the proposed architecture's ability to support a reduced subset of PCMs from within a family.

Table 4.4 characterises FF1, FF2, FF3, and FF4, along with the designs of [27] and [33] mentioned previously. 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 FF1, FF2, and FF4. Accordingly, the values of Q displayed in Table 4.4 for each of these decoders were selected such that they all employ $\psi = 27$ VNPUs. Furthermore, the results of Table 4.4 are plotted in Figure 4.9, together with the rest of the results of the survey conducted in Chapter 3.

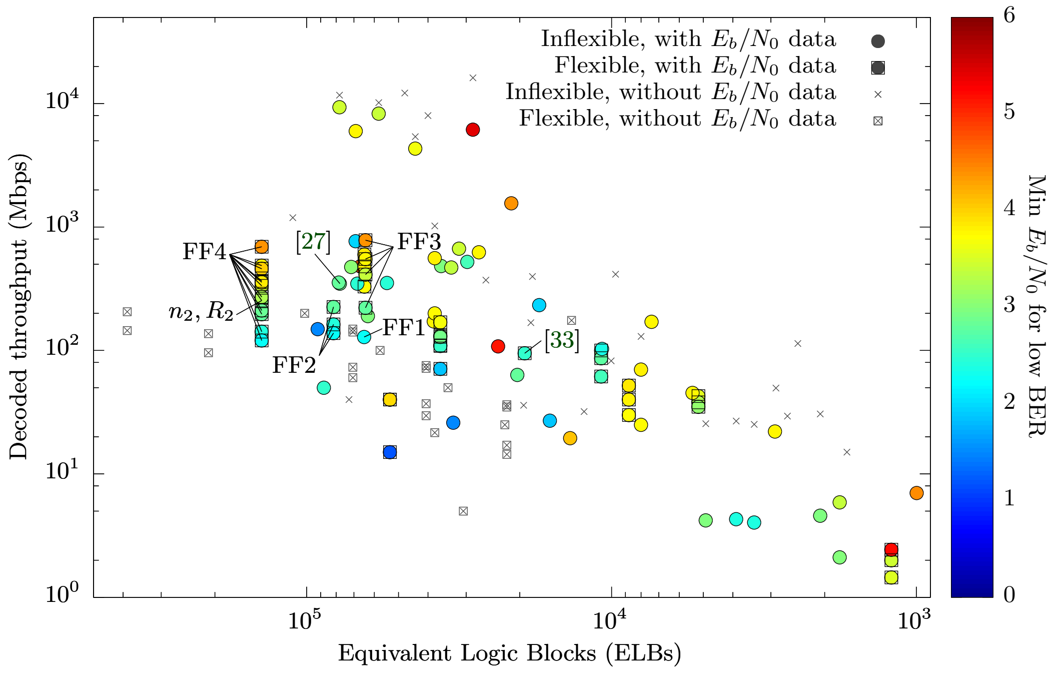


FIGURE 4.9: WiFi characterisation of the proposed fixed-point architecture

As described above, decoder FF1 may be directly compared to the benchmarker of [27], since neither of them possess run-time flexibility and since they both target the same PCM having frame length $n_3 = 1944$ and coding rate $R_1 = 1/2$. As shown in Table 4.4, decoder FF1 suffers from a 63% lower processing throughput than the benchmarker of [27], but requires 17% fewer hardware resources and achieves the target BER with a 0.6 dB lower transmission energy requirement. Additionally, the throughput presented

Table 4.4: WiFi characterisation of the proposed fixed-point architecture

Decoder	Q	Active PCM n	Active PCM R	t_i	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
FF1	3	1944	1/2	72	1	64.8	9.91	128.6	7.56	2.23
FF2	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
FF3	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
FF4	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
[27]	-	1944	1/2	-	1	77.8	10	348	11.2	2.83
[33]	-	1296	2/3	-	12	19.3	15	95	9.09	2.49

in [27] is achieved by decoding four frames simultaneously using four parallel decoder copies, which does not improve the associated processing latency. Owing to this, it may be inferred that the processing latency of decoder FF1 is 33% lower than that of [27], as shown in Table 4.4. Furthermore, [27] does not provide any details regarding the complexity or the impact of introducing run-time flexibility upon any of its measured characteristics, whereas the architecture of decoder FF1 includes an overhead that is associated with being completely generalisable to any number of QC PCMs.

Similarly, decoder FF4 may be directly compared to the design of [33], which offers run-time flexibility for any PCM in the WiFi family. However, in order to maintain a fair comparison, the results from the decoding process for the same PCM having coding rate $R_2 = 2/3$ and frame length $n_2 = 1296$ should be considered. The characteristics of decoder FF4 when using this frame length and coding rate are indicated accordingly in Figure 4.9. Table 4.4 shows that for this PCM, decoder FF4 achieves a processing throughput that is 167% higher than that of [33]. Additionally, the error correction performance of the two decoders is approximately the same, with a BER of 10^{-4} being achieved at an E_b/N_0 level of 2.99 dB and 2.83 dB for decoder FF4 and [33], respectively. However, decoder FF4 suffers from a 7.3 times larger hardware resource usage than that of [33]. This cost may be attributed to the highly flexible routing networks employed by the proposed architecture, which may be generalised to multiple code families with a wider variety of submatrix sizes and node degrees than those of the WiFi family, as discussed in Section 4.3.3.3. By contrast, the design of [33] 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 FF2 and FF3, 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 ψ. 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 4.4.6. 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. Furthermore, the comparatively low hardware resource requirement of decoder FF3 also demonstrates the effectiveness of the methods discussed in Sections 4.3.3.2 and 4.3.4.3, 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 Figure 4.10, wherein the main characteristics of decoders FF1, FF2, and FF4 are plotted radially with respect to the characteristics of [27], 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. Figure 4.10 shows that each of the decoders presented in this section attain a lower processing throughput than that of [27], but with a smaller latency, greater error correction performance, and equal or greater run-time flexibility. Figure 4.10 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.

4.4.3 Decoders targeted at WiMAX LDPC PCMs

A total of 114 QC PCMs are defined by the IEEE 802.16e standard on which WiMAX is based. This range is comprised of 19 different frame lengths, $n_1 = 576$ to $n_{19} = 2304$, increasing in multiples of 96. For each frame length there exists the same four coding rates used in IEEE 802.11n/ac, although two of these (namely $R = 2/3$ and $R = 3/4$) are associated with two slightly different base PCMs $\mathbf{H}_b$, denoted as a and b. This leads to

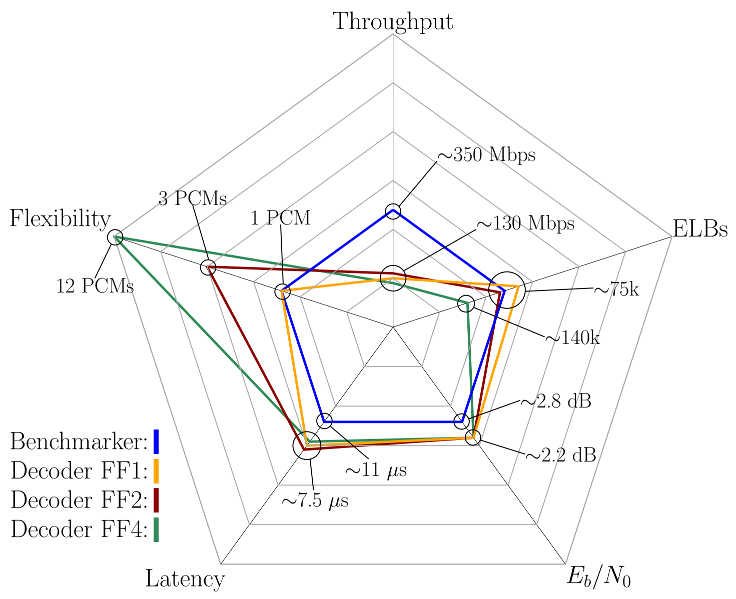


Figure 4.10: Comparison of decoders targeted at PCMs from the WiFi family. Characteristics are normalised relative to the benchmarker from [27].

a total of six rates, according to $R_1 = 1/2$, $R_2 = 2/3$a, $R_3 = 2/3$b, $R_4 = 3/4$a, $R_5 = 3/4$b, and $R_6 = 5/6$. Similarly to IEEE 802.11n/ac, the number of block-columns $n_b = 24$ is the same for every PCM, with the result that the frame length is solely dependent on the 19 different values of z, namely $z_1 = 24$ to $z_{19} = 96$, increasing in multiples of 4. Furthermore, the coding rate is solely dependent on the number of block-rows m_b, according to $m_b = n_b \times (1 - R)$ for each of the four values of R.

Similarly to the WiFi PCMs, a value for the parallelism factor Q has to be chosen when designing a decoder for the WiMAX codes with the higher frame lengths, due to the corresponding values of z which would result in an unmanageable degree of parallelism. However, in contrast to the WiFi PCMs, this high resolution of values of z poses a challenge for the selection of a desirable value of Q in order to implement a decoder which is compatible with all 114 PCMs. To illustrate this, suppose that a value of $Q = 4$ is chosen. The largest value of z in the WiMAX family is $Z = 96$, resulting in $\psi = 24$. When decoding PCMs with the smallest value of $z = 24$, clearly this decoder would be able to calculate every block-column in $t_{vb} = 1$ clock cycle, resulting in $t_i = 24$. However, when decoding the next largest frame length with $z = 28$, $t_{vb} = 2$ clock cycles would be required per block-column, with $\psi = 24$ columns being processed in the first clock cycle and $z - \psi = 4$ being processed in the second clock cycle. As a result of this, a small increase in the frame length n would double the total number of clock cycles per iteration t_i. Since the processing throughput is proportional to n/t_i according to (4.8), this would result in a significant reduction in processing throughput for several of the PCMs in the set.

In order to avoid this problem, a value of Q may be selected which results in the parallelism ψ being a factor of all submatrix sizes. For the complete set of WiMAX PCMs, this value is $Q = 24$, which leads to $\psi = 4$. A version of this decoder for these parameters was implemented, which will be referred to as decoder FM2a, which flexibly supports all 114 PCMs in the WiMAX family. Since the low degree of parallelism in FM2a results in low processing throughputs, a second version, decoder FM2b, was implemented with $Q = 12$, leading to $\psi = 8$, with the same level of run-time flexibility. Decoder FM2b suffers from the problem described above, though its effect is small. More explicitly, for each of the PCMs having one of the nine QC submatrix sizes z that are not an exact multiple of $\psi = 8$, exactly $(z \bmod 8) = 4$ VNPUs are idle for the final clock cycle of each block-column.

Similarly to the discussion in Section 4.4.2, additional comparisons are made with decoder designs published in the current literature. The only two comparable FPGA-based LDPC decoders found during the survey presented in Chapter 3 which target WiMAX PCMs are the works of [62] and [20]. The work presented in [62] proposes a fully-parallel non-flexible stochastic FPGA-based LDPC decoder, which targets the single WiMAX PCM with $R = R_1 = 1/2$ and $n = n_6 = 1056$, resulting in $z = z_6 = 44$. Correspondingly, another decoder of the proposed architecture, decoder FM1, was implemented to target the same PCM, with $Q = 2$ which leads to $\psi = 22$. Additionally, [20] proposes a decoder which claims the ability to switch between any of the WiMAX PCMs at run-time, which would make it comparable to the flexible decoders FM2a and FM2b described previously. Unfortunately, the characteristics presented in [20] do not include any BER results and do not specify which frame lengths they are valid for, which meant it was not suitable for inclusion in the survey conducted in Chapter 3. However, in order to use it as a benchmarker to compare with the proposed architecture, it is assumed here that the results presented in [20] are valid for all values of n within the WiMAX family. The characteristics of these decoders are presented in Table 4.5. Note that for brevity, Table 4.5 only presents the characteristics for a representative subset of the PCMs targeted by the flexible decoders FM2a, FM2b, and [20].

As before, these results are also depicted in the plot of Figure 4.11. Here, it may be observed that the inflexible decoder FM1 has a 49.9% lower hardware resource requirment and a 62.2% lower processing throughput than the inflexible fully-parallel stochastic decoder presented in [62], which has been designed to decode the same single PCM. However, the process of designing the decoder in [62] is shown to be a complex task which must be re-produced for any code, having no design-time flexibility. Additionally, the fully-parallel nature of this decoder renders it unlikely to be able to support any significant degree of run-time flexibility.

Figure 4.11 and Table 4.5 also show that the the two decoders having the proposed architecture that were implemented to support all 114 PCMs in the WiMAX family suffer from a significantly larger hardware resource requirement than any other non-commercial decoder surveyed in Chapter 3. Specifically, the hardware resource requirements on FM2a and FM2b are respectively 3.43 and 3.68 times higher than that of the

TABLE 4.5: WiMAX characterisation of the proposed fixed-point architecture

Decoder	Q	Active PCM n	Active PCM R	t_i	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
FM1	2	1056	1/2	48	1	34.2	9.97	131.4	4.02	2.85
FM2a	24	576	1/2	144	114	130.4	7.98	26.4	10.91	3.14
		576	2/3b	144		130.4	5.58	50.3	7.63	3.71
		576	3/4a	144		130.4	4.87	64.9	6.66	4
		576	5/6	144		130.4	3.99	88.0	5.45	4.45
		1248	1/2	312		130.4	10.55	20.0	31.24	2.77
		1248	2/3b	312		130.4	7.64	36.8	22.63	3.29
		1248	3/4a	312		130.4	6.59	48.0	19.52	3.64
		1248	5/6	312		130.4	5.47	64.2	16.20	4.09
		2304	1/2	576		130.4	13.08	16.1	71.51	2.52
		2304	2/3b	576		130.4	9.39	29.9	51.34	3.07
		2304	3/4a	576		130.4	7.85	40.3	42.92	3.45
		2304	5/6	576		130.4	6.97	50.4	38.11	3.88
FM2b	12	576	1/2	72	114	139.8	7.98	48.6	5.93	3.14
		576	2/3b	72		139.8	5.58	92.6	4.15	3.71
		576	3/4a	72		139.8	4.87	119.4	3.62	4
		576	5/6	72		139.8	3.99	161.9	2.97	4.45
		1248	1/2	168		139.8	10.55	34.1	18.29	2.77
		1248	2/3b	168		139.8	7.64	62.8	13.25	3.29
		1248	3/4a	168		139.8	6.59	81.9	11.43	3.64
		1248	5/6	168		139.8	5.47	109.7	9.48	4.09
		2304	1/2	288		139.8	13.08	29.6	38.88	2.52
		2304	2/3b	288		139.8	9.39	55.0	27.91	3.07
		2304	3/4a	288		139.8	7.85	74.1	23.33	3.45
		2304	5/6	288		139.8	6.97	92.7	20.72	3.88
[62]	-	1056	1/2	1	1	68.2	NA	348	1.5	2.45
[20]	-	NA	1/2	NA	114	38	20	10.4	NA	NA

benchmarker from [20]. 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 19 possible values of z. The impact of this may be observed by comparing decoders FM2a and FM2b, where decoder FM2b has half the number of NPUs of FM2a, but its hardware resource requirement is only 6.7% smaller. This may be explained by the domination of the overall hardware resource requirement by the BSs, which are of equal number and size in both FM2a and FM2b.

Furthermore, the 100% increase in the degree of parallelism present in decoder FM2b compared to decoder FM2a corresponds to an approximately 80% increased processing throughput, at the cost of a very small increase in hardware resource usage. However, this relationship is non-linear and is not sustainable for higher degrees of parallelism,

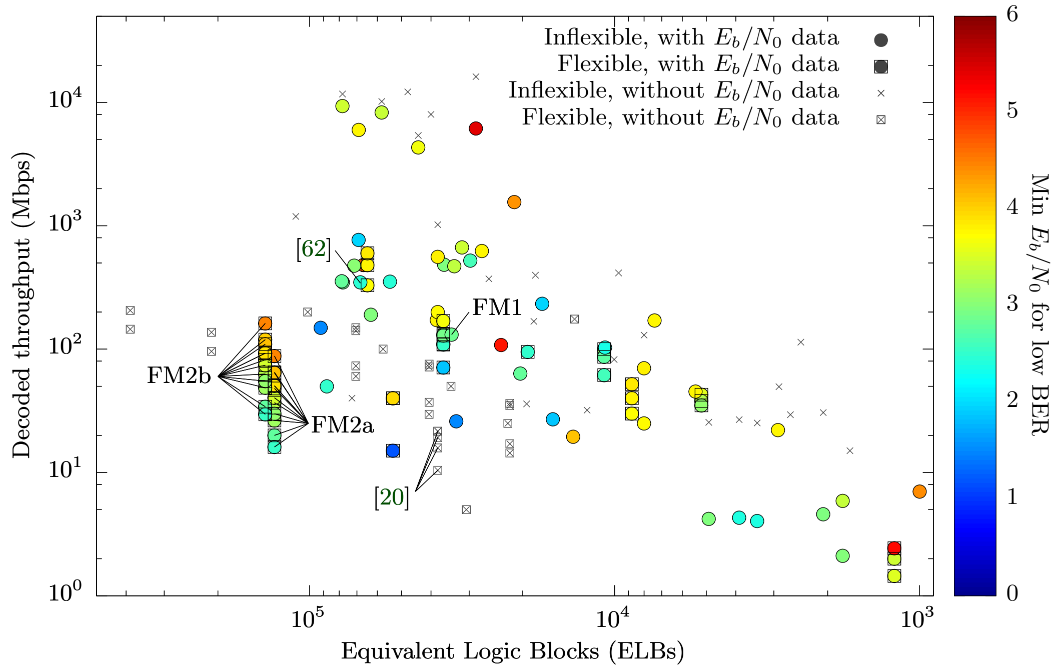


FIGURE 4.11: WiMAX characterisation of the proposed fixed-point architecture

owing to the routing congestion on the FPGA. More specifically, FM2b has a maximum operating frequency which is 8% lower than that of FM2a, due to the increased utilisation of the programmable routing structures of the FPGA. This in turn causes an increased critical path length, as signals have to be transported on less direct routes across the FPGA chip. Routing congestion for large numbers of multi-bit signals is one of the main issues motivating research into bit-serial stochastic LDPC decoders, which only require a single wire per message. However, stochastic decoders also present a number of different design problems, which are addressed further in Chapter 5.

4.4.4 Decoders targeted at WPAN LDPC PCMs

The family of QC LDPC PCMs denoted in this work as WPAN originates from the IEEE 802.15.3c standard for mm-wave transmissions in the 60 GHz frequency band. Four PCMs are defined, all with the same frame length $n = 576$, QC submatrix size $z = 21$ and number of block-columns $n_b = 32$. Once again, the number of block-rows varies to accommodate the four coding rates, $R_1 = 1/2$, $R_2 = 5/8$, $R_3 = 3/4$, and $R_4 = 7/8$. Unfortunately, no published flexible FPGA-based decoders of WPAN LDPC codes could be found in the published literature with which to compare the proposed architecture. However, the work of [27] mentioned in Section 4.4.2 also implemented a non-flexible decoder which targets the WPAN PCM with $R = R_1 = 1/2$. Accordingly, two versions of a WPAN decoder using the proposed architecture were developed: FP1, which supports the PCM with the lowest rate R_1 to compare with [27], and FP2, which supports all four PCMs in the family to demonstrate the flexibility of the proposed architecutre. The moderate QC submatrix size z of these PCMs permits the use of $Q = 1$, with the result that each decoding iteration requires $t_i = N_B = 32$ clock cycles. The characteristics of these decoders are presented in Table 4.6, and plotted in Figure 4.12.

TABLE 4.6: WPAN characterisation of the proposed fixed-point architecture

Decoder	Q	Active PCM n	Active PCM R	t_i	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
FP1	1	672	1/2	32	1	27.9	6.76	194.6	1.73	3.02
FP2	1	672	1/2	32	4	65.2	6.76	157.5	2.13	3.02
		672	5/8	32		65.2	4.84	275.1	1.53	3.12
		672	3/4	32		65.2	3.99	400.4	1.26	3.56
		672	7/8	32		65.2	3.11	599.3	0.98	4.12
[27]	-	672	1/2	-	1	63.0	10	190	5.31	2.97

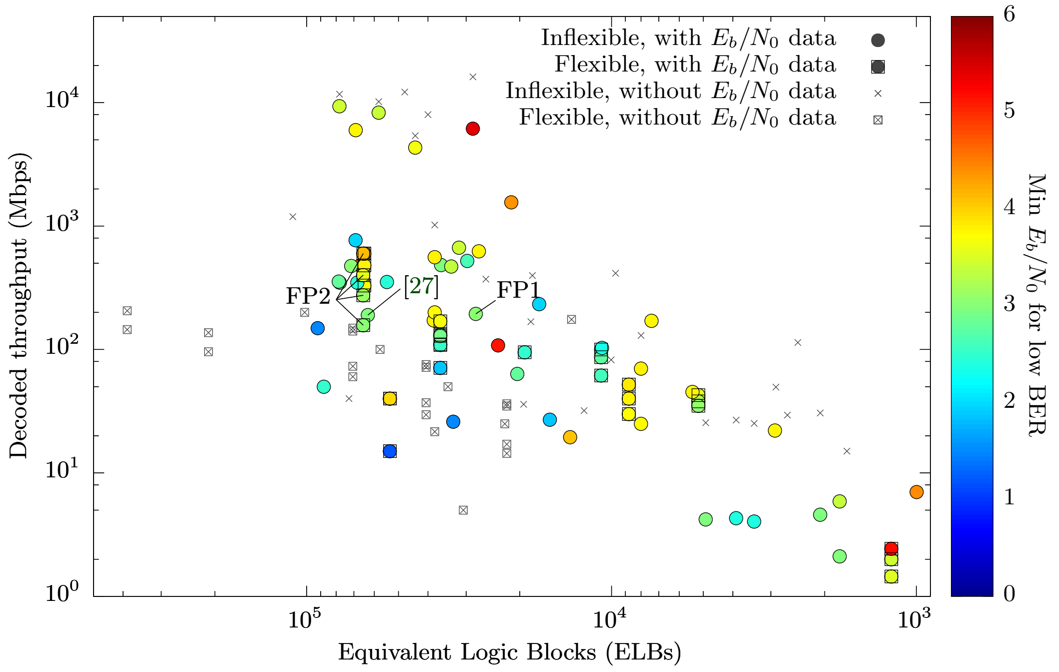


FIGURE 4.12: WPAN characterisation of the proposed fixed-point architecture

Of all the QC PCMs detailed in this work, the WPAN PCM with coding rate R_4 has the highest row degree, $D_C = 32$. Ordinarily this would call for the employment of a large number of very large CNPUs, dominating the hardware resource requirements of the decoder. However, the optimisations described in Sections 4.3.1.1 and 4.3.3.2 work together here to prevent this, producing the favourable results shown in Table 4.6 and Figure 4.12. Specifically, rather than employ $\psi = 21$ CNPUs in decoder FP2, $\psi_c = 11$ may instead be used due to the value of $G_{min} = 2$ in this PCM set. Furthermore, the linked CNPUs described in Section 4.3.4.3 may be employed for the PCMs having the two highest code rates R_3 and R_4, which halves the number of inputs and outputs required by each of the ψ_c CNPUs to ${}^{D_C}/_2 = 16$.

It may hence be observed in Figure 4.12 that the flexible decoder FP2 achieves a generally higher processing throughput than the inflexible benchmarker of [27], whilst requiring an approximately equivalent quantity of hardware resources. Meanwhile, the inflexible decoder FP1 achieves an equivalent processing throughput to that of the benchmarker, but with a 55.7% lower hardware resource requirement. Furthermore, due to the fact that the throughput in [27] is obtained by running three decoders in parallel, its processing latency is approximately three times larger than that of FP1.

4.4.5 Decoders targeted at WiGig LDPC PCMs

The IEEE 802.11ad standard defines the four QC PCMs used by WiGig, which are similar in several ways to the WPAN family described in the previous section. Specifically, the standard defines four PCMs with the same frame length $n = 672$ and a similar set of coding rates, namely $R_1 = {}^1/_2$, $R_2 = {}^5/_8$, $R_3 = {}^3/_4$, and $R_4 = {}^{13}/_{16}$. However, the number of block-columns $n_b = 16$ is half that of the WPAN PCMs, which limits the maximum row degree to $D_C = 16$, but requires a QC submatrix size $z = 42$, which is twice as large as the submatrix size used in WPAN. This moderate submatrix size facilitates an investigation into the effects of the parallelism reduction factor Q, without the decoder characteristics being dominated by the routing, as was seen in Section 4.4.3.

As before, only the work in [27] proposes comparable FPGA-based LDPC decoder results using a WiGig PCM, once more for the non-flexible case of only targeting a single PCM with $R = R_1 = {}^1/_2$. Accordingly, decoders FG1a and FG1b were also implemented to do so, where FG1a has a value of $Q = 2$ (hence $\psi = 21$), and FG1b has $Q = 1$ (hence $\psi = 42$). Furthermore, again no comparable flexible FPGA-based decoders for multiple WiGig PCMs could be found in the published literature. However, in order to further characterise the proposed architecture, decoders FG2a and FG2b were implemented to support the entire code family, having the same variations of Q and ψ as the corresponding non-flexible versions. The characteristics of these decoders are presented in Table 4.7, and plotted in Figure 4.13.

Both the flexible and non-flexible variants of the proposed decoders illustrate the expected result that increasing the level of parallelism increases both the hardware resource consumption and the processing throughput, though this relationship is not directly proportional. For example, FG1a has 50% of the parallelism of FG1b, but only a 33.9% reduced hardware resource requirement. Once again, this may be explained by the BSs which must remain a constant size in order to handle all of the shift values in the PCM, as well as the control and memory overhead which exists regardless of the size of the datapath. Additionally, FG1a has a 36.4% lower processing throughput than FG1b. In

TABLE 4.7: WiGig characterisation of the proposed fixed-point architecture

Decoder	Q	Active PCM n	Active PCM R	t_i	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
FG1a	2	672	1/2	32	1	21.1	6.7	184.3	1.82	3.14
FG1b	1	672	1/2	16	1	31.9	6.7	289.6	1.16	3.14
FG2a	2	672	1/2	32	4	34.4	6.7	164.9	2.04	3.14
		672	5/8	32		34.4	5.45	253.3	1.66	3.27
		672	3/4	32		34.4	4.17	397.3	1.27	3.72
		672	13/16	32		34.4	3.23	555.7	0.98	4.3
FG2b	1	672	1/2	16	4	51.8	6.7	338.9	0.99	3.14
		672	5/8	16		51.8	5.45	520.9	0.81	3.27
		672	3/4	16		51.8	4.17	816.9	0.62	3.72
		672	13/16	16		51.8	3.23	1142.5	0.48	4.3
[27]	-	672	1/2	-	1	71.4	10	475	4.24	3.02

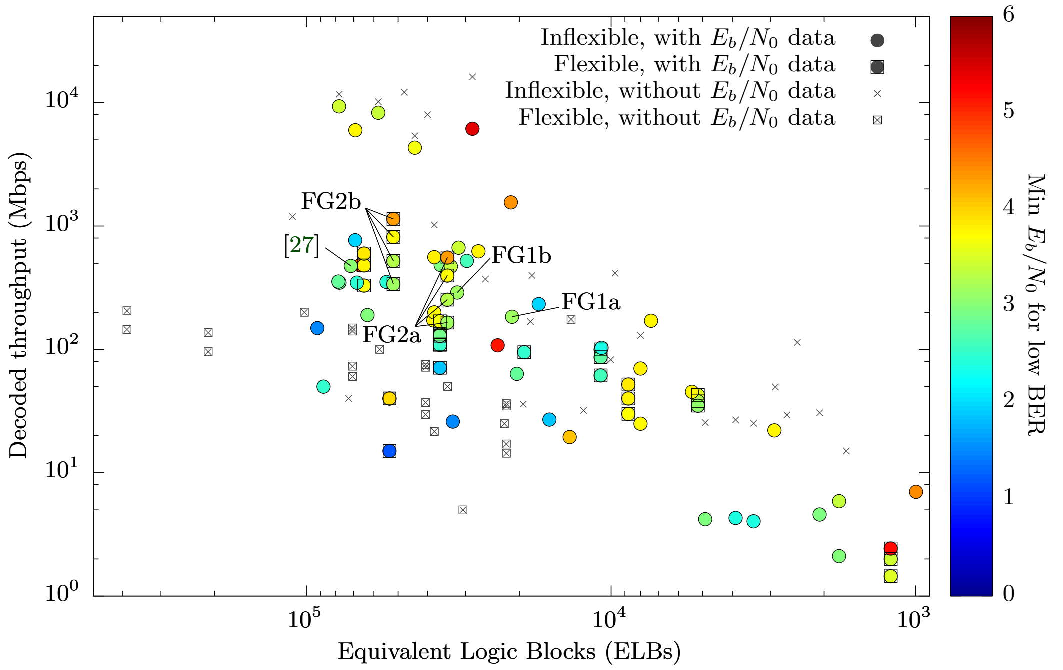


FIGURE 4.13: WiGig characterisation of the proposed fixed-point architecture

line with the discussions in Section 4.4.3, this may be attributed to the fact that FG1b achieves a maximum clock frequency that is 21.4% lower than that of FG1a.

Meanwhile, the flexible decoders FG2a and FG2b exhibit a similar relationship, with FG2b having a 50.6% higher hardware resource consumption, but with approximately double the processing throughput. This is due to the fact that the fitting and routing operations performed by the synthesis tool resulted in an approximately equal value of f_{max} for the two decoder variations, in contrast to the assumption that higher logic utilisation should result in longer critical paths. As a result, it may be seen in Figure 4.13 that decoder FG2b achieves processing throughputs higher than any other run-time flexible FPGA-based LDPC decoder surveyed in Chapter 3.

Another of the trade-offs described throughout Chapter 3 may be observed through comparison of Figure 4.13 with Figure 4.12 shown previously. By comparing the hardware resource consumptions of decoders FG2a and FP2, both of which employ $\psi = 21$, the benefits of the lower row degrees in the WiGig standard on the decoder's hardware resource consumption can be seen. However, by viewing the colours of the associated points, it may also be observed that PCMs with larger degrees tend to have better error-correction performance due to the increased number of edges. As such, decoder FG2a is observed to have reasonable processing throughput and hardware resource consumption compared to the surrounding points whilst maintaining run-time flexibility, though it does so at the cost of a poorer transmission energy efficiency.

4.4.6 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 4.3 is also capable of supporting PCMs from multiple different code families. This inter-standard flexibility is beneficial for the implementation of efficient Centralised Radio Access Network (C-RAN) cloud servers, which are capable of performing the LDPC decoding for multiple basestations of varying standards. Note that the automated design flow presented presented in Chapter 6 ensures that the process of designing and implementing such a decoder is no more complicated than the corresponding process for one of the decoders presented previously. More explicitly, the computations performed as part of the offline design process will automatically select desirable values for the decoder parameters in order to produce a design that can support all of the input PCMs, regardless of which standard they originate from.

In order to demonstrate this key feature, multiple instances of the proposed decoder were again implemented and characterised using the methods described in Section 4.4.1. Firstly, decoder FS1 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 $\psi = 42$ 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 FS2 targets the PCMs with the lowest frame length and coding rate from only the WiFi, WiMAX, and WPAN families. Here, the parallelism ψ is reduced to 27. Thirdly, decoder FS3 supports the PCMs from both the WiGig and WPAN families having the two highest coding rates, with a parallelism of $\psi = 11$. Furthermore, two additional decoders were implemented targeting 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 FS4a employs $Q = 24$, which results in a parallelism of $\psi = 4$, since $Z = 96$ is the maximum submatrix size among this selection of PCMs. Finally, decoder FS4b supports this same set of 134 PCMs, but adopts $Q = 8$, which leads to $\psi = 12$.

Throughout the survey of Chapter 3, only [26] and [25] 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 [25] 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 [25] 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 [26] were not generated using one of the standardised codes described previously either.

Table 4.8 and Figure 4.14 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 FS1 to FS4b.[4] These results show that the proposed decoders offer similar processing throughputs to the majority of FPGA-based LDPC decoders considered in Chapter 3, while offering a very high degree of run-time flexibility, albeit with a larger hardware resource requirement.

As anticipated, the reduced parallelism of decoder FS2 compared to FS1 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 FS1 to decode a WiGig PCM in addition to those supported by decoder FS2 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 FS3, which offers a high throughput at a relatively low hardware resource usage.

Finally, decoders FS4a and FS4b offer a much higher degree of run-time flexibility, at the cost of higher hardware resource requirements. This is again for the reasons stated in Section 4.4.3, whereby the hardware resource requirements are dominated by the large complex programmable BSs and other associated routing hardware. Flexible

[4]Note that in the third column of Table 4.8 the names of the code families featured here are abbreviated, where 'F' means 'Wi**F**i', 'M' means 'Wi**M**AX', 'P' means 'W**P**AN', and 'G' means 'Wi**G**ig'.

Table 4.8: Multiple-family characterisation of the proposed fixed-point architecture

Decoder	Q	Active PCM family	Active PCM n	Active PCM R	t_i	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
FS1	1	F	648	1/2	24	4	91.5	6.08	215.2	1.5	2.77
		M	576	1/2	24			7.98	145.7	2.0	3.14
		G	672	1/2	16			6.70	303.7	1.1	3.14
		P	672	1/2	32			6.76	150.5	2.2	3.02
FS2	1	F	648	1/2	24	3	58.8	6.08	218.6	1.5	2.77
		M	576	1/2	24			7.98	148.1	1.9	3.14
		P	672	1/2	32			6.76	152.9	2.2	3.02
FS3	4	G	672	3/4	64	4	44.9	4.17	129.1	2.6	3.72
		G	672	13/16	64			3.23	250.1	2.0	4.30
		P	672	3/4	64			3.99	135.0	2.5	3.56
		P	672	7/8	64			3.11	259.7	1.9	4.12
FS4a	24	F	648	5/6	168	134	208.0	2.88	105.8	5.1	4.36
		F	1296	3/4	336			5.37	51.1	19.0	3.42
		F	1944	1/2	504			9.91	18.5	52.7	2.23
		M	576	5/6	144			3.99	79.2	6.1	4.45
		M	864	3/4a	216			5.46	52.1	12.4	3.75
		M	1248	2/3b	312			7.64	33.1	25.1	3.29
		M	1920	1/2	480			12.26	15.5	62.1	2.61
		M	2304	2/3b	576			10.24	24.7	62.2	2.94
		G	672	1/2	176			6.70	27.0	12.4	3.14
		G	672	3/4	176			4.17	65.1	7.7	3.72
		P	672	5/8	192			4.84	42.9	9.8	3.12
		P	672	7/8	192			3.11	93.4	6.3	4.12
FS4b	8	F	648	5/6	72	134	228.9	2.88	227.1	2.4	4.36
		F	1296	3/4	120			5.37	131.5	7.4	3.42
		F	1944	1/2	168			9.91	50.9	19.1	2.23
		M	576	5/6	48			3.99	218.5	2.2	4.45
		M	864	3/4a	72			5.46	143.7	4.5	3.75
		M	1248	2/3b	120			7.64	79.1	10.5	3.29
		M	1920	1/2	168			12.26	40.6	23.6	2.61
		M	2304	2/3b	192			10.24	68.1	22.5	2.94
		G	672	1/2	64			6.70	68.3	4.9	3.14
		G	672	3/4	64			4.17	164.7	3.1	3.72
		P	672	5/8	64			4.84	118.2	3.6	3.12
		P	672	7/8	64			3.11	257.6	2.3	4.12
[25]	-	NA	NA	NA	NA	-	30.6	15	5.0	NA	NA
[26]	-	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

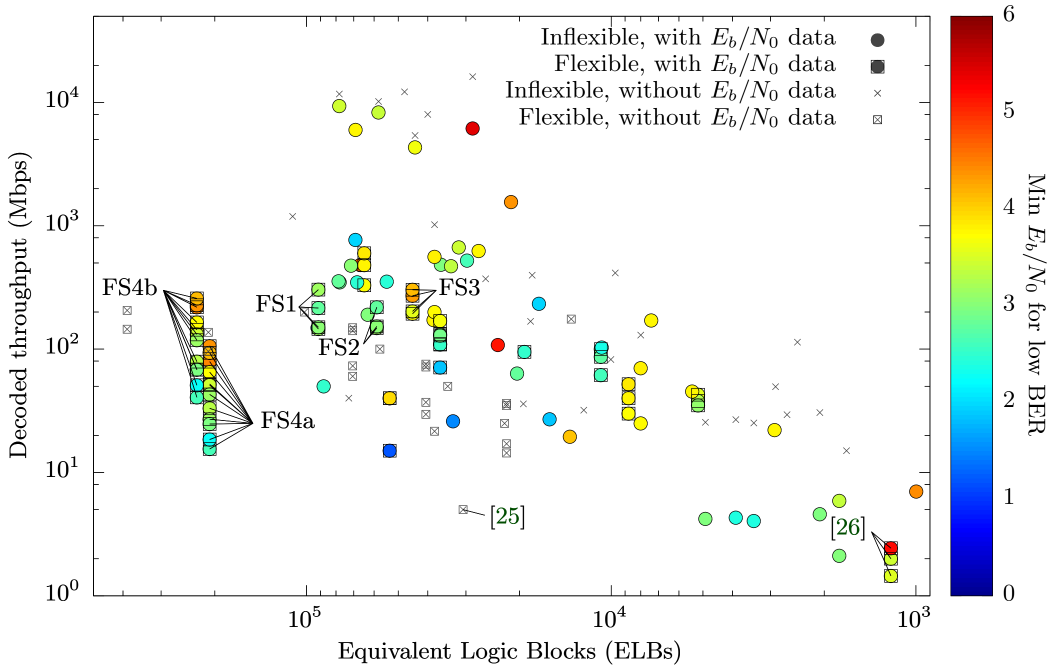


FIGURE 4.14: Multiple-family characterisation of the proposed fixed-point architecture

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) [141], which is required to have a much greater degree 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 [136] 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 frequency and hence throughput.

The results of Table 4.8 and Figure 4.14 show that all of the proposed decoders offer much higher processing throughputs than those of [26] and [25], 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 lower processing throughputs. This may be explained by the fully-serial architecture employed by the design of [26], which results in the very low hardware resource requirements and processing throughput shown here. Conversely, [25] 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 [25], 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 [25] 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.

4.5 Conclusion

In this chapter, we have presented the design and implementation of a fixed-point FPGA-based LDPC decoder, which may be used to decode LDPC codewords associated with any QC PCM, including those used in several state-of-the-art communications standards. By taking advantage of the characteristics of QC LDPC codes discussed in Section 4.2, the architecture presented in Section 4.3 is able to support any set of one or more QC PCMs, with the option to flexibly switch between them at run-time within a single clock cycle. This high level of run-time flexibility has motivated many of the design decisions discussed throughout this chapter, such as the allocation of the parameters N_B, M_B, Z, D_V, D_C, and the degree of parallelism ψ. Furthermore, the method of characterising each supported PCM in the ROMs (discussed in Section 4.3.1.3) in a hardware-optimised form ensures that any one of the supported PCMs may be used to control the decoding process at any time. These ROM values are mainly used to control the selection and routing elements in the flexible datapath discussed in Section 4.3.3, particularly the programmable BSs which have been designed to flexibly support multiple submatrix sizes. Meanwhile, the memory management strategies discussed in Section 4.3.2 ensure a high enough memory bandwidth for reading and writing all of the inter-node LLRs required each clock cycle, for any of the supported PCMs.

Two further optimisations have been presented to reduce the hardware requirements of decoders having the proposed architicture, based on some key observations about the PCMs. Firstly, the number of CNs in the factor graph (respectively rows in the PCM) will always be lower than the number of VNs (respectively columns) by a factor of $(1 - R)$. This motivates the discussion in Section 4.3.1.1 of reducing the CNPU parallelism ψ_c relative to the VNPU parallelism ψ. Furthermore, the majority of the LDPC code families found in modern communications standards vary the coding rate

R by decreasing the number of CNs in the factor graph whilst keeping the number of VNs the same. This has the additional effect of increasing the average CN degree d_c, as described in Section 4.3.3.2. Accordingly, where a decoder instance requires a high number of low-degree CNPUs for some of its supported PCMs, but a low number of high-degree CNPUs for others, the flexible CNPU architecture presented in Section 4.3.4.3 may be employed to further reduce the hardware resource requirements of this aspect.

The implementation results presented in Section 4.4 demonstrate the success of the proposed architecture for a large range of use-cases. Particularly, decoders targeted at PCMs from three of the four featured code families obtain throughputs that are higher than those achieved by any other flexible FPGA-based decoder featured in Chapter 3. Note that this is achieved even without taking advantage of any specific characteristics within a particular PCM family, as is the case in several of the other published designs. The proposed architecture was also demonstrated to be capable of supporting PCMs from multiple code families simultaneously, albeit at the expected cost of larger hardware resource requirements, as discussed in Section 4.4.6. One possible method of reducing this characteristic is to use stochastic computations, in which connections between the NPUs take the form of a single bit rather than a W-bit word. This is investigated through the design and implementation of a stochastic counterpart to the fixed-point architecture proposed here, which is discussed in the following chapter.

Chapter 5

A flexible stochastic Quasi-Cyclic LDPC decoder architecture

5.1 Introduction

The fixed-point FPGA-based LDPC decoder architecture presented in Chapter 4 has been demonstrated to facilitate a very high degree of run-time flexibility, which was identified as a gap in the state of the art in FPGA-based LDPC decoder designs during the survey in Chapter 3. More specifically, this fixed-point architecture may be implemented to support any arbitrary set of Quasi-Cyclic (QC) Parity-Check Matrices (PCMs), taken from any combination of several current communications standards. However, the cost of doing so is a requirement for a higher quantity of hardware resources than for other non-flexible decoders in the published literature. This has been attributed to the large quantity of flexible routing required by flexible LDPC decoders to implement the complex connections in varying factor graphs. This problem is exacerbated by the employment of inter-node Logarithmic-Likelihood Ratio (LLR) messages formed of multiple bits, which dominate the FPGA's routing and logical resources. These high hardware utilisations also result in long critical paths through the decoder, lowering the maximum possible operating frequency and hence throughput.

The number of bits W in these messages may be increased or decreased, which would effect a corresponding increase or decrease in the hardware resource requirements of the decoder. However, it was shown in Section 3.3.2.4 that the number of bits used for

This chapter is based on the following publications:

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Flexible Stochastic LDPC Decoder Architecture," in preparation for *IEEE Access.*

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Hardware-efficient Node Processing Unit Architectures for Flexible LDPC Decoder Implementations," in *IEEE Trans. Circuits Syst. II, Exp. Briefs.*

the fixed-point LLRs in an LDPC decoder has a significant impact on its achievable error correction performance. More specifically, attempting to reduce the hardware requirements of a fixed-point decoder by decreasing the width of the fixed-point LLRs may have a significant detrimental impact on the decoder's error correction performance. This may be explained by the reduced quantity and quality of information carried in LLRs having shorter widths, and also the fact that they become saturated at maximum or minimum values more easily. This then has the effect of increasing the possibility that a small transmission error may lead to an erroneous bit with absolute certainty, which would then introduce further errors in any connected nodes.

One alternative solution to the problem of transmitting messages having a high information content without causing routing congestion is to employ stochastic computing, described in Section 2.7. Here, each inter-node message is transmitted using only a single bit at a time, which greatly decreases the complexity of the inter-node routing. However, unlike single-bit LLRs, each stochastic bit represents a small part of the value of the message being communicated. These messages are then able to gradually adapt over time, increasing their resolution and decreasing the likelihood of small errors leading to saturated erroneous values. It has hence been shown that stochastic LDPC decoders may exhibit error correction performance that is comparable to decoders employing multiple-bit fixed-point messages [14, 60, 97].

Previous research into stochastic LDPC decoders has focused on fully-parallel decoder architectures [10, 62, 91, 93]. These take advantage of the reduced hardware requirements of the stochastic processing elements and routing to increase the parallelism far beyond what is realistically achievable with fixed-point decoders, granting the potential for increased processing throughputs. However, as discussed previously, fully-parallel decoders must also implement fixed inter-node routing, eliminating any potential for true run-time flexibility. In the case of fixed-point decoders, this run-time flexibility is often realised through the use of a partially-parallel architecture. However, the complex memory operations required to implement stochastic bit streams has previously inhibited the development of any partially-parallel stochastic decoders. Note that the design published in [97] supports run-time flexibility over the four PCMs in the WPAN family discussed in Section 4.4.4, using a fully-parallel architecture. However, this is only achieved by taking advantage of certain structures that only exist within this code family, and so would not be able to present true run-time flexibility for any other PCMs.

Against this background, in this chapter we present a novel stochastic FPGA-based LDPC decoder architecture, which is the first reported example of a stochastic decoder having a partially-parallel structure. Furthermore, many of the techniques and optimisations described for the proposed flexible fixed-point architecture in Section 4.3 have been adapted and used again in the stochastic architecture described here. Consequently, the proposed architecture also represents the first example of a stochastic LDPC decoder having true multi-standard run-time flexibility. Once again, this level of flexibility accommodates any QC PCM, including those of the state-of-the-art communications standards discussed in Section 4.4, with run-time switching between PCMs achievable in only a single clock cycle.

The structure of this chapter is as follows. Firstly, Section 5.2 expands on the introduction to stochastic decoding presented in Section 2.7, and suggests a novel interpretation of the layered belief propagation decoding schedule described in Section 2.4.1.2 that is applicable to stochastic decoders. Following this, Section 5.3 details the proposed flexible stochastic decoder architecture, including both high-level discussions of its overall structure, and in-depth descriptions of its most complex and most novel components. Section 5.4 then characterises parametrisations of this architecture supporting a variety of PCM sets, and also makes comparisons between the characteristics of the two architectures proposed in this thesis. Finally, concluding remarks are offered in Section 5.5.

5.2 Stochastic decoding

This section expands on the introduction to stochastic decoding presented in Section 2.7 to provide context to the decoder architecture presented in Section 5.3. More specifically, Section 5.2.1 summarises the calculations and implementations of previous stochastic LDPC decoders, before Section 5.2.2 presents a novel variation on traditional layered LDPC decoding which is compatible with stochastic computations.

5.2.1 Stochastic computation

The fundamental principles of stochastic computation were originally described in Section 2.7, but are summarised again here for convenience. In stochastic computing, a probability P is represented by a serial stream of bits, in which the fraction of bits that adopt the value 1 is equal to P. These streams, known as Bernoulli sequences, are generated by statistically independent (psuedo-)random processes, meaning that the 1s may appear within the stream in any position.

Within the context of an LDPC decoder, stochastic numbers may be used to represent the messages exchanged between Variable Nodes (VNs) and Check Nodes (CNs) over the edges of the factor graph. More specifically, random number generators may be employed to convert received channel probabilities into stochastic bit streams in the VNs. These bits are then used to perform the stochastic versions of the VN update equation (2.23) and the CN update equation (2.17) in the manner described in Section 2.7.2. The decoding of each frame may typically last 100s or 1000s of Decoding Cycles (DCs) [94], where each DC represents the generation of one new bit in every stochastic bit stream. The bit-serial nature of the stochastic messages facilitates a drastic reduction in the inter-node routing complexity [91,97].

5.2.2 Layered stochastic decoding

As described in Section 2.4.1.2, traditional layered LDPC decoding is typically performed in fixed-point LDPC decoders in a row-parallel fashion [47]. More explicitly, the PCM is subdivided into horizontal layers, each comprising a number ϕ of rows. During the decoding process, the CNs associated with every row in a layer are processed simultaneously. Once the CNs within a layer have been processed, their results are then immediately used to update their connected VNs. Consequently, the updated VN results are then available to be used by the next layer of CNs, and so on [48].

In this work, we propose for the first time a variation of layered LDPC decoding which is compatible with stochastic LDPC computations. Here, column-parallel processing is employed, in which the PCM is subdivided into vertical layers, each comprising a number ψ of columns. Similarly to traditional layered decoding, all of the VNs associated with each layer are processed in parallel, and their results are then immediately used to update their connected CNs before the next vertical layer is processed.

In the case of stochastic LDPC decoding, the schedule may be defined mathematically as follows. Each CN C_i, $1 \leq i \leq m$, stores a single-bit stochastic message β_i, which it broadcasts to all connected VNs V_j. When VN V_j is activated in DC T, for each connected CN C_i, calculate

$$\beta^T_{i \to j} = \beta_i \oplus \alpha^{T-1}_{j \to i}, \tag{5.1}$$

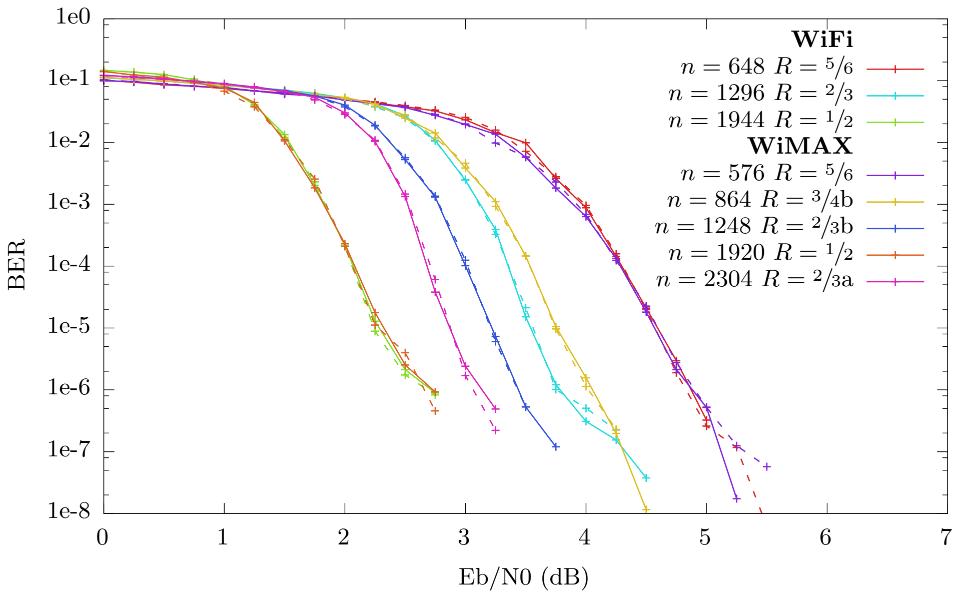
where $\alpha^{T-1}_{j \to i}$ is the message that was calculated by V_j for C_i in the previous DC $T-1$, and $\oplus$ is the binary XOR operation. VN V_j may then calculate new outgoing messages as normal,

$$\alpha^T_{j \to i} = F_{VN}(\beta^T_{i \to j}, P_{ch}), \tag{5.2}$$

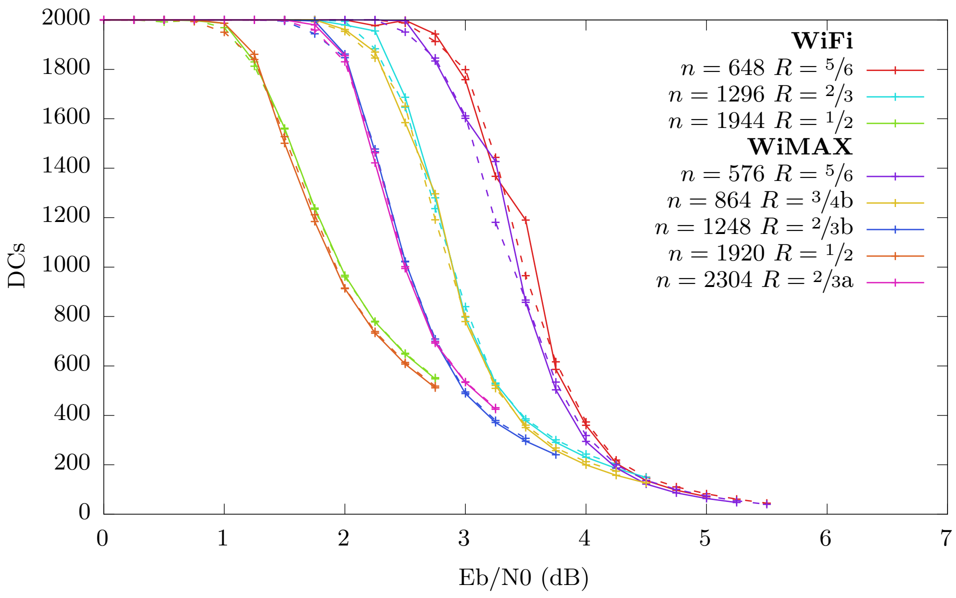
where the stochastic VN update function $VN(\cdot)$ is described in [89] and Section 2.7.2 (2.23). Here, the channel input probability P_{ch} is represented as a stochastic bit stream generated by the decoder, which effectively functions as an additional input bit to the VN update. Finally, the updated value of the broadcast CN message β_i may be calculated for each connected C_i, according to

$$\beta_i = \alpha^T_{j \to i} \oplus \beta^T_{i \to j}. \tag{5.3}$$

Through extensive Bit Error Rate (BER) simulations, we have identified that the performance of this update schedule is equivalent to the flooding update schedule, as shown in Figure 5.1. Here, the performance of a variety of codes are demonstrated, both with flooding (solid lines) and the proposed layered decoding schedule (dashed lines). These results are taken from the bit-accurate performance simulations used to characterise the performance of the proposed decoder in Section 5.4, simulating Binary Phase-Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel, with



(a) BER performance



(b) Average number of DCs

FIGURE 5.1: Comparison of flooding (solid lines) and layered (dashed lines) decoding schedules for stochastic decoding.

a maximum limit of 2000 DCs per frame and 70 frame errors per point. Figure 5.1(a) plots the BER performance, whilst Figure 5.1(b) plots the average number of DCs used per frame. It may hence be seen that regardless of the coding rate, frame length or PCM structure, the difference between the layered and flooding decoding schedules is negligable at virtually all E_b/N_0 levels.

5.3 The proposed stochastic decoder architecture

In this section, we detail the proposed FPGA-based stochastic LDPC decoder architecture, which has a similarly high level of run-time and design-time flexibility to the fixed-point LDPC decoder presented in Section 4.3. More specifically, the architecture presented here is again discussed in generalised terms, based on the properties of any set of one or more QC LDPC PCMs. This framework may then be used to create specific instances of a stochastic FPGA-based LDPC decoder which supports run-time flexibility over any chosen QC PCM set.

A block diagram of the proposed stochastic architecture is presented in Figure 5.2. By

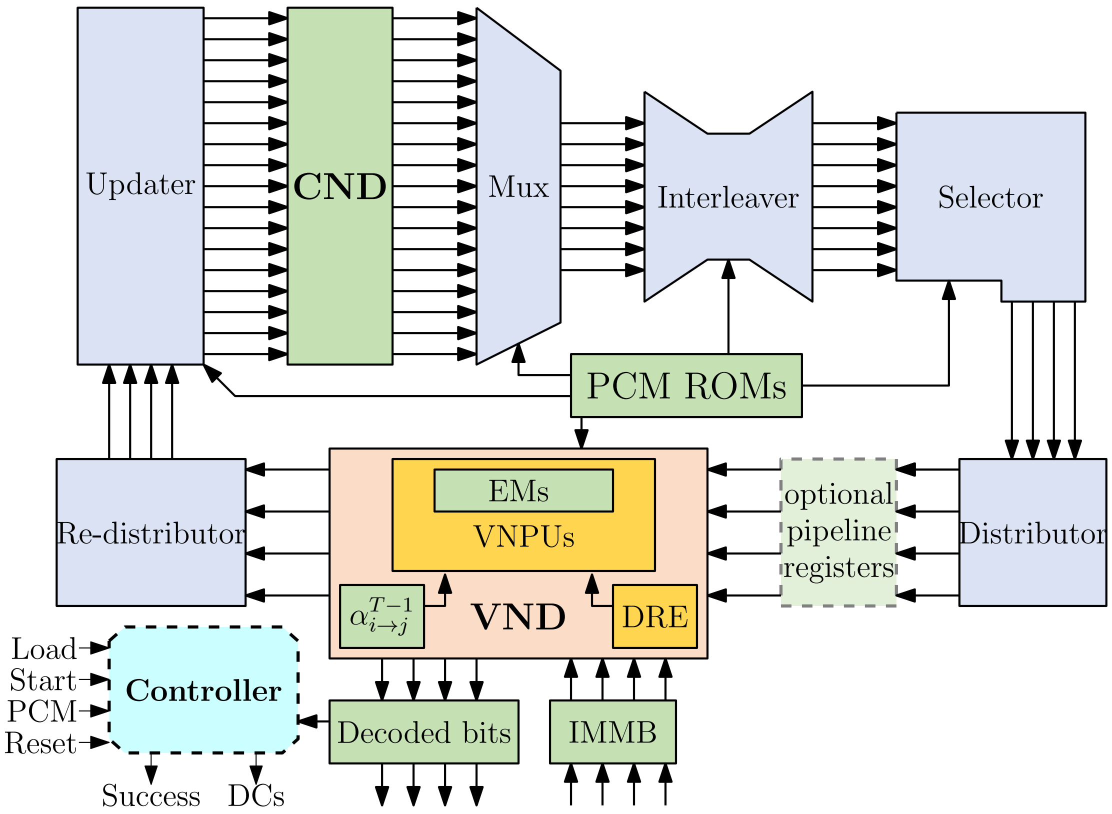


FIGURE 5.2: Block diagram of the proposed stochastic flexible architecture

comparing Figure 4.2 with Figure 5.2, it may be observed that there are many similarities, but also many differences, between the structures of the proposed fixed-point decoder architecture and the proposed stochastic decoder architecture. This section discusses these similarities and differences, the design decisions motivating them, and their effects on the resultant decoder architecture. More specifically, Section 5.3.1 discusses the degree of parallelism, the decoding schedule, and the PCM ROMs. The flexible selection and routing of the messages between the Check Node Decoder (CND) and Variable Node Decoder (VND) is then discussed in Section 5.3.2. Section 5.3.3 then details the VND, which performs the stochastic VN calculations on messages stored in the CND. Finally, Section 5.3.4 discusses the controller and early stopping detection mechanisms employed by the proposed decoder architecture.

5.3.1 Parallelism and decoding schedule

In line with the discussions presented in Section 4.2, the proposed stochastic flexible LDPC decoder architecture presented here is again partially-parallel in nature. This level of parallelism is ideally suited to flexible decoders, as it accommodates higher processing throughputs than fully-serial decoders, whilst also avoiding the complexity of simultaneously flexibly routing every edge within the set of supported factor graphs, as required by fully-parallel decoders. Furthermore, the degree of parallelism, ψ, may again be varied in the range $1 \leq \psi \leq Z$, where Z represents the maximum value of the QC expansion factor z within the supported PCM set[2]. Doing so facilitates a flexible trade-off between hardware resource usage and processing throughput, which will be illustrated further in Section 5.4. This trade-off is controlled according to (4.3) by the integer parallelism reduction factor Q introduced in Section 4.3.1.1. More specifically, the decoder parameter Q subdivides each block-column of each QC PCM having size z into q layers of size ψ, according to (4.4).

5.3.1.1 Schedule

The flexible fixed-point architecture of Section 4.3 adopts a parallelised subdivided flooding schedule, which utilises both a CND and VND in parallel, each having their own datapath and memory structures. Doing so benefits from an increased flexibility to add pipelining stages throughout the datapaths, at the expense of a large hardware resource requirement which is dominated by the routing elements, as discussed in Section 4.4.

Conversely, the flexible stochastic architecture proposed here uses the layered decoding schedule described in Section 5.2.2. The motivation for doing so is to benefit further from the reduction in routing complexity offered by the use of single-bit stochastic messages, by only implementing a single datapath. More specifically, the proposed architecture implements a CND comprised of M D-type Flip-Flops (DFFs), which are used to store the current values of β_i from (5.3). The flexible routing architecture depicted in Figure 5.2 and described further in Section 5.3.2 connects the CND to the VND, which comprises ψ Variable Node Processing Units (VNPUs), where ψ is the degree of parallelism described previously. Accordingly, these VNPUs process a layer of ψ columns of the PCM during each clock cycle according to (5.1) and (5.2), with the

[2]The reader is reminded of the notation introduced in Chapter 4 to describe the parameters of a PCM set, namely N, N_B, M, M_B, Z, D_V, and D_C, which are used to denote the maximum values of n, n_b, m, m_b, z, d_v, and d_c within the set, respectively.

updated values of β_i being written back to the CND immediately. Each DC therefore requires $t_{DC} = n_b \times q$ clock cycles.[3]

It may hence be seen that, unlike in a decoder having a separate VND and CND, this decoding schedule never requires the row-centric representation of the messages within a PCM block described in Section 4.2. Instead, the VND outputs for each block-row may be stored in the CND in their current rotation, without being shifted back. The differences between subsequent shift values on each block-row of the PCM may therefore be used to incrementally shift each block of values each time they are required by the VND. Doing so permits the use of only one interleaver module, whilst still facilitating a processing parallelism of up to Z columns every clock cycle.

5.3.1.2 Pipelining

It was stated previously in Section 2.5.3 that adding pipelining registers within the datapath of a fixed-point layered decoder may decrease its error correction performance. This may be attributed to the fact that the updates from a recent layer may not have been stored before they are required by the next layer, resulting in an older message being used instead. When employing multi-bit fixed-point messages, these older messages may represent the same bit value with a different confidence (i.e. have the same sign with a different magnitude), or a different but equally uncertain bit value (i.e. switch sign but remain at low magnitude). However, it is unlikely that old messages will differ entirely from the updated message in a significant way. Conversely, in stochastic layered decoding, due to the nature of the binary operations used, each layer's message updates will either completely change the previous message or leave it the same. Due to this fact, the stochastic layered decoding schedule presented in Section 5.2.2 requires every previous layer's updates to have been stored before the next layer may begin. This principle has been verified through extensive BER simulations, in which delaying the update of a single layer leads to a complete failure of the decoding process, even at high Signal to Noise Ratio (SNR) levels where decoding would otherwise succeed in only a few DCs.

Intuitively, this fact alone would prohibit the use of any pipelining in the datapath of the proposed architecture, resulting in a very long critical path and hence low maximum operating frequency f_{max}.[4] However, there are certain conditions under which adding a single pipeline stage to the proposed architecture is possible, without negatively affecting the BER performance. When this is the case, the optional pipeline stage indicated

[3]Note that unlike in previous stochastic decoders [10, 62] each DC in the proposed architecture requires multiple clock cycles. However, the term DC is still used throughout this description in order to differentiate from a decoding iteration in a fixed-point decoder.

[4]An alternative solution would be to add any number x of pipeline stages and utilise $x+1$ clock cycles to process each layer before moving on to the next. However, this would only improve the processing throughput if it also increased f_{max} by a factor of $x + 1$, which would be unlikely due to other sources of signal propagation delay such as I/O timing.

in Figure 5.2 approximately halfway along the architecture's critical path may be synthesised. Doing so drastically reduces the maximum propagation delay of signals within the proposed architecture, increasing the maximum operating frequency.

The possibility of adding this pipeline stage, without causing old messages to be read from the CND, depends firstly on the observation that the block-columns of a PCM may be processed in any order [142], as long as they are each processed exactly once within each DC. Secondly, it is noted that for some PCMs, there exists a column-wise permutation such that every block-column exhibits column-orthogonality with respect to its two neighbouring columns. When processed in this order, adding a pipelining stage during the datapath has no effect on the decoding performance, due to the fact that the updates from each layer are not required until at least two clock cycles later. Furthermore, when such a permutation does not naturally exist, it may be possible to create one by artificially inserting a small number τ of empty "stall" layers between non-orthogonal columns. This principle is illustrated in Figure 5.3, in which a QC base PCM $\mathbf{H}_b$ is re-shuffled and $\tau = 2$ stall layers are inserted, in order to make it compatible with including the optional pipeline stage. Note that in this example, the shift values of

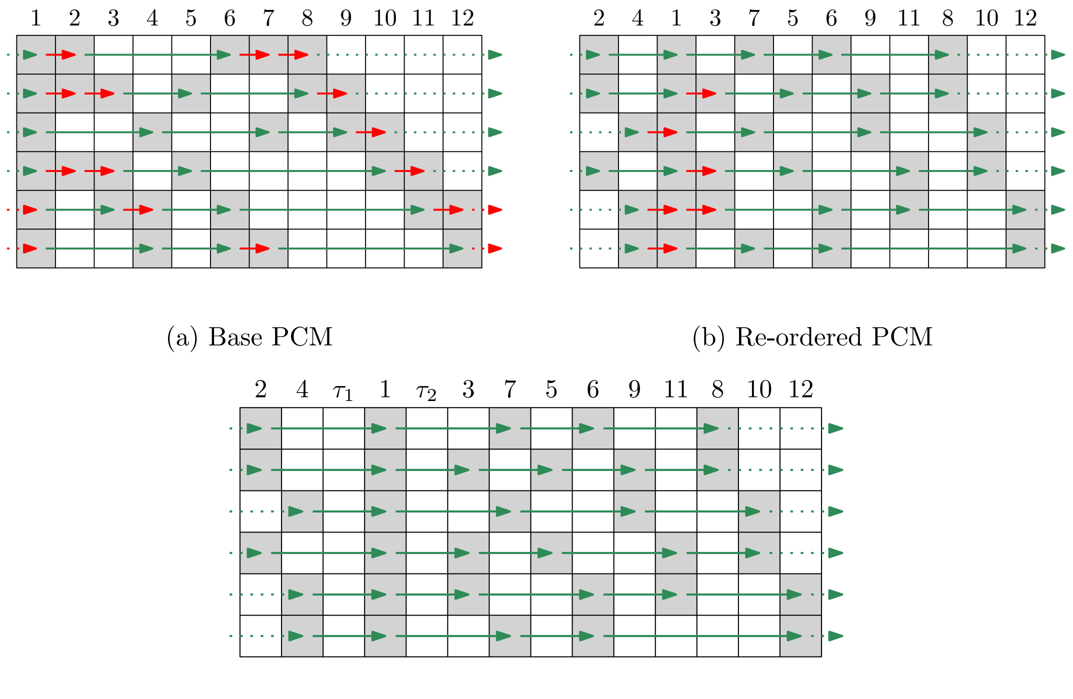


FIGURE 5.3: Example of modifying a base PCM to give it the property of column-orthogonality

the non-null locations of $\mathbf{H}_b$ are not important, and are hence depicted as grey blocks. The arrows in Figure 5.3 represent the flow of data throughout each decoding cycle, where red arrows indicate clashing (neighbouring) non-null submatrices. Green arrows, therefore, indicate that there is at least 1 clock cycle for the updated messages to get back to the CND before the next time they are required by the VND.

The addition of τ stall layers increases the number of clock cycles required per DC by τ, and the value of τ depends on the specific locations of the non-null entries of each QC base PCM $\mathbf{H}_b$. However, in some cases, the increase to the maximum operating frequency f_{max} that results from adding a pipeline stage is great enough that the overall processing throughput still increases. This decision may be characterised mathematically using the processing throughput equation (4.8) presented in Section 4.4.1. Let f'_{max} represent the increased maximum operating frequency facilitated by synthesising the optional pipeline stage in the architecture of Figure 5.2. It may hence be seen that the processing throughput is improved by using the pipeline stage when

$$\frac{f'_{max} \times n \times R}{(t_{DC} + \tau) \times I_a} > \frac{f_{max} \times n \times R}{t_{DC} \times I_a}, \tag{5.4}$$

which may be simplified to

$$t_{DC}(\frac{f'_{max}}{f_{max}} - 1) > \tau. \tag{5.5}$$

Accordingly, in the example of Figure 5.3 where $\tau = 2$ and $t_{DC} = 12$, the optional pipeline stage should only be synthesised if doing so increases the operating frequency by a factor of at least $\frac{\tau}{t_{DC}} + 1 = 1.167$. The implementation results presented in Section 5.4 include a discussion of when adding this optional pipeline stage was deemed beneficial, along with the required minimum values of τ found for each of the PCMs used in that section.

5.3.1.3 PCM ROMs

The role of the PCM ROMs in the proposed stochastic decoder architecture is similar to those in the fixed-point architecture of Section 4.3.1. Specifically, the ROMs contain values which characterise each PCM within the supported set in a hardware-optimised form, which are then used extensively throughout the rest of the architecture. The fact that these ROMs are addressed in part by the signal which selects the current active PCM is a key factor in enabling the run-time flexibility of the proposed architecture.

Similarly to Section 4.3.1, three separate sets of data are required for each PCM. Firstly, the values in the **SELECT** ROM represent the row-indices of each of the non-null submatrices within each block-column. Secondly, for each non-null submatrix in each block-column, the **SHIFT** ROM stores the difference between its shift value and that of the previous non-null submatrix in the same block-row, as discussed in Section 5.3.1.1. This value can be calculated at design-time and stored as described in Section 6.2.2, in order to simplify the complexity of the interleaver, which is described in Section 5.3.2. Finally, the **DEGREE** ROM stores the degree of each block-column of $\mathbf{H}_b$. Note that this is different from the NONZERO ROM of the fixed-point architecture of Section 4.3.1, which stores binary values to indicate whether the values stored in the other ROMs represent non-null submatrix blocks or simply empty data. Instead, the DEGREE ROM values are used to determine the function of the VND, as will be described in Section 5.3.3.

Note that, unlike in Section 4.3.1, only one set of these three ROMs is required for each PCM, rather than one corresponding to the VND and one corresponding to the CND. This is another benefit of the single datapath architecture employed by the proposed stochastic architecture, which in turn is a result of the layered decoding schedule described in Section 5.2.2.

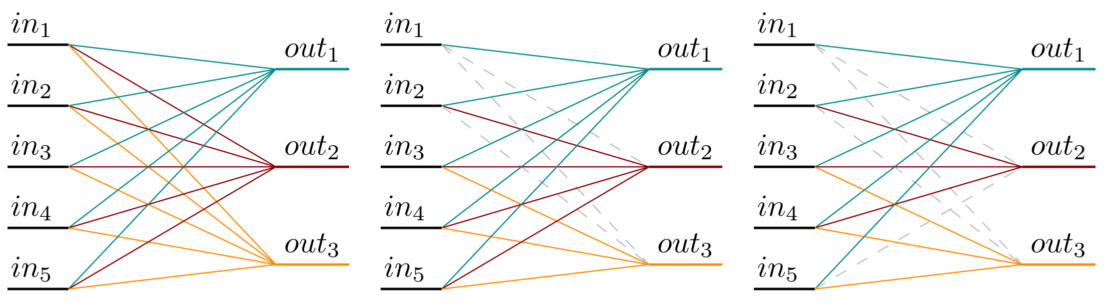
5.3.2 Datapath

As mentioned previously, the role of the datapath is to transport single-bit messages β_i from the CND registers to the ψ VNPUs in the VND, and then transport the updated messages from the VND back to the CND. It may be observed in Figure 5.2 that this routing may be subdivided into separate modules, labelled Mux, Interleaver, Selector, Distributor, Re-distributor, and Updater. The functions of each of these blocks are described in turn in this section.

The maximum number of β_i inputs required by any VNPU at any time is D_V, which is the maximum number of non-null submatrices in any block-column within the supported PCM set. Accordingly, the **Mux** selects D_V blocks of Z bits from the CND, according to the values stored in the SELECT ROM. However, rather than connecting every input to every output, the hardware resource requirements of this module may be reduced by considering the format of the SELECT ROM values, as follows. For each block-column of any PCM having degree d_v, the SELECT ROM stores D_V values. Of these, the top d_v are the block-row indices of the non-null submatrices in that block-column in ascending order, while the bottom $D_V - d_v$ are “don’t care” data. Figure 5.4 depicts an example mux with parameters $M_B = 5$, $D_V = 3$, and $Z = 1$, as well as all possible SELECT ROM values for degrees $d_v = 1$, $d_v = 2$, and $d_v = D_V = 3$ in Figure 5.4(a). Note that here a value of X represents “don’t care”, which allows the synthesis tool to optimise the hardware. Firstly, Figure 5.4(b) depicts the simplistic case in which every output is connected to every input, requiring D_V parallel M-to-1 multiplexers. However, due to the fact that the SELECT indices are stored in ascending order, and must all represent different block-rows, it may be observed that output j will never connect to inputs lower than j. This is depicted in Figure 5.4(c), where the removed connections are shown with dashed grey lines. More specifically, it may be seen in Figure 5.4(c) that output 2 never connects to input 1, and output 3 never connects to inputs 1 or 2. Furthermore, by analysing the possible SELECT ROM values in Figure 5.4(a), it may also be observed that output j will only connect to the bottom input M if $d_v = j$ is one of the degrees

$d_v = 1$					$d_v = 2$										$d_v = 3$									
1	2	3	4	5	1	1	1	1	2	2	2	3	3	4	1	1	1	1	1	1	2	2	2	3
X	X	X	X	X	2	3	4	5	3	4	5	4	5	5	2	2	2	3	3	4	3	3	4	4
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	3	4	5	4	5	5	4	5	5	5

(a) All possible values for entries in the SELECT ROM for degrees $d_v \in \{1, 2, 3\}$



(b) Simplistic mux module (c) Reduced mux (d) Further reduced mux

FIGURE 5.4: Optimisations to the mux module in the stochastic datapath

in the supported PCM set. As an example, Figure 5.4(d) depicts a mux which has been designed to only support degrees $d_v \in \{1, 3\}$. Here, input 5 does not connect to output 2 due to the fact that support for $d_v = 2$ is not required. This optimisation may also be extended for larger degree sets, where input $M - 1$ will also never connect to output j if $d_v = j + 1$ is not supported, and so on. In the trivial example of Figure 5.4, these optimisations reduce the number of connections required from 15 to 11. However, the reduction in real examples can be much greater still. For example, a mux designed to support the 12 LDPC PCMs in IEEE 802.11n [3], having $M = 12$ and $D_V = 12$, would simplistically require 144 connections. Using the optimisations described here, this number may be reduced to 73 connections, which represents a reduction of 49%.

The D_V mux output blocks are then cyclically shifted by the D_V parallel Barrel Shifters (BSs) in the **Interleaver**. These BSs are identical to those described in Section 4.3.3.3, and perform the incremental shifting described in Section 5.2.2 using the values stored in the SHIFT ROM. Following this, the **Selector** module is only required when implementing a decoder having a parallelism reduction factor of $Q > 1$. In this case, the number of parallel VNPUs ψ is less than Z, and therefore $Z - \psi$ interleaver outputs must be latched before being presented to the VND in later clock cycles. The selector hence functions as a parallel-in serial-out converter, with $D_V \times \psi$ outputs and $D_V \times Q \times \psi$ inputs, which are received once every q clock cycles. These D_V blocks of ψ bits are then rearranged by the **Distributor** into ψ blocks of D_V bits, and then optionally latched in the pipeline registers discussed in Section 5.3.1.2, before being processed by the VND. The operation of the VND is detailed in Section 5.3.3.

The ψ blocks of D_V VND output bits are then rearranged back into D_V blocks of ψ bits by the **Re-distributor** in an identical manner. Finally, the **Updater** module writes these D_V blocks of ψ bits back into the CND at the appropriate place, utilising the values in the SELECT ROM once again. The DEGREE ROM is also used here in order to ensure that only the current d_v values are written, eliminating any interference from null or "don't care" data. The optimisations described above to reduce the number of connections within the mux module can once again be employed here to minimise the hardware requirements of this module. Note that if the intermediate decoding data was stored in Block RAMs (BRAMs) (as in the fixed-point decoder of Section 4.3), messages would have to be buffered at this point until a full block of Z had arrived. Instead, implementing the CND using DFFs means that each layer can be written immediately.

5.3.3 Variable node decoder

For each of its ψ parallel VNPUs, the VND accepts D_V *a priori* input bits β_i from the connected CNs, along with a single W-bit intrinsic channel probability from the Intrinsic Message Memory Bank (IMMB), P_{ch}. Using these, the VND performs equations (5.1)–(5.3) and outputs updated extrinsic values of β_i as well as one *a posteriori* decision bit. The manner in which these operations are performed is detailed in this section. Specifically, Section 5.3.3.1 describes the processes used to convert the W-bit channel probability to a single bit, and the global broadcast CN messages β_i to specific VNPU input messages $\beta^T_{i\rightarrow j}$. Following this, the specific flexible architecture of each VNPU is described in Section 5.3.3.2 as an arrangement of 2-input 1-output subnodes, the structure of which is subsequently described in Section 5.3.3.3. Finally, Section 5.3.3.4 describes the Distributed Randomisation Engine (DRE) which is used to generate the random numbers employed throughout the VND.

5.3.3.1 VNPU inputs and outputs

The VNPU input and output arrangements are exemplified in Figure 5.5, for the case of a degree $D_V = 4$ VNPU V_j having connections to CNs C_i where $i \in \{1, 2, 3, 4\}$. The inputs β_i from these CNs may be observed on the left of Figure 5.5 alongside the W-bit channel probability P_{ch}. Meanwhile, the updated values for each β_i are depicted on the right of Figure 5.5 alongside the calculated decision bit. The internal structure of the VNPU itself is described in Section 5.3.3.2.

Following the example set in [92], the proposed stochastic decoder architecture employs $W = 6$-bit fixed-point messages to represent received intrinsic channel probabilities. These are stored in the IMMB shown in Figure 5.2 in blocks of ψ, as will be described further in Section 5.3.4. As described in [62], each time a VNPU is activated the corresponding channel probability P_{ch} is compared to a random number generated by the DRE, as described in Section 5.3.3.4. The resulting channel bit from this comparison is then used as an additional input to the VNPU, according to (5.2).

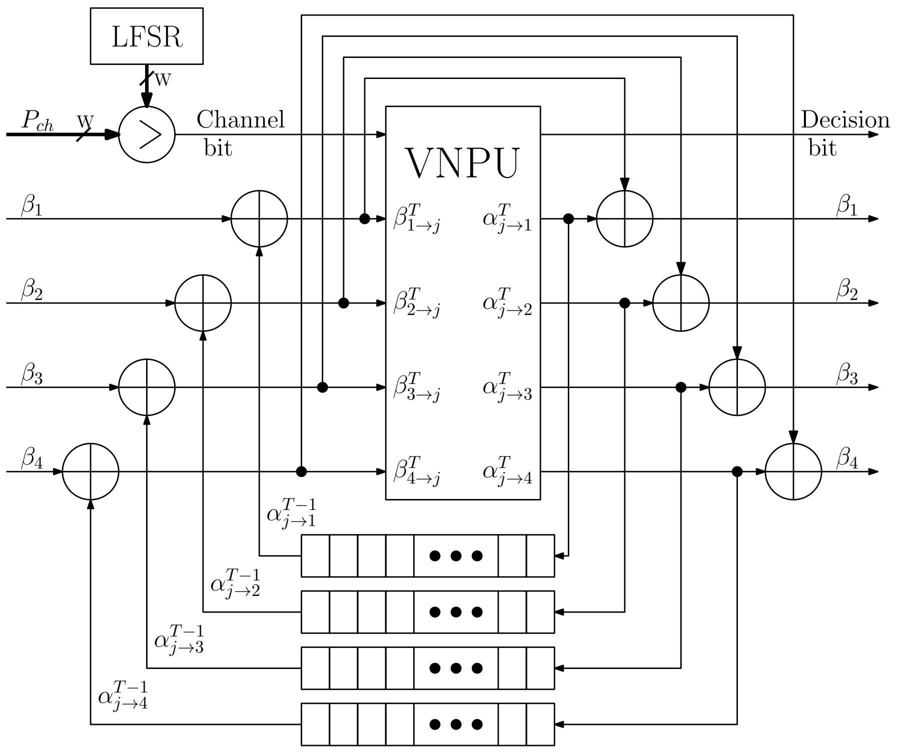


FIGURE 5.5: Calculation of inputs and outputs for an LBP stochastic VNPU

In a manner similar to that described in [47], for each VNPU the VND also maintains D_V shift registers of previous output messages $\alpha^{T-1}_{j\to i}$. These shift registers must contain $N_B \times Q$ locations, which is the number of different VNs that will be represented by each VNPU during a DC. These previous messages are used to perform calculation (5.1), to remove the contribution of VN j from the CN message β_i, leaving $\beta^T_{i\to j}$. Once a value of $\alpha^T_{j\to i}$ has been calculated by the VNPU, it is added to the back of the shift register, where it will be used again in DC $T+1$. Finally, this value is also combined with $\beta^T_{i\to j}$ in order to compute an updated CN message β_i according to equation (5.3).

5.3.3.2 Node processing unit architecture

The VNPU architectures presented in Section 4.3.4 all presume the existence of a "null" value which can be provided on all inputs i for which i is greater than the degree d_v of the active block-column. However, in the case of stochastic bit streams, a null value does not exist since both values 0 and 1 affect the represented probability, thereby necessitating an alternative VNPU architecture. Here we propose a generalisation to an architecture shown previously [21, 47, 100], extending it to any number Λ of inputs and outputs.[5] This structure, henceforth referred to as a *dual-tree* architecture, is comprised of $N_{sub} = 3 \times \Lambda - 6$ subnode blocks, each having 2 inputs and 1 output. This is identical to the number of operations required by NPUs adopting the forwards-backwards

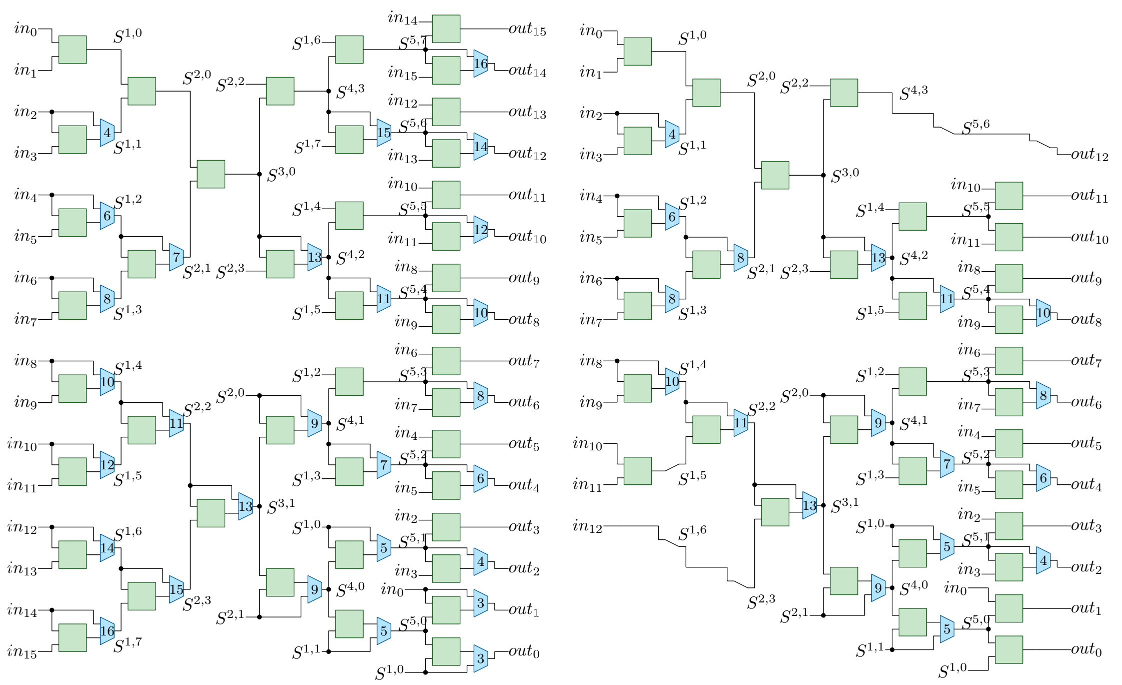
[5]Note that the number Λ of inputs and outputs for a VNPU includes the intrinsic channel bit and *a posteriori* decision bit, hence $\Lambda = D_V + 1$.

algorithm [139], and hence represents a minimal hardware requirement for NPUs implementing a non-invertible function (as described in Section 4.3.4). Note that in the proposed flexible stochastic decoder architecture, these subnodes are the flexible Edge Memories (EMs) and Internal Memories (IMs) described in Section 5.3.3.3. However, the generalisation of the dual-tree architecture permits any 2-input 1-output subnode function to be used instead, facilitating its deployment in VNPUs or Check Node Processing Units (CNPUs) in other decoder architectures.

Furthermore, we also propose that NPUs adopting the dual-tree architecture may be adapted to exhibit flexibility over the number of active inputs and outputs, through the strategic placement of multiplexers throughout the architecture. These multiplexers are placed at the outputs of certain subnodes, and may flexibly replace the subnode output with a value from an earlier active input, when one of the subnode's inputs originates from an unused node input. This subsection briefly describes the structure and run-time properties of the dual-tree architecture. A generalised method for its design-time construction, along with optimisations to reduce its hardware resource requirements, is then presented along with the offline design flow in Section 6.4.2.

Two examples of the dual-tree architecture are depicted in Figure 5.6. Firstly, Figure 5.6(a) exemplifies a VNPU having the proposed structure for the special case of flexibly supporting any number of inputs and outputs up to a maximum Λ that is a power of 2, namely $\Lambda = 16$, hence $D_V = 15$. Figure 5.6(b) then depicts an optimised VNPU which can provide the functionality of any variable node within the 12 PCMs of the IEEE 802.11n LDPC code [3], where $\Lambda = 13$. In both examples of Figure 5.6, each shaded block represents a single 2-input 1-output subnode, while the VNPU inputs in_0 to $in_{\Lambda-1}$ are depicted on the left, with corresponding outputs out_0 to $out_{\Lambda-1}$ vertically flipped on the right. When the number of active inputs and outputs λ_a is less than Λ, data is provided to only the top λ_a inputs, in_0 to in_{λ_a-1}, and the corresponding results are presented on the bottom λ_a outputs, out_0 to out_{λ_a-1}. Note that in Figure 5.6, the channel input bit is provided as in_0, and the decision output bit may be observed at out_0. More specifically, unlike out_1 to $out_{\Lambda-1}$, out_0 is a function of all λ_a inputs, which is achieved by providing the intermediate signal $S^{1,0}$ to the bottom subnode in the right-most stage. A CNPU with a maximum degree of $D_C = 16$ would possess an equivalent structure, but the second input to the bottom subnode in the right-most stage would be in_1 rather than $S^{1,0}$. This would result in out_0 becoming a function of only $\lambda_a - 1$ of the inputs, namely in_1 to in_{λ_a-1}, as required.

The overall structure of the proposed architecture may be described as follows. The dual-tree topology is comprised of two main sections, *summing* and *combining*, each of which contains $Y = \lceil \log_2(\Lambda) \rceil - 1$ stages. Let $S^{y,x}$ represent the value of the x-th intermediate signal in the y-th stage, with $0 \leq y \leq 2Y$. Furthermore, let X^y represent the total number of intermediate signals in stage y, with $0 \leq x < X^y$. Stage $y = 0$ comprises the set of NPU inputs, and so $X^0 = \Lambda$. The summing section functions similarly to



(a) Example dual-tree VNPU with $\Lambda = 16$ (b) dual-tree VNPU supporting all PCMs in IEEE 802.11n

FIGURE 5.6: Examples of the proposed flexible dual-tree NPU architecture

a conventional binary tree, in which each intermediate signal $S^{y,x}$ is calculated as the combination of two previous signals, namely $S^{y-1,2x}$ and $S^{y-1,2x+1}$, giving $X^y = \left\lceil \frac{X^{y-1}}{2} \right\rceil$. This continues until only $X^Y = 2$ signals remain, $S^{Y,0}$ and $S^{Y,1}$, each of which contains the combination of a different set of up to $2^{\lceil \log_2(\Lambda) \rceil - 1}$ inputs.

Subsequently, in each combining section stage y, where $Y + 1 \leq y \leq 2Y$, each previous signal $S^{y-1,x}$ is connected to two subnodes and combined with two earlier signals calculated during the summing section. More specifically, each combining section signal $S^{y,x}$ is the combination of one signal from stage $y - 1$ and one from stage $2Y - y$. The number of intermediate signals X^y therefore doubles in each successive stage during the combining section, $X^y = 2X^{y-1}$. It can be readily seen that this structure facilitates an increasingly specific combination of signals, such that each output contains the combination of $\Lambda - 1$ inputs as required.

For the generalised case having Λ inputs and outputs, it may hence be seen that the total number of subnodes required by the proposed architecture is $N_{sub} = 3\Lambda - 6$, which is identical to that of the forwards-backwards architecture [139]. However, the proposed architecture has a shorter propagation delay, whilst also facilitating the flexibility mentioned previously. More specifically, the propagation delay of the forwards-backwards architecture increases linearly with $D = \Lambda - 2$, whereas for the proposed dual-tree architecture it increases logarithmically, according to

$$\begin{aligned} D &= \max\left(2\ ,\ 2 \times \lfloor \log_2(\Lambda - 1) \rfloor + \left\lfloor \frac{\Lambda - 1}{2^{\lfloor \log_2(\Lambda-1) \rfloor - 1}} \right\rfloor - 3\right) \\ &\approx 2 \times \lfloor \log_2(\Lambda - 1) \rfloor . \end{aligned} \tag{5.6}$$

As mentioned previously, the flexibility of the proposed architecture is facilitated by multiplexers placed at the outputs of key subnodes throughout the NPU. More specifically, during the summing section, multiplexers are placed at each intermediate signal $S^{y,x}$ where $x > 0$; during the combining section, they are placed at each signal where x is even, and additionally at signals where $x = 1$. These *bypass multiplexers* optionally allow the corresponding intermediate signal to replicate a value from an earlier connected stage, rather than using the subnode output. This option is exercised when the subnode result would otherwise depend upon an unused NPU input. More specifically, for each subnode calculating term $S^{y,x}$, let $\lambda_n^{y,x}$ denote the minimum value of λ_a for which both of the subnode's inputs are active. Therefore, whenever $\lambda_a < \lambda_n^{y,x}$ the multiplexer is used to bypass the corresponding subnode, ensuring that $S^{y,x}$ is not corrupted by unused inputs. The value of $\lambda_n^{y,x}$ for each signal is denoted in each multiplexer in Figure 5.6, and is calculated according to

$$\lambda_n^{y,x} = \begin{cases} x \times 2^y + 2^{y-1} + 1, & \textit{summing section} \text{ and } x > 0 \\ 2^y + 1, & \textit{summing section} \text{ and } x = 0 \\ 2^y \times (2\left\lfloor \frac{x}{2} \right\rfloor + 1), & \textit{combining section} \text{ and } x \geq 2 \\ 2 \times 2^y \times (2\left\lfloor \frac{x}{2} \right\rfloor + 1), & \textit{combining section} \text{ and } x < 2 \end{cases} . \tag{5.7}$$

Note that (5.7) still holds for signals which do not require bypass multiplexers. In these cases, when $\lambda_a < \lambda_n^{y,x}$ the erroneous result will only affect outputs out_i having $i > \lambda_a$, which will not affect the decoding process. In this way, the proposed NPU architecture offers full flexibility for any number of inputs $\lambda_a \leq \Lambda$, by using only an additional $2 \times (\Lambda - 2)$ multiplexers.

Furthermore, by comparing the two examples of Figure 5.6, it may be observed that Figure 5.6(b) also has a reduced number of bypass multiplexers compared to Figure 5.6(a), even beyond those removed along with their corresponding subnodes. This novel optimisation depends on the set of supported degrees, and is described further in Section 6.4.2.

5.3.3.3 Flexible stochastic subnodes

As described in Section 5.3.3.2, the proposed VNPU architecture is constructed on the basis of 2-input 1-output subnode blocks. For stochastic VNPUs, these subnodes take the form of EMs and IMs along with their associated logic [62], as discussed in Section 2.7.3.2. An example of the subnodes employed in this work is presented in Figure 5.7, and its features are explained throughout this subsection. Note that this work adopts the optimisation presented in [57] of replacing the shift register inside EMs and IMs with a ring buffer, reducing the hardware resource requirements. However, due to the fact that each VNPU represents up to $N_B \times Q$ different VNs during each DC, these subnodes must be adapted to utilise multiple independent ring buffers.

In a rudimentary case, each EM and IM could be implemented having $N_B \times Q$ ring buffers, where each is accessed according to the index of the current decoding layer. However, as discussed in Section 5.3.3.2, a subnode may be bypassed at any time, depending on its position within the dual-tree architecture and the current active number of inputs and outputs λ_a. This observation leads to an optimisation in which each subnode only requires a number of ring buffers equal to the number of times it will be activated during each DC, which varies from subnode to subnode within the same VNPU. This activation count may be derived prior to synthesis, such that each subnode is only synthesised to use the hardware resources it requires, as described further in the discussion of the offline automated design flow in Section 6.4.2.1. Futhermore, given

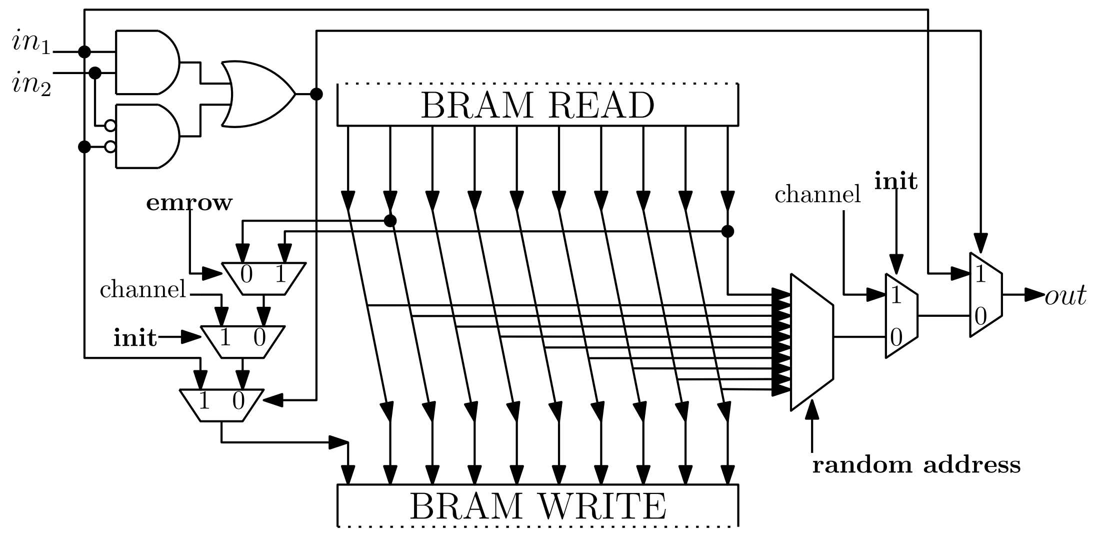


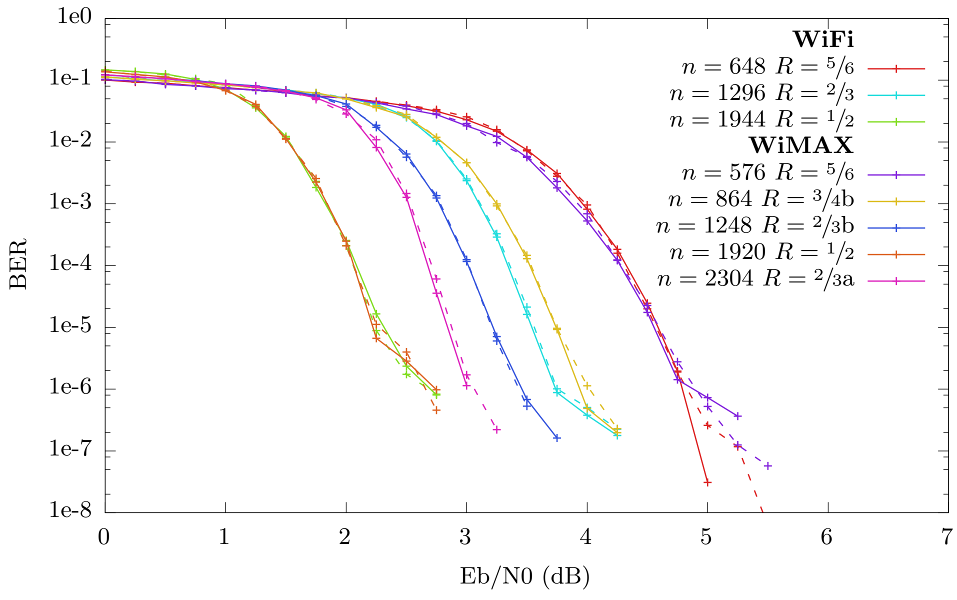
FIGURE 5.7: VNPU subnode design for the proposed flexible stochastic architecture

that only one ring buffer is read from or written to during any one clock cycle, their contents can be stored in the target FPGA's BRAMs, in order to reduce the number of DFFs required. The addresses for these BRAMs are then calculated and maintained by the VND at run-time. More specifically, for each unique value of λ_n within the VNPUs, the VND maintains a separate address counter, which is connected to the appropriate subnodes at design-time. The address counter for address λ_n is then incremented once for each clock cycle in which λ_a is greater than or equal to λ_n, and reset at the end of each DC.

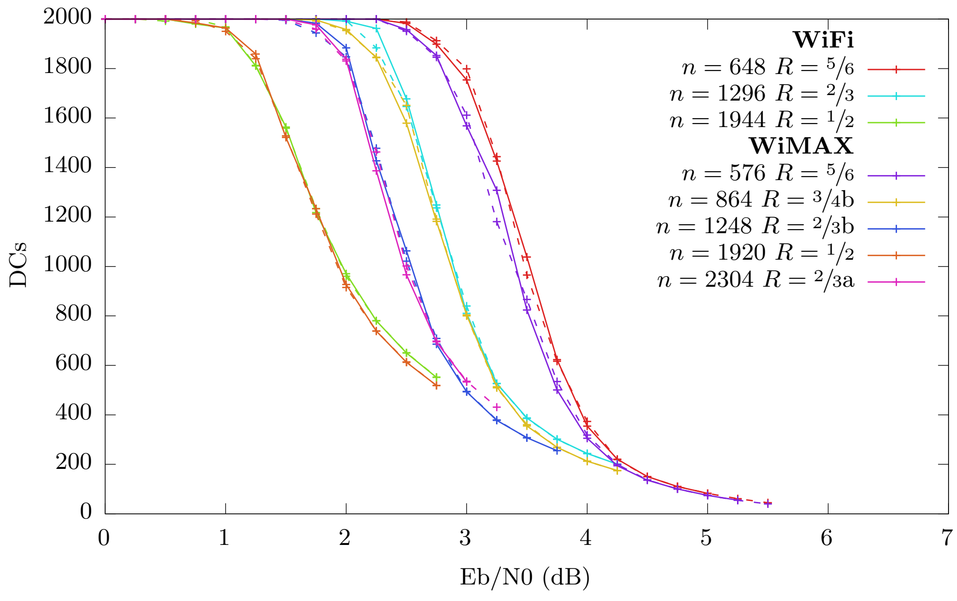
In this work, following the recommendation of [100], each EM ring buffer has a length of $L_E = 64$ bits, and each IM has a length of $L_I = 2$ bits. Clearly, every ring buffer of every subnode at the final stage $y = 2Y$ will serve as an EM every time is is activated. However, it may also be seen that some of the earlier subnodes within the dual-tree VNPU will also act as an EM during some activations and an IM during others, depending on the current value of λ_a. For example, consider the subnode at intermediate signal $S^{4,3}$ in Figure 5.6(a). Whenever $\lambda_a = 13$, intermediate signal $S^{4,3}$ will bypass its connected subnodes and connect directly to output out_{12}. Throughout the dual-tree VNPU, this may occur for any summing section subnode at intermediate signal $S^{y,x}$ having $x = 0$, and where x is odd and greater than 2 in the combining section. These subnodes must therefore be implemented having a BRAM of width L_E, but also an extra control signal, **emrow** (visible in Figure 5.7), to indicate whether or not the current activation is as an EM. This control signal is high whenever λ_a is equal to $\lambda_n^{y,x}$, and controls whether the ring buffer uses all L_E bits of the read BRAM word, or only the first L_I bits, as depicted in Figure 5.7.

It has been recommended previously [62,89] that spending some clock cycles before each new frame initialising the EM contents with corresponding channel bits improves the performance of a stochastic LDPC decoder. However, these clock cycles present a delay in which no decoding progress is being made, lowering the effective decoding throughput. This would be even more significant in the proposed partially-parallel decoder, in which only ψ EMs could be initialised during each clock cycle, requiring $t_{DC} = n_b \times q$ clock cycles to initialise one bit of every EM. Instead, we propose that equivalent decoding performance may be achieved without this delay by initialising the EMs during the first L_E DCs of each new frame. More specifically, the bits at the top of the EM ring buffers are always written to during the first L_E DCs. This writing operation uses either the new generative bit when the inputs agree (as per normal operation, described in Section 2.7.3.2), or the channel bit when the inputs disagree. Furthermore, when a generative bit is not produced during these DCs, the output bit is also set as the channel bit. Doing so avoids the possibility of reading an un-initialised bit from the ring buffer. This mode of operation may be achieved through the use of the **init** control signal, depicted in Figure 5.7, which is high for the first L_E DCs of each frame and low thereafter.

Figure 5.8 plots the performance of employing this initialisation method compared to the conventional method, in a similar analysis to that of Figure 5.1. More specifically, for each simulated code, the solid line plots the performance using the conventional initialisation method of [62], whilst the dashed line plots the corresponding performance when employing the run-time initialisation method proposed here. It may hence be seen that the proposed initialisation method has a negligible impact on the decoder's error correction capabilities or the number of DCs required to successively converge to a corrected codeword. However, this comes with the benefit of a reduced number of clock cycles required at the start of each new frame, ultimately leading to increased throughput and decreased latency over the method of [62].



(a) BER performance



(b) Average number of DCs

FIGURE 5.8: Performance of the proposed EM initialisation method (dashed lines), with respect to the original method from [62] (solid lines).

5.3.3.4 Distributed randomisation engine

Similarly to the example of [62], the proposed stochastic decoder includes multiple randomisation engines to generate the random numbers required throughout the VND. These random numbers are used to generate the channel bits, and to select a random conservative bit from the current ring buffers in the EMs and IMs. Here, we propose the use of ψ randomisation engines, each of which is comprised of a 10-bit full-sequence Linear Feedback Shift Register (LFSR) and initialised with a different seed. Each LFSR is independently responsible for providing the psuedorandom bits used to generate the channel bit for one of the corresponding ψ VNPUs. In addition, a combination of the bits from different LFSRs is used to generate the random EM bit addresses, to ensure there is no correlation between the channel bits and the selected conservative output bits.

5.3.4 Controller

The controller of an LDPC decoder is responsible for managing the external and internal control signals, as well as determining whether to finish or continue the iterative decoding process. A discussion of the ways in which these tasks are performed in the proposed stochastic decoder is presented here.

5.3.4.1 Control signals

The control signals of the proposed stochastic decoder are very similar to those used by the proposed fixed-point decoder described in Section 4.3, as can be seen by comparing Figure 5.2 to Figure 4.2. The **load**, **reset**, **start**, and **PCM** signals work in an identical manner to their description in Section 4.3.5.1. Note that during each clock cycle whilst the load signal is high, $\psi \times W$-bit channel probabilities are loaded into the IMMB. The controller also maintains a counter of the number of DCs that have passed since the start of the current frame, which is used to generate the subnode **init** signal, as described in Section 5.3.3.3.

5.3.4.2 Early stopping detection

During each decoding cycle, ψ *a posteriori* decision bits are output from the VND. In order to determine whether these bits constitute a valid codeword, the proposed decoder also maintains the cumulative output of the m parity checks within the current PCM. These bits are reset at the start of each DC, and are stored and routed in an identical way to the CN bits β_i. Accordingly, during each DC, the appropriate $d_v \times \psi$ parity-check bits are exclusive-OR'd with the ψ VNPU decision bits, and then re-stored in the appropriate positions. At the end of each DC, if all of these bits have the value of 0, the **success** signal is set, and the decoding process ends.

5.4 Implementation results

Similarly to Section 4.4 for the fixed-point architecture of Section 4.3, this section characterises and discusses several instances of the proposed stochastic decoder architecture. Once again, the parametrisations presented here are targeted at QC PCMs from the major wireless communications standards described previously, namely IEEE 802.11n/ac (WiFi) [3], IEEE 802.16e (WiMAX) [4], IEEE 802.15.3c (referred to as WPAN) [140], and IEEE 802.11ad (WiGig) [135], and compared with relevant benchmarkers where possible. Decoders supporting PCM sets for each of these families in isolation are presented in Sections 5.4.2 — 5.4.5, before Section 5.4.6 characterises decoders supporting inter-standard flexibility. A comparison of equivalent decoder implementations having the two proposed architectures of Sections 4.3 and 5.3 is then offered in Section 5.4.7.

As in Section 4.4, each specific decoder parametrisation is identified using an abbreviated name, comprised of two letters and an identifying number. The first letter is always 'S' in this section regarding the **S**tochastic architecture, which may then be compared to the **F**ixed-point architecture results of Section 4.4. The second letter identifies the code family targeted by that decoder instance, which is be 'F' for Wi**F**i, 'M' for Wi**M**ax, 'P' for W**P**AN, 'G' for Wi**G**ig, or 'S' to represent support for **S**everal of these families simultaneously. The specific parameters of each decoder instance are discussed in the relevant sections. Firstly, however, Section 5.4.1 discusses the methods employed to measure the presented characteristics, in line with the analysis presented in Chapter 3 and Section 4.4.

5.4.1 Method

The implementation characteristics presented here were generated according to the method presented in Section 4.4.1, in order to facilitate comparisons between the fixed-point decoder architecture proposed earlier in Section 4.3, and the stochastic decoder architecture proposed here. More specifically, the Hardware Description Language (HDL) code for each separate decoder instance was generated automatically and synthesised, in order to quantify its hardware requirements and maximum operating frequency. Additionally, bit-accurate BER simulations were performed for each code, to quantify the transmission energy and number of DCs required for a BER of 10^{-4}. Once again, these simulations assumed BPSK transmission over an AWGN channel, with a maximum limit of 1000 DCs per frame, and a minimum measurement of 100 frame errors per BER measurement.

5.4.1.1 Use of pipelining

As discussed in Section 5.3.1.2, the design flow for the proposed stochastic decoder architecture must also consider whether or not to use the optional pipeline registers visible in Figure 5.2. These registers reduce the critical path length of the design, facilitating an increase to the maximum clock frequency f_{max}. However, doing so also requires each PCM to be re-ordered such that every block-column is orthogonal with respect to its two neighbouring columns. If no such permutation naturally exists, a number τ of artificial empty columns may be introduced to empty the pipeline, as depicted in Figure 5.3. The value of τ varies between PCMs, and may only be determined experimentally. The values of τ that have been found for the PCMs discussed in this section are presented in Table 5.1.

TABLE 5.1: Number of pipeline stalls required to produce column-orthogonality in the featured QC PCMs

PCM family	**Frame length(s)**	**Coding rate(s)**	τ
WiFi	All (n_1–n_3)	R_1	4
WiFi	All (n_1–n_3)	R_2	6
WiFi	n_1	R_3	11
WiFi	n_2–n_3	R_3	9
WiFi	All (n_1–n_3)	R_4	23
WiMax	All (n_1–n_{19})	R_1	0
WiMax	All (n_1–n_{19})	R_2	6
WiMax	All (n_1–n_{19})	R_3	2
WiMax	All (n_1–n_{19})	R_4	16
WiMax	All (n_1–n_{19})	R_5	12
WiMax	All (n_1–n_{19})	R_6	23
WPAN	n_1	R_1–R_2	0
WPAN	n_1	R_3	1
WPAN	n_1	R_4	32
WiGig	n_1	R_1	1
WiGig	n_1	R_2	8
WiGig	n_1	R_3	15
WiGig	n_1	R_4	16

The values in Table 5.1 illustrate the expected result that higher-rate PCMs are less amenable to column-orthogonality. This is due to the fact that all 4 of the featured code families increase the coding rate R by decreasing the number of block-rows m_b whilst keeping the number of block-columns n_b constant. The result of this is a reduction in the ratio of null submatrices to non-null submatrices, making non-overlapping columns less frequent. Accordingly, none of the PCMs featured in Table 5.1 having the highest rate for their respective family naturally posses more than one non-overlapping column, requiring a high number of stall cycles.

Unfortunately, the only way to reliably ascertain the effect of the pipelining stage on f_{max} is to synthesise two instances of each decoder, with and without it. This may be accomplished very easily using the offline design flow described in Chapter 6, which permits the user to specify whether or not to include pipelining as an extra input parameter. By using the subsequent values of f_{max} and f'_{max} along with τ from Table 5.1 in equation (5.5), the decision of which version to use can then be made formulaically. In cases where the decoder supports multiple PCMs having varying values of τ, it is possible to re-arrange (5.5) into a single expression and sum this for each supported PCM p. More explicitly, the pipelining registers are employed whenever

$$\sum_{p} \left(t_{DC,p} \left(\frac{f'_{max}}{f_{max}} - 1 \right) - \tau_p \right) > 0, \tag{5.8}$$

where $t_{DC,p}$ represents the number of clock cycles per DC of p without pipelining, and τ_p represents the value of τ for PCM p from Table 5.1.

Note that adding the pipelining registers has a negligible impact on a decoder's hardware resource requirements, despite adding $\psi \times D_V$ extra registers and extra associated control logic. This may be attributed to the fact that the shortened critical path length reduces the need for the synthesis tool to perform extra optimisations which increase f_{max} at the cost of some extra hardware, for example the duplication of high fan-out signals. During the accumulation of the results presented in this section, it was generally found that adding pipelining to a decoder instance only changed the hardware resource requirements by approximately ±1-3%, in a non-predictable way. Accordingly, the hardware resource requirements did not factor into the decision of whether or not to use the optional pipeline registers, leaving the result of (5.8) as the sole deciding factor. The presence or absence of pipelining will be reported for each considered scheme in the following discussions.

5.4.1.2 Noise-dependent scaling

As mentioned in Section 2.7, the decoding performance of stochastic LDPC decoders is naturally limited at high SNR levels due to a reduction in switching activity. This is caused by the received channel probabilities for each bit becoming increasingly close to the absolute values of $P_{ch} = 0$ or $P_{ch} = 1$, resulting in a corresponding stochastic bit stream of entirely 0s or 1s, respectively. Noise-Dependent Scaling (NDS), first presented in [60], mitigates this problem by scaling the received channel probabilities, according to equation (2.26) presented in Section 2.7.3.1. Here, γ is an empirical factor, the derivation of which has not been formally defined previously.

We have conducted an investigation into the effects of γ over multiple BER simulations, targeting a variety of code families, frame lengths, and coding rates, in order to select optimal values for a wider range of codes than those presented in [62]. Consequently, we have determined that the optimal value of γ increases with an increased coding rate R, but does not appear to be affected by changes to the frame length n, nor the format of the PCM $\mathbf{H}$. However, variations in γ can produce BER curves having varying error floor and turbo-cliff performance, such that no single value of γ may be deemed optimal in all cases for a given PCM. A full investigation into the nature of this parameter is therefore recommended for future work in Section 7.2.

As performed previously [88,92], the proposed decoder has been designed on the assumption of receiving scaled channel probabilities from the modulator. Through analysing the effects of a broad range of γ values on both the BER performance and the corresponding average numbers of DCs required, values of γ were selected for each of the coding rates in the featured PCMs. These values are presented in Table 5.2, and were employed in the bit-accurate BER simulations from which the error correction performance of the decoders presented in the following sections was characterised.

Table 5.2: Values of γ used in the simulations of the proposed decoder

Coding rate R	γ
$1/2$	0.5
$5/8$	0.65
$2/3$	0.7
$3/4$	0.9
$13/16$	1.0
$5/6$	1.1
$7/8$	1.1

5.4.2 Decoders targeted at WiFi LDPC PCMs

In order to facilitate comparisons with the results generated in Section 4.4.2, corresponding parametrisations of the proposed stochastic architecture were synthesised targeting the WiFi family of LDPC PCMs. Once again, the results from [27] and [33] provide non-flexible and flexible FPGA-based benchmarkers respectively. Decoder SF1 supports the single WiFi PCM having frame length $n_3 = 1944$ and coding rate $R_1 = 1/2$, making it directly comparable to the benchmarker from [27]. Meanwhile, decoder SF2a supports the three PCMs of varying frame lengths and the single coding rate R_1. However, this coding rate employs a high maximum variable node degree of $D_V = 12$, leading to very large VNPUs in the corresponding hardware. Accordingly, decoder SF2b was also synthesised to support the three PCMs having coding rate R_4 (where $D_V = 4$), in order to compare their characteristics. Furthermore, decoder SF3 supports the four PCMs having varying coding rates and the single frame length $n_1 = 648$, whilst decoder SF4 supports all 12 of the WiFi PCMs. The characteristics of these decoders are presented in Table 5.3 and plotted in Figure 5.9. The column indicated as t_{DC} $(+\tau)$ in Table 5.3 provides the number of clock cycles per DC for each decoder, which includes the value of τ from Table 5.1 when the decoder employs the optional pipelining registers discussed in Section 5.3.1.2. Note that, similarly to Section 4.4.2, these decoders have been generated with a value of Q chosen such that they all employ $\psi = 27$ VNPUs.

TABLE 5.3: WiFi characterisation of the proposed stochastic architecture

Decoder	Q	Pipelined?	Active PCM n	Active PCM R	t_{DC} $(+\tau)$	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
SF1	3	Yes	1944	1/2	76	1	24.9	656	2.22	437.0	2.46
SF2a	3	Yes	648	1/2	28	3	33.1	411	2.94	110.4	2.91
			1296	1/2	52		33.1	575	2.26	286.6	2.55
			1944	1/2	76		33.1	656	2.04	477.5	2.46
SF2b	3	No	648	5/6	24	3	12.0	181	13.43	40.2	4.36
			1296	5/6	48		12.0	306	7.94	136.0	4.04
			1944	5/6	72		12.0	408	5.96	272.0	3.87
SF3	1	No	648	1/2	24	4	17.3	411	3.21	100.9	2.91
			648	2/3	24		17.3	286	6.17	70.0	3.52
			648	3/4	24		17.3	247	8.01	60.7	3.76
			648	5/6	24		17.3	181	12.2	44.4	4.36
SF4	3	Yes	648	1/2	28	12	35.2	411	2.76	117.3	2.91
			648	2/3	30		35.2	286	4.95	87.3	3.52
			648	3/4	35		35.2	247	5.51	88.2	3.76
			648	5/6	47		35.2	181	6.23	86.7	4.36
			1296	1/2	52		35.2	575	2.13	304.7	2.55
			1296	2/3	54		35.2	445	3.53	244.8	3.16
			1296	3/4	57		35.2	402	4.17	233.2	3.45
			1296	5/6	71		35.2	306	4.88	221.4	4.04
			1944	1/2	76		35.2	656	1.91	507.7	2.46
			1944	2/3	78		35.2	540	3.02	429.1	3.01
			1944	3/4	81		35.2	504	3.51	415.6	3.28
			1944	5/6	95		35.2	408	4.10	395.0	3.87
[27]	-	-	1944	1/2	-	1	77.8	(10)	348	11.2	2.83
[33]	-	-	1296	2/3	-	12	19.3	(15)	95	9.09	2.49

As described previously, decoder SF1 may be directly compared with the benchmarker of [27], since they both target the same PCM having frame length $n_3 = 1944$ and coding rate $R_1 = 1/2$, without run-time flexibility. Conversely to the comparison made with decoder FF1 in Section 4.4.2, here the proposed architecture requires fewer hardware resources than [27], but at the cost of a significantly lower processing throughput. More specifically, the hardware resource requirement of decoder SF1 is 68% lower than that of [27], and the processing throughput is 99.6% lower. This low throughput is characteristic of stochastic decoders [86] due to the high number of DCs required to converge to a correct codeword. Due to the degree of randomness involved in stochastic decoding, this is exacerbated by long frame lengths, as the likelihood of there being a single bit which randomly assumes an incorrect value (even despite the influences from neighbouring nodes) in any given DC increases when the number of bits is higher. The effect of this high number of DCs is then further compounded by the partially-parallel decoder architecture, which also sacrifices decoding throughput for lower hardware resource usage by requiring multiple clock cycles per DC. For example, Table 5.3 indicates that decoder SF1 requires $T_a = 656$ DCs per frame, with each DC requiring $t_{DC} = 76$ clock cycles.

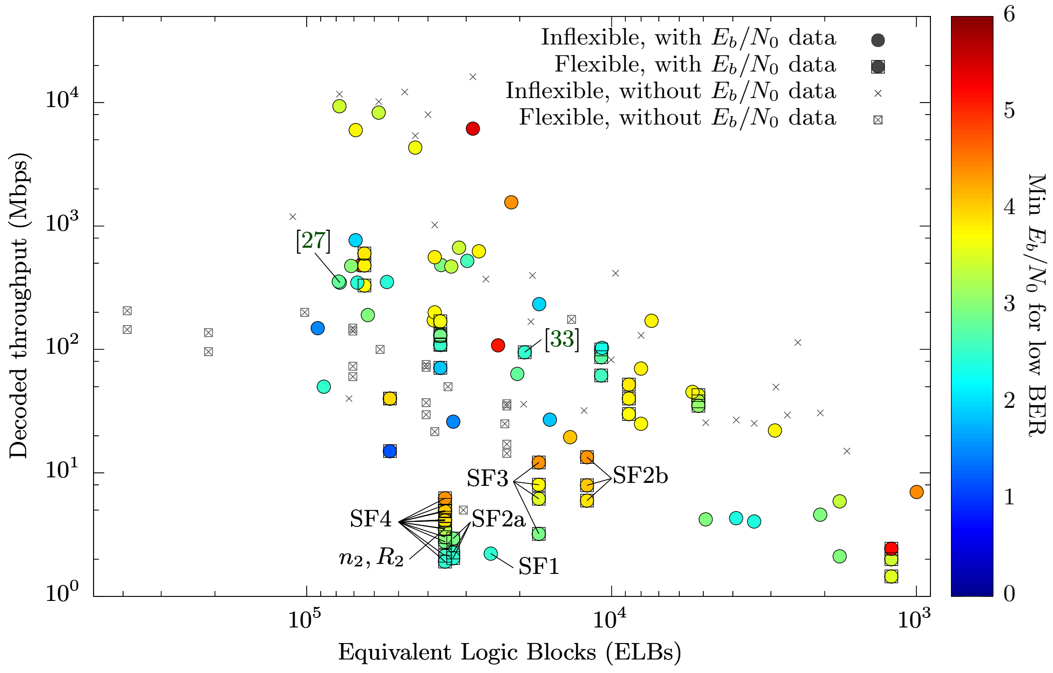


FIGURE 5.9: WiFi characterisation of the proposed stochastic architecture

The effects of the high VN degree $d_v = 12$ present in WiFi PCMs having rate $R_1 = 1/2$ may be observed by comparing the hardware requirements of decoders SF2a and SF2b. Specifically, SF2a exhibits a hardware resource requirement that is almost three times greater than that of SF2b, despite supporting an equal number of PCMs, having the same frame lengths. Note that this threefold increase is proportional to the increase in the value of D_V between the two decoders, namely $D_V = 12$ for decoder SF2a and $D_V = 4$ for decoder SF2b. This may be attributed to the fact that D_V is used as a dimension of the connections and modules throughout the entire decoder datapath described in Section 5.3.2, such that the decoder as a whole grows in size proportionally to D_V. Note that decoder SF2b also achieves approximately a 3–4 times higher processing throughput than that of decoder SF2b, whilst requiring approximately 1.5 dB higher transmission energy. This may be mainly attributed to its higher coding rate R, which employs a greater proportion of message bits to parity bits in transmitted frames. More explicitly, codes with a higher coding rate comprise a lower number of parity bits m for the number of message bits k, and therefore require a higher signal to noise ratio to be capable of correcting transmission errors. However, even when increasing the average number of DCs per frame for PCMs with low error correction performance, the high coding rate still results in a higher decoded processing throughput due to the larger number of message bits.

These factors also help to explain the performance of the flexible decoder SF4, which has a 82.4% higher hardware resource requirement and 27 times lower processing throughput than those of [33]. However, as mentioned in Section 4.4.2, the results in [33] take advantage of multiple optimisations which are only applicable to the specific set of PCMs in the WiFi family. Meanwhile, the proposed stochastic decoder is built in a way which makes it compatible with any QC PCM set, as will be demonstrated in the following sections. It is therefore argued that SF4 achieves a far larger degree of design-time flexibility than that of [33], which is not considered in the results presented in Table 5.3 and Figure 5.9.

5.4.3 Decoders targeted at WiMAX LDPC PCMs

The WiMAX code family contains 114 QC PCMs, namely the combination of 6 coding rates and 19 frame lengths. It was shown in Section 4.4.3 that the high granularity of z values within this family makes it difficult to choose a value of Q that balances high throughput at large frame lengths against efficient hardware utilisation at shorter frame lengths. However, the proposed stochastic architecture benefits from reduced hardware requirements for each of its elements, compared to the proposed fixed-point architecture of Section 4.3. Accordingly, a higher degree of parallelism may be utilised without suffering from excessive hardware requirements for the decoder as a whole, which may help to reduce the impact of the higher number of DCs required per frame. In line with the discussions of Section 4.4.3, the decoder instances that have been implemented and characterised are as follows. Firstly, decoder SM1 supports the single PCM having rate $R_1 = {}^1/_2$ and frame length $n_6 = 1056$, which makes it comparable with the fully-parallel stochastic decoder of [62]. Secondly, decoder SM2 supports all 19 of the PCMs having the highest coding rate $R_6 = {}^5/_6$, whilst decoder SM3 supports the 6 PCMs having the lowest frame length, $n_1 = 576$. The PCM at the intersection of these two sets, having rate R_6 and frame length n_1, is then supported by two additional non-flexible decoders, SM4a and SM4b, in order to provide context to the characteristics of SM2 and SM3. Here, SM4a has $Q = 1$, similarly to the other decoder implementations listed here. Meanwhile, SM4b has $Q = 8$, reducing its level of parallelism to $\psi = 4$ VNPUs. Finally, decoder SM5 supports all 114 of the PCMs in the WiMAX family. As discussed in Section 4.4.3, extra assumptions about the results from [20] are made here in order to use it as a run-time flexible benchmarker. The characteristics of these decoders are presented in Table 5.4 and Figure 5.10.

Table 5.4: WiMAX characterisation of the proposed stochastic architecture

Decoder	Q	Pipelined?	Active PCM n	Active PCM R	t_{DC} $(+\tau)$	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
SM1	1	Yes	1056	1/2	24	1	15.3	513	5.87	90.0	2.57
SM2	1	No	576	5/6	24	19	29.2	169	10.84	44.3	4.35
			1248	5/6	24		29.2	303	13.07	79.6	4.01
			2304	5/6	24		29.2	444	16.48	116.5	3.83
SM3	1	No	576	1/2	24	6	10.0	375	3.49	82.6	2.87
			576	2/3b	24		10.0	264	6.62	58.0	3.51
			576	3/4a	24		10.0	225	8.73	49.5	3.82
			576	5/6	24		10.0	169	12.94	37.1	4.35
SM4a	1	No	576	5/6	24	1	5.7	169	15.29	31.4	4.35
SM4b	8	Yes	576	5/6	215	1	2.0	169	2.54	189.3	4.35
SM5	1	No	576	1/2	24	114	76.8	375	2.09	137.8	2.87
			576	2/3b	24		76.8	264	3.97	96.8	3.51
			576	3/4a	24		76.8	225	5.23	82.6	3.82
			576	5/6	24		76.8	169	7.75	61.9	4.35
			1248	1/2	24		76.8	547	3.11	200.8	2.51
			1248	2/3b	24		76.8	427	5.30	156.9	3.13
			1248	3/4a	24		76.8	378	6.75	138.6	3.46
			1248	5/6	24		76.8	303	9.35	111.3	4.01
			2304	1/2	24		76.8	671	4.67	246.4	2.38
			2304	2/3b	24		76.8	573	7.30	210.4	2.93
			2304	3/4a	24		76.8	518	9.09	190.0	3.29
			2304	5/6	24		76.8	444	11.79	162.9	3.83
[62]	-	-	1056	1/2	1	1	68.2	NA	348	1.5	2.45
[20]	-	-	NA	1/2	NA	114	38	20	10.4	NA	NA

Table 5.4 and Figure 5.10 show that decoder SM1 has a 77.6% lower hardware resource requirement than [62], but also suffers from a 98.3% lower processing throughput. Again, this may be attributed mainly to the fact that SM1 adopts a partially-parallel architecture, whereas [62] is fully-parallel. However, alongside a reduced hardware resource requirement, partially-parallel decoders also benefit from having the capability to implement run-time flexibility without major modifications, which is not the case for fully-parallel architectures such as that of [62].

By comparing the characteristics of SM2 and SM3 to SM4a, it may be observed that the cost incurred by supporting only multiple frame lengths is greater than that incurred by supporting only multiple coding rates. More specifically, SM3 suffers from a 75.4% higher resource requirement than SM4a, whilst for SM2 it is 412% greater. Note, however, that

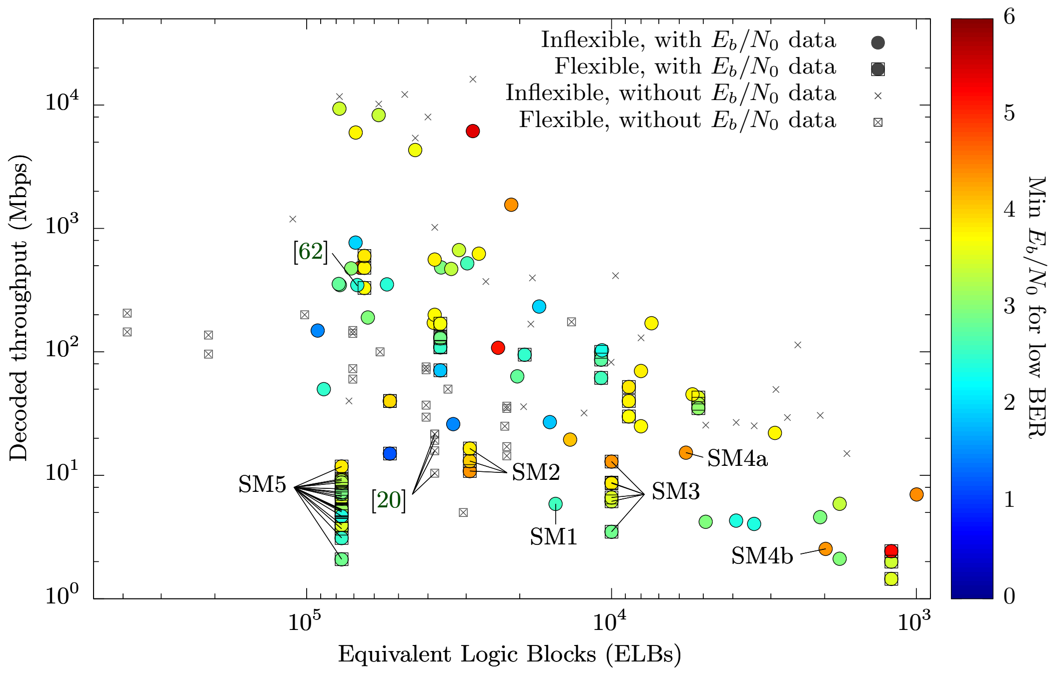


FIGURE 5.10: WiMAX characterisation of the proposed stochastic architecture

where SM3 supports 6 PCMs, SM2 supports 19, which provides another example of the trade-off between flexibility and hardware resource usage. Furthermore, it may be observed that the data point corresponding to the single PCM supported by SM4a is clearly visible in Figure 5.10 amongst the results from SM2 and SM3 as the darkest orange circle, having the highest throughput for SM3 and the lowest for SM2. Note that the processing throughput appears to decrease linearly with the hardware resource requirements, which may be attributed to the reduced clock frequency that occurs as a result of higher FPGA resource utilisation. If the three results sets were normalised such that these points were aligned horizontally, the plot of Figure 5.10 would then serve as an example of the fact that higher throughputs may be obtained by increasing the frame length (as seen from SM2) and increasing the coding rate (as seen from SM3).

Additionally, comparing SM4a to SM4b provides another example of the efficacy of the parallelism reduction factor Q. More specifically, SM4b has an eight times lower parallelism than SM4a, which manifests as a six times lower throughput and a three times lower hardware requirement. Meanwhile, decoder SM5 achieves throughputs that range from 2.09–11.79 Mbps, which are approximately the same as those from the benchmarker of [20]. However, this comes at the cost of approximately doubled hardware resource requirements. This may be attributed again to the highly complex BSs necessitated by the high number of frame lengths in the WiMAX code family. The design discussed in Section 4.3.3.3 was chosen to reduce the hardware resource requirements of BSs having compatibility for any set of z values up to Z. However, it is possible that alternative designs may exist that could take advantage of any regularity in a particular set of z values. This presents an additional argument in favour of the offline design flow presented in Chapter 6, which has the capability to generate a variety of differing module structures dependent upon the specified supported PCM set.

5.4.4 Decoders targeted at WPAN LDPC PCMs

The WPAN family of codes contains 4 PCMs having $n = 672$, $z = 21$, and $n_b = 32$, across four coding rates. The decoder of [27] provides an inflexible benchmarker for the PCM having rate $R_1 = 1/2$; accordingly, decoder SP1 was implemented to do the same, in order to compare the proposed stochastic architecture against another recent fixed-point decoder. Furthermore, in order to demonstrate the effects of the optional pipeline registers, two instances of a decoder that flexibly supports all 4 WPAN PCMs were implemented: SP2a, which does not use pipelining, and SP2b, which does. The characteristics of these decoders may be observed in Table 5.5 and Figure 5.11.

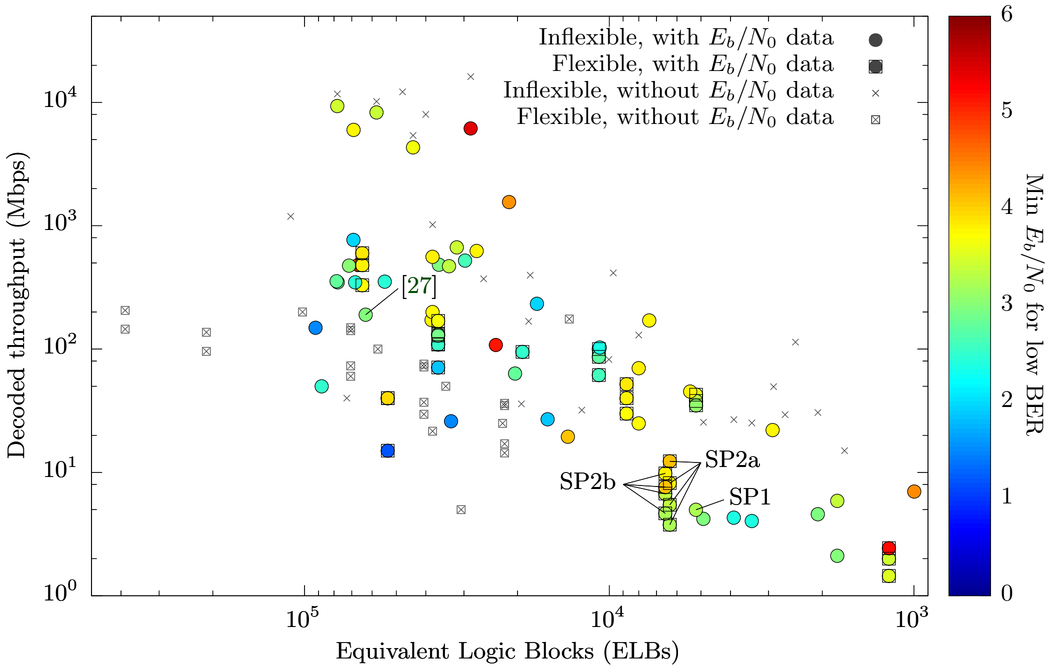


FIGURE 5.11: WPAN characterisation of the proposed stochastic architecture

By comparing SP1 to [27], the proposed stochastic architecture's reduction to both throughput and hardware resource requirements compared to other decoders is again illustrated. Here, SP1 has a 12.1 times lower hardware resource requirement and a 38.2 times lower processing throughput than [27], though only a 12.7 times higher latency due to the fact that the throughput is derived from a single frame.

Note that the high value of D_C in the highest-rate code of the WPAN family does not affect the proposed stochastic architecture in the same way that it does for the fixed-point case, as discussed in Section 4.4.4. This is due to the layered architecture described in Section 5.3.1.1, which performs the CN computations serially each clock cycle, rather

TABLE 5.5: WPAN characterisation of the proposed stochastic architecture

Decoder	Q	Pipelined?	Active PCM n	Active PCM R	t_{DC} $(+\tau)$	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
SP1	1	Yes	672	1/2	32	1	5.2	348	4.98	67.5	3.21
SP2a	1	No	672	1/2	32	4	6.3	348	3.77	89	3.21
			672	5/8	32		6.3	301	5.46	76.9	3.34
			672	3/4	32		6.3	241	8.2	61.5	3.79
			672	7/8	32		6.3	187	12.32	47.7	4.08
SP2b	1	Yes	672	1/2	32	4	6.5	348	4.66	72.1	3.21
			672	5/8	32		6.5	301	6.74	62.3	3.34
			672	3/4	33		6.5	241	9.82	51.3	3.79
			672	7/8	64		6.5	187	7.61	77.3	4.08
[27]	-	-	672	1/2	-	1	63.0	10	190	5.31	2.97

than in parallel. This is evidenced by the low hardware resource requirements of all three of the decoder parametrisations listed here, which represent some of the smallest FPGA-based partially-parallel LDPC decoders supporting run-time flexibility available in the published literature.

As discussed in Section 5.4.1, the decision of whether or not to utilise the pipeline registers in the proposed architecture depends on the number of clock cycles per DC, the value of τ for the supported PCMs, and the measured improvement to the maximum clock frequency, f_{max}. For the examples of SP2a and SP2b, the values of f_{max} before and after pipelining were 125.3 MHz and 154.7 MHz respectively, which equates to an improvement of $\frac{154.7}{125.3} = 1.235$. It may be seen in Table 5.1 that the PCMs of the WPAN family having rates R_1–R_3 may be rearranged to require no more than a single extra stall cycle. However, the high value of $D_C = N_B = 32$ in the highest-rate PCM results in this PCM having a requirement for 32 extra clock cycles per DC, which halves the effective throughput in the case where $Q = 1$, hence $t_{DC} = 32$.

The implications of this may be observed in Table 5.5, in which the increased clock frequency leads to higher processing throughputs in decoder S2b over S2a for rates R_1–R_3, which drops dramatically for rate R_4 due to the requirement for $t_i + \tau = 64$ clock cycles per DC. In this case, the decision of whether to use the optional pipelining registers may not be as simple as calculating the average processing throughput benefit, as it may depend on the estimated proportion of usage for each coding rate. However, Table 5.1 shows that the distribution of τ values for the other PCM families is more even, allowing this decision to normally be made according to (5.8) as described previously.

5.4.5 Decoders targeted at WiGig LDPC PCMs

Similarly to WPAN, the WiGig code family includes four PCMs, each of which have a single frame length of $n = 672$. However, unlike WPAN, the base PCMs $\mathbf{H}_b$ in this family all have $z = 42$, hence $n_b = 16$ block-columns. This permits a greater opportunity to vary the level of parallelism in the decoder parametrisations, according to the performance objectives of the resultant hardware. As before, both non-flexible and flexible decoders were synthesised, each with 'a' and 'b' variants having $\psi = 21$ and $\psi = 42$ VNPUs respectively.

TABLE 5.6: WiGig characterisation of the proposed stochastic architecture

Decoder	Q	Pipelined?	Active PCM n	Active PCM R	t_{DC} $(+\tau)$	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
SG1a	2	Yes	672	1/2	33	1	7.9	315	4.84	69.4	3.04
SG1b	1	Yes	672	1/2	17	1	11.5	315	8.62	39	3.04
SG2a	2	Yes	672	1/2	33	4	8	315	4.83	69.6	3.04
			672	5/8	40		8	256	6.12	68.6	3.17
			672	3/4	47		8	197	8.13	62	3.7
			672	13/16	48		8	184	9.23	59.1	4.32
SG2b	1	No	672	1/2	16	4	11.2	315	7.89	42.6	3.04
			672	5/8	16		11.2	256	12.13	34.6	3.17
			672	3/4	16		11.2	197	18.93	26.6	3.7
			672	13/16	16		11.2	184	21.96	24.9	4.32
[27]	-	-	672	1/2	-	1	71.4	10	475	4.24	3.02

By comparing the characteristics of the non-flexible and flexible decoders in Table 5.6 and Figure 5.12, it may be observed that the implementation cost of supporting multiple PCMs from the WiGig family compared to the single PCM having rate R_1 is negligible for the proposed architecture. This may be explained by the fact that the set of different VN degrees within the PCMs of the WiGig family stays constant for each of the four coding rates, whilst d_c changes. Since the datapath of the proposed architecture is only dependent on d_v not d_c, this leads to the property seen here. This is in contrast to the fixed-point decoder architecture characterised in Section 4.4.5, in which the hardware resource requirements depend on both d_v and d_c, causing it to require a greater quantity of hardware when supporting multiple PCMs from this family.

Note that for the case of the flexible decoder, the parametrisation SG2b having $Q = 1$ achieved higher processing throughputs without using the optional pipelining registers, whereas the increased clock frequency achievable by employing pipelining resulted in more favourable performance in SG2a where $Q = 2$. This relationship may also be observed for many of the decoder implementations discussed previously, where pipelining

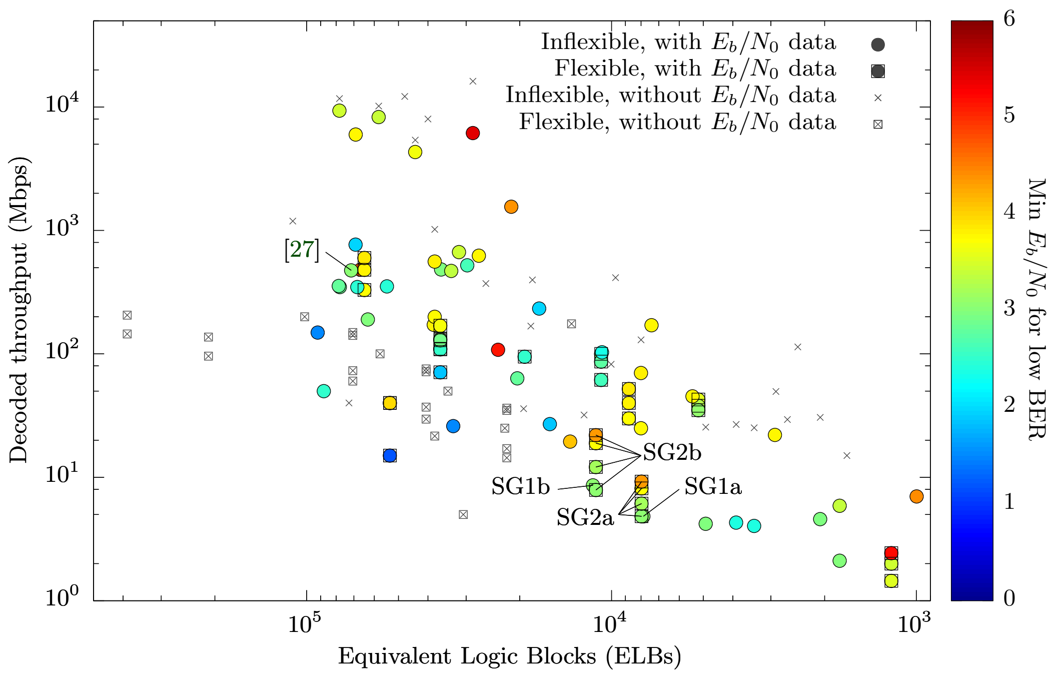


FIGURE 5.12: WiGig characterisation of the proposed stochastic architecture

is often only employed in decoders having values of Q greater than 1. This may be explained by the fact that the number of clock cycles per decoding iteration before pipelining t_{DC} is approximately proportional to Q, whereas the number of added clock cycles required by adding the pipeline registers τ is not related to Q. Accordingly, the processing throughput is affected proportionally less by the addition of τ clock cycles per DC when Q is larger, permitting the increased clock frequency f_{max} to have a larger impact.

5.4.6 Decoders targeted at PCMs from multiple families

Like the fixed-point architecture of Section 4.3, the proposed stochastic decoder architecture is also capable of flexibly supporting sets of LDPC codes from multiple families simultaneously. This key feature has been achieved due to the generalised design and construction of the architecture presented in Section 5.3, which does not depend on any of the specific characteristics of any individual code family. Furthermore, the offline design flow presented in Chapter 6 may once again be employed to generate the robust and optimised HDL for any such decoder parametrisation in seconds.

In order to facilitate comparisons with the parametrisations in Section 4.4.6, a similar set of decoder instances have been implemented here. However, the reduced hardware resource requirement of the proposed stochastic decoder architecture facilitates a higher degree of parallelism than was possible for the previous fixed-point implementations. Firstly, decoder SS1 targets the same four PCMs as FS1 (namely those from the four featured families having the lowest rates and frame lengths), but with a higher parallelism of $\psi = 42$. Accordingly, in this case the removal of the WiGig PCM having $z = 42$ from this set in decoder SS2 lowers the overall parallelism to $\psi = 27$. Similarly to decoder FS3, the stochastic decoder implementation SS3a supports two PCMs each from WiGig and WPAN, whilst SS3b supports all four of the PCMs in both of these families. Finally, decoder SS4 supports the complete set of 134 PCMs from across the four different families, with a parallelism of $\psi = 12$ VNPUs. Again, the characteristics of the fully-flexible benchmarkers from [25] and [26] are presented alongside the measured characteristics of the proposed implementations in Table 5.7[6] and Figure 5.13.

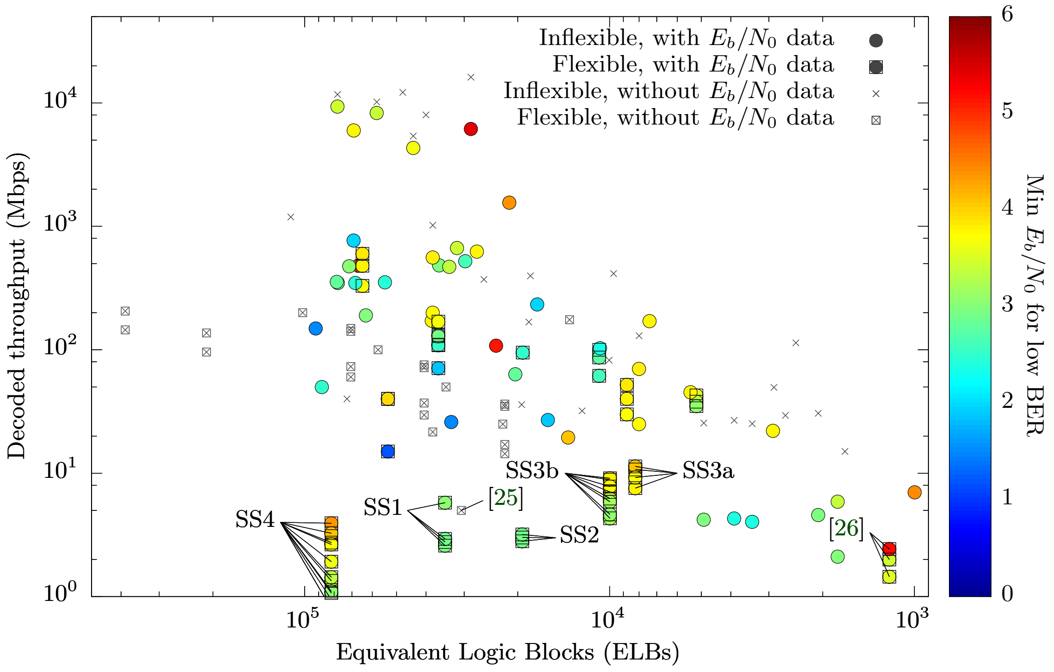


FIGURE 5.13: Multiple-family characterisation of the proposed stochastic architecture

As expected, Table 5.7 and Figure 5.13 show that the hardware resource requirements of SS2 are significantly lower than that of SS1, with slightly improved processing throughputs for the reduced set of supported PCMs. This throughput improvement may be explained by the increased clock frequency f_{max} that accompanies FPGA-based designs which utilise a smaller fraction of the logical resources and hence routing network. The cost of supporting an increased number of PCMs may once again be observed by comparing the characteristics of SS3a and SS3b, where SS3a supports half as many PCMs as SS3b, but does so with 18% fewer hardware resources and an increased mean processing throughput of 44.5%.

Finally, Table 5.7 and Figure 5.13 also show that decoder SS4 has far larger hardware resource requirements and lower processing throughput than many of the competing FPGA-based LDPC decoder designs surveyed in Chapter 3. This may be attributed to

[6]Note that in the fourth column of Table 5.7 the names of the code families featured here are abbreviated, where 'F' means 'Wi**F**i', 'M' means 'Wi**M**AX', 'P' means 'W**P**AN', and 'G' means 'Wi**G**ig'.

TABLE 5.7: Multiple-family characterisation of the proposed stochastic architecture

Decoder	Q	Pipelined?	PCM family	PCM n	PCM R	t_{DC} $(+\tau)$	Supported num. PCMs	ELBs (k)	I_a	T (Mbps)	Latency (μs)	E_b/N_0 (dB)
SS1	1	Y	F	648	1/2	28	4	34.6	411	2.59	125.1	2.91
			M	576	1/2	24		34.6	375	2.94	97.8	2.87
			G	672	1/2	17		34.6	315	5.77	58.2	3.04
			P	672	1/2	32		34.6	348	2.77	121.1	3.21
SS2	1	Y	F	648	1/2	28	3	19.3	411	2.81	115.3	2.91
			M	576	1/2	24		19.3	375	3.19	90.2	2.87
			P	672	1/2	32		19.3	348	3.01	111.6	3.21
SS3a	2	N	G	672	3/4	32	4	8.2	197	9.24	54.6	3.7
			G	672	13/16	32		8.2	184	10.72	50.9	4.32
			P	672	3/4	32		8.2	241	7.56	66.6	3.79
			P	672	7/8	32		8.2	187	11.37	51.7	4.08
SS3b	2	Y	G	672	1/2	33	8	10	315	4.63	72.6	3.04
			G	672	5/8	40		10	256	5.87	71.6	3.17
			G	672	3/4	47		10	197	7.79	64.7	3.7
			G	672	13/16	48		10	184	8.85	61.7	4.32
			P	672	1/2	32		10	348	4.31	77.9	3.21
			P	672	5/8	32		10	301	6.24	67.3	3.34
			P	672	3/4	33		10	241	9.09	55.5	3.79
			P	672	7/8	64		10	187	7.04	83.5	4.08
SS4	8	N	F	648	5/6	72	134	81.8	181	2.74	196.8	4.36
			F	1296	3/4	120		81.8	402	1.34	727.7	3.45
			F	1944	1/2	168		81.8	656	0.58	1663	2.46
			M	576	5/6	48		81.8	169	3.93	122.2	4.35
			M	864	3/4	72		81.8	310	1.92	337.2	3.62
			M	1248	2/3	120		81.8	427	1.07	774.1	3.13
			M	1920	1/2	168		81.8	632	0.6	1602	2.43
			M	2304	2/3	192		81.8	592	0.9	1715	2.88
			G	672	1/2	64		81.8	315	1.1	304.4	3.04
			G	672	3/4	64		81.8	197	2.65	190.4	3.7
			P	672	5/8	64		81.8	301	1.44	291	3.34
			P	672	7/8	64		81.8	187	3.26	180.5	4.08
[25]	-	-	-	NA	NA	NA	-	30.6	15	5.0	NA	NA
[26]	-	-	-	4095	0.82	NA	-	1.21	10	2.0	1679	3.42
		-	-	4095	0.82	NA			10	1.45	2316	3.52
		-	-	4095	0.94	NA			10	2.43	1584	5.19

the fact that it flexibly supports all of the QC PCMs mentioned previously, and hence must simultaneously be equipped to handle the most challenging facets of any one of these codes. More specifically, the large value of $D_V = 12$ from the lowest-rate WiFi PCMs result in very large VNPUs, a large number of which must be employed to limit the number of clock cycles required per DC. Furthermore, SS4 must also implement the large complex BSs described in Sections 4.4.3 and 5.4.3 in order to accommodate the large variation of submatrix sizes z required by the WiMAX code family.

It is for precisely these reasons that implementing FPGA-based LDPC decoders having a high degree of run-time flexibility poses such a challenge to the research community. The fact that inter-standard flexibility requires a decoder to be compliant with multiple problematic characteristics from varying code families simultaneously motivates research into architecture-aware LDPC PCMs, as exemplified by the current discussions of the 3GPP standardisation organisation [8]. The partially-parallel stochastic decoder supporting inter-standard flexibility presented here suffers from processing throughputs that are similar to those of the fully-serial decoders of [25]. However, fully-serial decoders exhibit low processing throughputs due to the extremely large number of clock cycles required per decoding iteration. Meanwhile, the partially-parallel stochastic architecture presented here requires a modest number of clock cycles per DC, and a large number of DCs per frame. Accordingly, the architecture presented here could now be improved to require fewer DCs per frame using several techniques posed in related research [91,94,95], as discussed in Section 7.2.

5.4.7 Comparison of the proposed architectures

An additional analysis of the proposed architectures is presented in Figure 5.14, which compares the characteristics of five previously-discussed decoder implementations having the proposed fixed-point architecture of Chapter 4, and five decoder implementations having the proposed stochastic architecture from this section. Each of the plotted decoder instances support all of the PCMs in one of the featured families, in combination with an implementation of each architecture that supports all 134 of the PCMs discussed previously. As with Figure 4.10, characteristics are plotted radially on logarithmic scales, with more favourable values for each characteristic extending outwards. Since each decoder supports multiple PCMs, and its characteristics are dependent on the currently active PCM, average values across the entire supported PCM set of each decoder are plotted for throughput, latency, and error correction measurements. The values are then plotted proportionately to the average of each characteristic, across the set of 10 decoder instances.

Figure 5.14 graphically exemplifies many of the trade-offs that have been discussed throughout the previous discussions. For example, clearly the stochastic decoders all suffer from a lower processing throughput and higher latency than their fixed-point counterparts, but also achieve lower hardware resource requirements. This trade-off is particularly evident when comparing the decoders targeting the WPAN and WiGig code families (i.e. FP2, FG2b, SP2b, SG2a), which have the highest throughputs and lowest hardware requirements of the set, for the fixed-point and stochastic variants, respectively. Furthermore, the methods described in Sections 4.4.1 and 5.4.1 to measure the

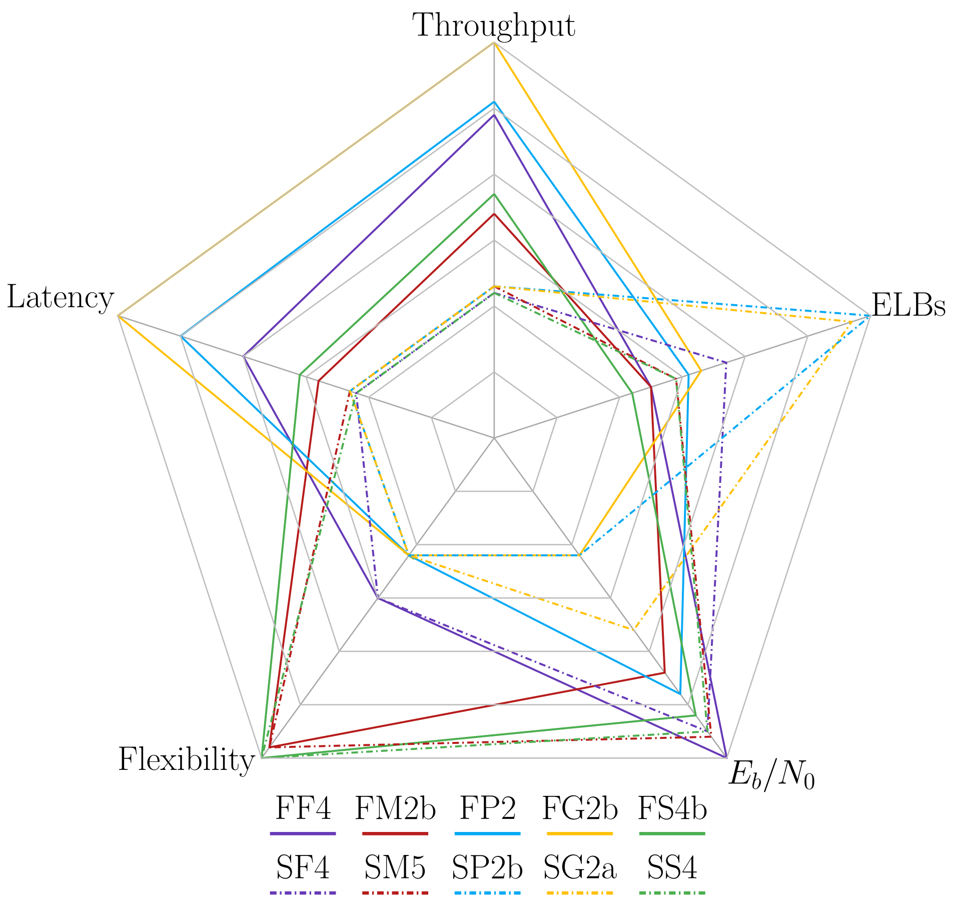


FIGURE 5.14: Comparison of decoders having the proposed architectures. Characteristics are normalised relative to the mean value of each property.

characteristics of each decoder instance have resulted in the stochastic decoders having approximately equivalent error correction performance to their fixed-point counterparts. This is a characteristic which is greatly affected by the number of decoding iterations or DCs, which subsequently linearly affects the decoding throughput, producing the variation of values shown here.

It may also be observed in Figure 5.14 that the ranking of the proposed decoders from best to worst in terms of hardware resource requirements is almost exactly the inverse of the ranking in terms of the degree of run-time flexibility. The one notable exception to this is decoder FF4, which suffers from approximately the same hardware resource requirement as decoder FM2b, which supports almost 10 times more PCMs. However, this may be explained again by the large node degrees in the WiFi code family, which subsequently lead to this decoder's superior error correction performance.

To an extent, the partially-parallel stochastic decoder architecture of Section 5.3 succeeds at reducing the hardware requirements compared to the fixed-point architecture of Section 4.3. This is primarily achieved by the use of single-bit messages throughout the stochastic decoders, rather than $W = 4$-bit words. It is notable, then, that the hardware resource requirements of the stochastic decoders are much less than 4 times smaller than their fixed-point equivalents. This may be explained by a number of factors, including the increased complexity of the stochastic VNPU and the requirements for a large number of random number generators. Additionally, this fact may also be attributed to the physical composition of an FPGA. More specifically, the structure of an FPGA is dominated by multiple-input single-output logical elements, which can be programmed so that the output bit is any function of its inputs. These elements are ideally suited for performing complex calculations, but may be under-utilised when performing simpler functions, such as a single-bit multiplexer. Accordingly, the highly flexible routing requirements of flexible FPGA-based LDPC decoders result in a high utilisation of these logic blocks, which does not decrease linearly with the message width. It may therefore be argued that this issue would be less significant in ASIC implementations of the proposed decoder architectures, where higher hardware utilisations are possible.

5.5 Conclusion

In this chapter, we have presented the design and implementation of a novel stochastic FPGA-based LDPC decoder architecture, which represents the first known implementation of a stochastic LDPC decoder employing a partially-parallel architecture. This was achieved using the modified stochastic VNPU subnode design described in Section 5.3.3.3, in which multiple independent ring buffers (which may be implemented using an FPGA's BRAM) are employed to enable the VNPU to function as several different time-multiplexed VNs, over the course of each DC. Additionally, the flexible dual-tree NPU architecture proposed in Section 5.3.3.2 results in VNPUs which are capable of adapting to variations in the VN degree d_v, which is another requirement for partially-parallel decoders capable of supporting irregular PCMs. This general NPU architecture combines a requirement for a minimal number of subnodes with a short critical path length, whilst also being able to flexibly process varying numbers of inputs and outputs at run-time.

Furthermore, the proposed stochastic decoder architecture is also the first to implement a layered decoding schedule. This schedule is based on the stochastic variation of traditional layered belief propagation, presented in Section 5.2.2. Adopting this schedule significantly reduces the quantity of hardware resources required by the datapath, at the cost of slightly increased storage requirements in the VND, as described in Section 5.3.3.1. Additionally, the discussions of Section 5.3.1.2 present a method in which the natural incompatibility of this schedule with pipelining in the stochastic case may be mitigated in some situations.

Finally, an additional aspect of the proposed architecture's novelty is its high degree of run-time flexibility, which is another first for stochastic decoders. Once again, this level of flexibility is such that decoders having the proposed architecture may be rapidly designed to be compatible with any set of one or more QC PCMs, and subsequently have the capability to switch between them at run-time, in a single clock cycle. This flexibility is achieved by adopting many of the same elements presented in Chapter 4 for the corresponding fixed-point decoder architecture, such as the overall parametrisation method, the flexible datapath structure, and the programmable BSs. Additionally, Section 5.3.2 also describes a further optimisation which uses the characteristics of the supported PCM set to reduce the hardware requirements of the flexible CND selection multiplexers and de-multiplexers in the datapath.

The implementation results presented in Section 5.4 indicate that the proposed stochastic architecture successfully reduces the hardware resource requirements in comparison to the fixed-point architecture. This is achieved whilst maintaining an equivalent error correction performance and high degree of run-time flexibility, but at the cost of significantly lower processing throughputs. These may be attributed to the combination of the large number of clock cycles required for each DC, and the large number of DCs required to successfully correct a received frame. It is therefore recommended in Section 7.2 that further research is performed to find ways of increasing this throughput by reducing the number of DCs required per frame. Alternatively, it would also be desirable to further decrease the hardware requirements of the NPUs and the flexible routing, which would consequently facilitate a higher degree of parallelism and hence fewer clock cycles per DC.

Chapter 6

The proposed offline design flow

6.1 Introduction

The findings of Chapter 3 demonstrate the complexity of the interactions between the numerous parameters that must be selected for an FPGA-based LDPC decoder design, and also with the characteristics that are manifested for the resultant implementation. These interactions complicate the process of designing an FPGA-based LDPC decoder, which is exacerbated when that decoder must be designed to flexibly support multiple Parity-Check Matrices (PCMs) at run-time. Meanwhile, Chapter 4 and Chapter 5 presented fixed-point and stochastic FPGA-based LDPC decoder architectures that support flexibility over any set of one or more Quasi-Cyclic (QC) LDPC PCMs. The chosen PCMs may originate from one family or several different families, and may vary in any of their parameters, including the frame length n, the QC expansion factor z, the coding rate R, and the row/column degrees d_c and d_v. These decoder architectures are described in a generalised form which is not specific to any one PCM, allowing much of this design process to be automated. The algorithms and software capable of doing so may be considered as an offline design flow, which offers automated design-time flexibility. This automated design-time flexibility allows the designer to specify which PCMs should be supported, without requiring in-depth knowledge of the architectural decisions involved. Subsequently, run-time flexibility then facilitates switching between these supported PCMs at run-time.

This chapter details the proposed design flow, which facilitates this automated design-time flexibility for the proposed LDPC decoder architectures. More specifically, this

This chapter is based on the following publications:

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A Flexible FPGA-Based Quasi-Cyclic LDPC Decoder," in *IEEE Access*, vol. 5, pp. 20965–20984, mar 2017.

P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Hardware-efficient Node Processing Unit Architectures for Flexible LDPC Decoder Implementations," in *IEEE Trans. Circuits Syst. II, Exp. Briefs.*

design flow automatically generates an optimised LDPC decoder design for a chosen set of one or more QC LDPC codes, having one of the architectures described previously. In order to do so, the design flow extracts the key parameters and values from the set of PCMs, generates optimised submodule structures where necessary, and automatically produces a robust Hardware Description Language (HDL) description from which to synthesise hardware. A flowchart depicting this design flow is presented in Figure 6.1, where processes are represented as orange rectangles and process outputs are represented as blue parallelograms. The text in bold next to each process describes the executor of that process, which may be a piece of software or the human user.

Note that this design flow represents a different manner of automated HDL generation from conventional High-Level Synthesis (HLS) tools [74, 143, 144]. More specifically, HLS tools have been proposed to allow the functionality of a system to be written in a high-level language such as C++, which is then parsed, interpreted, and translated into HDL for synthesis. This design flow hence permits software engineers (having extensive programming capabilities but little to no knowledge of architectural concerns) to implement sophisticated algorithms in programmable hardware. Indeed, a recent survey [145] has demonstrated that non-flexible FPGA-based LDPC decoders generated using HLS may perform similarly well to those described using HDL directly, by comparing their performance to the data published from Chapter 3. Conversely, the design flow presented in this chapter may be used to generate specific optimised HDL descriptions of the novel flexible LDPC decoders presented earlier, hence without requiring any further cost to those already discussed in Sections 4.4 and 5.4. This may be performed without requiring any prior high-level programming expertise or hardware design experience.

This chapter is structured as follows. Section 6.2 presents the function of a custom C++ application that extracts the key parameters and values from the set of chosen QC PCMs and determines the decoder parameters presented in Figure 6.1. Following this, Section 6.3 briefly describes the process of automatically generating the HDL description of the specified decoder. Finally, Section 6.4 details the algorithmic methods used to specify the structures of the decoder modules having the greatest dependence on the PCM set, namely the Node Processing Units (NPUs) described in Sections 4.3.4 and 5.3.3.2, as well as the programmable Barrel Shifters (BSs) described in Section 4.3.3.3.

6.2 PCM interpretation

Before the HDL of the proposed design can be generated, a small amount of offline computation is required in order 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.

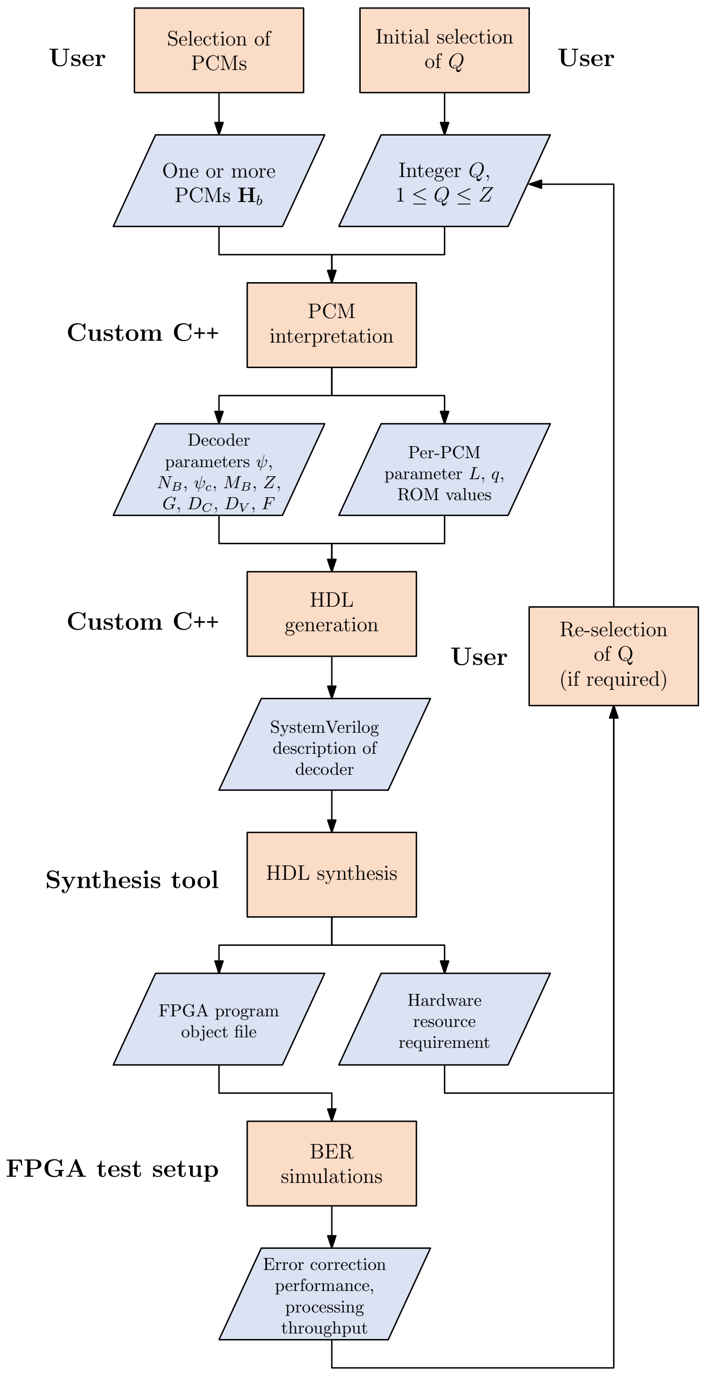

User
Selection of PCMs
Initial selection of Q
User
One or more PCMs $\mathbf{H}_b$
Integer Q, $1 \leq Q \leq Z$
Custom C++
PCM interpretation
Decoder parameters ψ, N_B, ψ_c, M_B, Z, G, D_C, D_V, F
Per-PCM parameter L, q, ROM values
Custom C++
HDL generation
User
Re-selection of Q (if required)
SystemVerilog description of decoder
Synthesis tool
HDL synthesis
FPGA program object file
Hardware resource requirement
FPGA test setup
BER simulations
Error correction performance, processing throughput

FIGURE 6.1: The design and implementation stages involved in the generation of an FPGA-based LDPC decoder having run-time flexibility

6.2.1 Calculating parameters

The first role of the PCM interpretation software is to calculate the decoder's parameters based on the parameters of the user-specified PCM set. The key parameters of the architectures were all introduced at the relevant points within the architecture descriptions in Sections 4.3 and 5.3, but are summarised together here.

Both architectures characterise the selected set of supported PCMs by finding the maxima of the matrix dimensions (n_b, m_b, z) and degrees (d_v, d_c), namely N_B, M_B, Z, D_V, and D_C, respectively. From these, the overall degree of parallelism ψ may be calculated according to (4.3) in Section 4.3.1.1, based on Z and the user-input parallelism reduction factor Q. The value of Q facilitates a trade-off between the decoder's processing throughput and hardware resource requirements, where a higher Q results in fewer NPUs but also increases the number of clock cycles required to complete each decoding iteration or Decoding Cycle (DC). Accordingly, a value of q may then be calculated for each PCM according to (4.4). As described in Section 4.3.1.1, q represents the effective value of Q for PCMs having a value of $z < Z$, for which the value of ψ may be high enough that fewer than Q clock cycles are required per block-row or block-column.

The fixed-point architecture presented in Section 4.3 also incorporates several further optimisations to reduce its hardware resource requirements, which necessitate the extraction of the following additional parameters from the PCM set. In this architecture, ψ represents the number of Variable Node Processing Units (VNPUs) implemented within the Variable Node Decoder (VND), whilst $\psi_c \leq \psi$ represents the number of Check Node Processing Units (CNPUs) implemented within the Check Node Decoder (CND). The value of ψ_c is calculated according to (4.6) based on ψ and G_{min}. Here, G_{min} is the minimum value of G within the PCM set, where G is calculated according to (4.5). As introduced in Section 4.3.3.2, the boolean parameter F indicates whether or not the decoder for the chosen PCM set would benefit from employing the flexible CNPUs of Section 4.3.4.3. More specifically, F adopts a value of 1 if any PCM p in the PCM set has $L_p = 1$, where L_p is calculated according to (4.7).

Note that the PCM interpretation software also performs further parsing of the supported PCM set in order to derive the parameters used in the construction of the NPUs and BSs. The natures of these values are discussed along with their use in the relevant descriptions in Section 6.4.

6.2.2 PCM ROMs

The second task of the PCM interpretation software is to extract the locations and shift values of the non-null submatrices in each QC PCM $\mathbf{H}_b$, and arrange them in a format compatible with the ROMs introduced in Sections 4.3.1.3 and 5.3.1.3. The details of these ROMs differ slightly between the two architectures, so are described separately here.

Two sets of ROMs are required for the fixed-point architecture of Section 4.3: one which views each PCM as n_b columns of d_v values, for the VND, and one which views each PCM as m_b rows of d_c values, used by the CND. Each location within each ROM in the set used by the VND contains information regarding the non-null submatrices in each block-column of the corresponding PCM. Therefore, all of the VND ROMs contain N_B locations, each with D_V values. Similarly, the ROMs utilised by the CND each contain M_B locations, each with D_C values. Within each set, this architecture requires three ROMs, namely SELECT, SHIFT, and NONZERO, which store the position, value, and presence of each non-null submatrix of $\mathbf{H}_b$, as described in Section 4.3.1.3. However, the mapping of the PCM submatrix values into these ROMs must reflect the manner in which the extrinsic Logarithmic-Likelihood Ratios (LLRs) associated with each submatrix are stored within the VMEM and CMEM, as described in Section 4.3.2. More specifically, LLRs are stored in an offset manner, in order to avoid clashes caused by simultaneous row-centric and column-centric operations on the inter-node message memories. To facilitate this, each row r_j, where $j \in [1, m_b]$, in each $\mathbf{H}_b$ must be cyclic-shifted to the right by $j - 1$ places, before being written into ROM.

Conversely, the single-datapath structure of the stochastic architecture of Section 5.3 requires only one set of ROMs. More specifically, as described in Section 5.3.1, this architecture only implements VNPUs, so only requires the column-centric ROM representations discussed previously. Additionally, this property avoids the requirement for the offset memory access mentioned previously. This architecture therefore requires a single set of three ROMs, namely SELECT, SHIFT, and DEGREE. Identically to its counterpart in the fixed-point architecture of Section 4.3, the SELECT ROM stores the block-row indices of the non-null submatrices in each block-column of $\mathbf{H}_b$. Conversely, for each non-zero submatrix, the SHIFT ROM stores the difference (modulo-z) between the current shift value s and the previous shift value of the most recent non-null submatrix in the same block-row. Doing so facilitates the incremental shifting operation explained in Section 5.3.1.1. Finally, the DEGREE ROM stores the value of d_v for each block-column in $\mathbf{H}_b$, which is used in the flexible dual-tree VNPUs as described in Section 5.3.3.2, and to ensure that only updated messages are written to the CND memory, as described in Section 5.3.2.

6.3 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 either of the proposed architectures, using the parameters and values calculated in the previous stage. In addition to automatically generating robust code with the appropriate parameters, this approach permits the optional inclusion of certain elements which may not be required in all applications. For example, in the stochastic architecture, the additional user-input boolean **pipe** controls whether or not the architecture is implemented having the optional pipeline registers described in Section 5.3.1.2.

Furthermore, these optional elements also include the PCM input signal (which is not required when the decoder is designed to only support a single PCM), the flexible CNPUs in the fixed-point architecture of Section 4.3 (which are not required when the decoder flexibility parameter adopts the value $F = 0$), and the selector module in the stochastic architecture of Section 5.3 (which is not required when the parallelism reduction factor adopts the value $Q = 1$, giving $\psi = Z$). 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.

In addition, having the code written by a high-level library creates possibilities for adding extra functionality in the future. An example of this could be a graphical user interface to allow a user to prioritise one or more of the characteristics affected by the trade-offs discussed in Chapter 3. For example, a suggested value of Q could be implemented, which would be high when the user opted to prioritise low hardware resource requirements, and lower if they opted to prioritise high processing throughput.

6.4 Module descriptions

This section provides the specific details of those modules described throughout Sections 4.3 and 5.3 having structures that depend on the supported PCM set. Specifically, this includes the general tree structures of the NPUs described in Section 4.3.4, the dual-tree NPU described in Section 5.3.3.2, and the programmable BS described in Section 4.3.3.3. The design-time generation of these modules is discussed in the following sections.

6.4.1 Binary tree NPU generation

It was observed in Section 4.3.4 that fully specifying the desired tree structures within the NPUs 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. This process is illustrated in Figure 6.2 for the summation of 4, 5, and 6 elements, referred to as a to f. In this manner, the internal structure of the Tree SM VNPUs architecture of Section 4.3.4.1 can be programmatically defined by setting $y = D_V + 1$, yielding the result depicted in Figure 4.6 for $D_V = 12$.

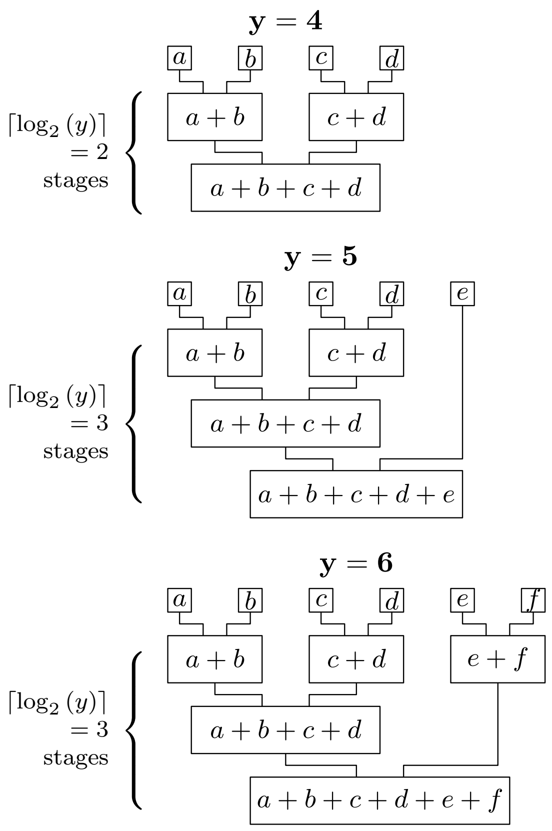


FIGURE 6.2: Examples of the tree generation algorithm for 4, 5, and 6 inputs

This structure may also be used repeatedly in the Min Tree CNPUs of Section 4.3.4.2, where the min function is used instead of additions. However, rather than creating one 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 [57]. For example, when $D_C = 4$, the minimum of inputs a and b may be used in the trees for both outputs c and d, which are obtained as $\min(a, b, d)$ and $\min(a, b, c)$, respectively.

This technique may hence be used in the proposed offline design flow 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.

6.4.2 Dual-tree NPU generation

An NPU having the novel generalised flexible dual-tree architecture proposed in Section 5.3.3.2, with Λ inputs and outputs, may be constructed using the algorithm presented in Figure 6.3. Recall from Section 5.3.3.2 that the proposed architecture is comprised of a *summing* section followed by a *combining* section, each of which contains $Y = \lceil \log_2(\Lambda) \rceil - 1$ stages y of intermediate signals x, denoted as $S^{y,x}$. Furthermore, the proposed architecture supports run-time flexibility through the use of *bypass multiplexers* at the outputs of certain subnodes, to facilitate the optional bypassing of that subnode output when one of its inputs originates from an unused NPU input. We employ the notation λ_a to denote the current number of active inputs and outputs, and $\lambda_n^{y,x}$ to denote the minimum value of λ_a for which a subnode is required at term $S^{y,x}$. Accordingly, when $\lambda_a < \lambda_n^{y,x}$ at a term $S^{y,x}$ having a bypass multiplexer, that multiplexer is used to bypass the subnode's output.

Furthermore, motivated by the observation that full flexibility over all possible node degrees is not required for most applications, it may be observed that the algorithm of Figure 6.3 can optimise the proposed architecture to only include the bypass multiplexers that are actually used, when a limited set of NPU degrees is required. For example, consider the signal $S^{1,7}$ at the output of the bottom-left subnode of Figure 5.6(a). According to the operation of the dual-tree architecture described in Section 5.3.3.2, when $\lambda_a = 16$, this signal is the combination of inputs in_{14} and in_{15}, whereas when $\lambda_a < 16$, this signal connects directly to in_{14} through the corresponding bypass multiplexer. Subsequently, the next connected signal, $S^{2,3}$, takes the combination of signals $S^{1,6}$ and $S^{1,7}$ when $\lambda_a \in \{15, 16\}$, or $S^{1,6}$ when $\lambda_a < 15$. Now, consider the case in which $\lambda_a = 15$ is known to never occur, because none of the supported PCMs contain any Variable Nodes (VNs) having a degree of $d_v = \lambda_a - 1 = 14$. Accordingly, the value obtained at $S^{1,7}$ will only ever be required[2] in cases where $\lambda_a = 16$, when it will use the subnode result. Hence, the bypass multiplexer at $S^{1,7}$ will never be utilised, and can be removed.

The set of multiplexers that can be removed in this way is identified using the *connection* parameter $\lambda_c^{y,x}$ of Figure 6.3, along with the set of numbers of inputs and outputs that are supported by the NPU, which we denote as $\boldsymbol{\lambda}_s$. For the multiplexer at signal $S^{y,x}$, the value of $\lambda_c^{y,x}$ represents the minimum value of λ_a for which the bypassed result provided by that multiplexer (i.e. when $\lambda_a < \lambda_n^{y,x}$) will be utilised in later stages. Accordingly, if the set $\boldsymbol{\lambda}_s$ does not include any values of λ for which $\lambda_c^{y,x} \leq \lambda < \lambda_n^{y,x}$, then the bypassed result at $S^{y,x}$ will never be used, and hence the corresponding multiplexer is not required. In this case, the signal that would otherwise be provided by the multiplexer can be connected directly to the output of the corresponding subnode. The value of $\lambda_c^{y,x}$ is calculated with respect to the value of $\lambda_n^{y,x}$ from (5.7), according to

$$\lambda_c^{y,x} = \begin{cases} \lambda_n^{y,x} - 2^{y-1}, & \textit{summing section} \\ \lambda_n^{y,x} - 2^{2Y-y}, & \textit{combining section} \end{cases}. \tag{6.1}$$

This may be observed in Figure 5.6(b), which depicts a VNPU that can provide the functionality of any variable node within the 12 PCMs of the IEEE 802.11n LDPC code [3]. More specifically, this $\Lambda = 13$ VNPU supports any number of inputs and

[2]Note that in this example, $S^{1,7}$ is also employed as an input to the subnode at $S^{5,6}$. However, due to the symmetry of the dual-tree architecture, it is also bypassed there with the same constraints.

FIGURE 6.3: Construction of a flexible dual-tree NPU

```
procedure CONSTRUCT_NPU($\Lambda$,  $\boldsymbol{\lambda}_s$)
    $Y := \lceil \log_2(\Lambda) \rceil - 1$        ▷ no. of stages per section
    $S^0 :=$ inputs        ▷ $S^{y,x}$ = signal $x$ at stage $y$
Summing section
    for $y = 1 : Y$ do
        for $x = 0 : \lceil \frac{\Lambda}{2^y} \rceil$ do
            $L := S^{y-1,2x}$        ▷ first connected signal
            if $(x = 0)$ then $\lambda_n := 2^y + 1$
            else $\lambda_n := x \times 2^y + 2^{y-1} + 1$
            if $(\Lambda \geq \lambda_n)$ then
                $R := S^{y-1,2x+1}$        ▷ second connected signal
                $t :=$ SUBNODE($L$,  $R$)
                $\lambda_c := \lambda_n - 2^{y-1}$
                if $(x \geq 1)$ and
                 (any $\lambda \in \boldsymbol{\lambda}_s$, $\lambda_c \leq \lambda < \lambda_n$ exists) then
                    $S^{y,x} :=$ MUX($(\lambda_a \geq \lambda_n)$?  $t \mid L$)
                else
                    $S^{y,x} := t$        ▷ no mux after subnode
                end if
            else
                $S^{y,x} := L$        ▷ no subnode
            end if
        end for
    end for
Combining section
    for $y = Y + 1 : 2Y$ do
        for $x = 0 : \lceil \frac{\Lambda}{2^{2Y-y}} \rceil$ do
            if $(y > Y + 1)$ then $L := S^{y-1,\lfloor \frac{x}{2} \rfloor}$
            else $L := S^{y-1,1+\lfloor \frac{-x}{2} \rfloor}$        ▷ First stage flips inputs
            if $(x \geq 2)$ then $\lambda_n := 2^y \times (2 \lfloor \frac{x}{2} \rfloor + 1)$
            else $\lambda_n := 2 \times 2^y \times (2 \lfloor \frac{x}{2} \rfloor + 1)$
            if $(\Lambda \geq \lambda_n)$ then
                $R := S^{2Y-y,x+(-1)^x}$
                $t :=$ SUBNODE($L$,  $R$)
                $\lambda_c := \lambda_n - 2^{2Y-y}$
                if (($x$ mod $2 = 0$ or $x = 1$) and
                 (any $\lambda \in \boldsymbol{\lambda}_s$, $\lambda_c \leq \lambda < \lambda_n$ exists)) then
                    if $x \leq 1$ then $S^{y,x} :=$ MUX($(\lambda_a \geq \lambda_n)$?  $t \mid R$)
                    else $S^{y,x} :=$ MUX($(\lambda_a \geq \lambda_n)$?  $t \mid L$)
                else
                    $S^{y,x} := t$        ▷ no mux after subnode
                end if
            else
                $S^{y,x} := L$        ▷ no subnode
            end if
        end for
    end for
    outputs $:= S^{2Y}$
end procedure
```

outputs in the set $\boldsymbol{\lambda}_s \in \{3, 4, 5, 7, 8, 9, 12, 13\}$. This set permits the removal of the bypass multiplexers at signals $S^{1,5}$, $S^{6,10}$, $S^{6,1}$, and $S^{6,0}$ (where the output terms $S^{6,x}$ at the final stage $y = 2Y = 6$ are denoted as out_x in Figure 5.6). This may be verified by calculating the values of $\lambda_n^{y,x}$ and $\lambda_c^{y,x}$ for these signals, which equate to 11 and 12 respectively for both $S^{1,5}$ and $S^{6,10}$, and 2 and 3 respectively, for both $S^{6,1}$ and $S^{6,0}$.

6.4.2.1 Flexible stochastic subnode generation

Additionally, it was described in Section 5.3.3.3 that the 2-input 1-output subnodes employed by the dual-tree VNPUs in the proposed stochastic decoder architecture are flexible Edge Memories (EMs) and Internal Memories (IMs), each of which maintain multiple independent ring buffers. Having these multiple ring buffers is necessary due to the fact that each VNPU represents multiple different VNs within the factor graph at different times. However, as has been shown previously, not every subnode is required in every VNPU activation, when the number of inputs and outputs λ varies. In this case, the total number of times each subnode is activated in any one DC can be calculated according to the minimum value of λ for which that subnode is not bypassed, λ_n. More specifically, the number of ring buffers required by the subnode at signal $S^{y,x}$ is equal to the maximum number of VNs in any of the supported PCMs having $d_v + 1 \geq \lambda_n^{y,x}$, divided by the number of parallel VNPUs ψ. These values may be calculated during the PCM interpretation stage described in Section 6.2.1 and included in the generated HDL code. For the example case of a decoder capable of supporting all 12 PCMs in IEEE 802.11n [3], where each of the $\psi = 27$ VNPUs contains $N_{sub} = 33$ subnodes, this optimisation reduces the required number of ring buffers per VNPU from 2376 to 891, which represents a 62.5% reduction.

6.4.3 Programmable barrel shifter generation

As described in Section 4.3.3.3, the design of flexible BSs capable of supporting multiple submatrix sizes can result in an excessive hardware resource requirement unless the design is carefully optimised. Accordingly, the HDL for the BSs employed in both of the proposed flexible decoder architectures is automatically tailored for the specific set of supported PCMs at design-time. During the PCM interpretation stage described in Section 6.2, the design flow software assembles a vector containing each unique value of z within the supported PCM set in ascending order. During the HDL generation stage, as mentioned previously in Section 4.3.3.3, each BS output B_e is selected from a u-to-1 multiplexer, where u is the number of supported z values greater than e [138]. The multiplexer input that is used for each PCM is determined by the value of z for that PCM, as described in [138].

6.5 Conclusion

The automated design flow presented in this chapter adds an additional element of automated design-time flexibility to the FPGA-based LDPC decoder architectures presented in Chapters 4 and 5. This is facilitated by the presentation of the proposed architectures in a fully generalised manner, such that tailoring them to any arbitrary set of QC PCMs may be performed entirely algorithmically. Doing so involves parsing the selected set of QC PCMs to extract the key decoder parameters described in Section 6.2.1, and translating the PCMs into a hardware-optimised form that is appropriate for the selected architecture variant. Following this, the robust and human-readable HDL description of the desired decoder is automatically generated by a custom C++ HDL generation library, which may be synthesised directly onto an FPGA. In this chapter, we have also provided details of the algorithmic approaches used to generate some of the more complex elements of the proposed architectures. These components represent the elements of the synthesised decoders that dominate the overall measured implementation characteristics, which motivates the careful optimisations described here.

The efficacy of this design flow is demonstrated in the discussions of the implementation characteristics of the proposed architectures in Sections 4.4 and 5.4. Here, multiple variations of each decoder were rapidly generated, synthesised, and characterised by utilising the automated design-time flexibility described in this chapter. Without this, manually adapting the HDL for each different set of supported PCMs or degree of parallelism would have been a laborious and time-consuming task, and the wealth of results available in these previous chapters would have been reduced.

Chapter 7

Conclusions and future work

In this thesis, we have made several significant contributions to the field of research into flexible FPGA-based LDPC decoder design and implementation. In doing so, we have also identified several opportunities for further research in this area, in order to further improve or complement the contributions made by this thesis. These are summarised in Sections 7.1 and 7.2 respectively, before Section 7.3 offers some concluding remarks.

7.1 Summary and conclusions

In the first half of this thesis, we investigated the multitude of factors involved in the design and implementation of FPGA-based LDPC decoders, whilst also surveying the current state of the art.

Firstly, in Chapter 2 we presented a tutorial-style introduction to all of the related theoretical and implementational concepts, establishing a foundation of knowledge and terminology upon which to build the forthcoming discussions. The basics of channel codes were reviewed, with a particular focus on the generation and operation of LDPC codes. The Parity-Check Matrix (PCM) and its factor graph representation were introduced, along with a description of the ways in which the structures and characteristics of these two related elements define the fundamental LDPC decoding process. This decoding process was then explored in more detail, including a discussion of the various decoding schedules which may be adopted, as well as variations of the equations performed by the Check Nodes (CNs) and Variable Nodes (VNs). The hardware-specific factors of LDPC decoder design were then discussed, followed by a review and discussion of the structure of FPGAs. Finally, Chapter 2 concluded with an introduction and explanation of the principles of stochastic decoding, which offers an alternative to fixed-point multiple-bit words for the inter-node messages transmitted throughout an LDPC decoder.

Following this, in Chapter 3 we enumerated the complete set of design-time parameters and resultant run-time characteristics involved in the design, implementation, and analysis of FPGA-based LDPC decoders. We then presented the results of an exhaustive survey of over 140 published FPGA-based LDPC decoder designs, from both academic research and industrial IP. By utilising the metrics and methods described in this chapter, the parameters and characteristics of these decoder designs were compared, discussed, characterised, and illustrated. Doing so lead to an intricate analysis of the manners in which these parameters and characteristics affect each other, and the fundamental trade-off that exists between hardware resource requirements, processing throughput, transmission energy efficiency, and run-time flexibility. From this, we were able to provide a recommended methodology to aid the design of future decoders, as well as a series of recommendations for future publications to follow, which will facilitate more in-depth comparisons of designs in the future.

The results presented in Chapter 3 also indicated a gap in the state of the art in FPGA-based LDPC decoder designs, namely for architectures having run-time flexibility. Accordingly, in the second half of this thesis, we utilised the knowledge gained during the first half to propose two novel highly-flexible FPGA-based LDPC decoders.

Firstly, in Chapter 4 we presented our proposed flexible fixed-point FPGA-based LDPC decoder architecture, which is capable of supporting any set of one or more Quasi-Cyclic (QC) PCMs. The degree of run-time flexibility of this architecture is such that switching the PCM that is being used to decode a specific codeword can be performed in a single clock cycle. Various facets of the architecture were described, with a particular focus on those elements that facilitate the run-time flexibility. Additionally, various optimisations were proposed to reduce the hardware resource requirements of the proposed architecture when the set of supported PCMs has a wide variation in certain parameters, such as the numbers of block-rows m_b and the check node degrees d_c. Multiple parametrisations of this architecture were then synthesised and characterised, demonstrating its capability to flexibly support any combination of the PCMs in several state-of-the-art wireless communications standards. These results indicate that the proposed fixed-point architecture achieves its high degree of parallelism without decreasing its error correction performance, and is also capable of achieving processing throughputs that are higher than any previous FPGA-based LDPC decoder having run-time flexibility. However, this comes at the cost of a generally higher hardware resource requirement, which is mainly due to the requirement for a large number of highly complex flexible routing components to connect the multi-bit messages between the Variable Node Processing Units (VNPUs) and Check Node Processing Units (CNPUs).

Motivated by this, in Chapter 5 we presented an alternative FPGA-based LDPC decoder architecture which employs stochastic probability representations rather than fixed-point Logarithmic-Likelihood Ratios (LLRs). This is the first stochastic LDPC decoder design to adopt a partially-parallel architecture, to utilise a layered decoding schedule, and to support run-time flexibility over multiple PCMs. This chapter detailed the layered decoding schedule employed by this architecture, which is a novel column-centric variation on the standard row-centric layered belief propagation schedule employed by many previous partially-parallel decoders. The high level of flexibility was achieved by adapting the structure of the proposed fixed-point architecture and re-using several of its features, albeit with the inclusion of some modifications to suit stochastic processing. Meanwhile, the methods required to modify the processing and routing elements of a stochastic decoder to make it suited to a partially-parallel architecture were also described. Most importantly, these include modifications to the VNPU subnodes to permit them to represent multiple different VNs at different times over the course of each Decoding Cycle (DC). Furthermore, this chapter also contains a description of a proposed Node Processing Unit (NPU) architecture, referred to as a dual-tree, which is also required for partially-parallel stochastic architectures to support PCMs having irregular node degrees. This generalised NPU architecture employs a minimal number of subnodes and exhibits a short critical path length, whilst also facilitating changes in the numbers of inputs and outputs at run-time without requiring the existence of a "null" input value. Similarly to the previous chapter, implementation results for decoders having the proposed stochastic architecture supporting a wide variety of PCM sets were used to characterise its performance. These showed that the large number of clock cycles required per frame in this architecture result in particularly low processing throughputs in many cases. However, otherwise the proposed architecture facilitates high flexibility and reasonable error correction performance, with hardware resource requirements that are generally lower than other comparable decoders.

It was stated throughout Chapters 4 and 5 that the proposed architectures were described in a generalised form, with every aspect dependent on the set of supported PCMs and target degree of parallelism. This motivates the offline design flow presented in Chapter 6, which automates the generation of optimised decoder designs employing the proposed architectures. This design flow comprises custom software which selects desirable values for the decoder parameters based on a user-selected set of QC PCMs, and outputs the full set of Hardware Description Language (HDL) files required to synthesise the design. This chapter also presented specific details of the design-time generation of some of the more heavily-optimised modules within the proposed architectures, namely the tree-based NPUs, the dual-tree NPU, the partially-parallel stochastic VNPU subnodes, and the programmable Barrel Shifters (BSs). The efficacy of this design flow is observed in the quantity of decoder parametrisations that it was able to generate for the characterisations of the proposed architectures in Chapters 4 and 5.

7.2 Future work

The architectures and automated design flow presented in Chapters 4, 5, and 6 are commendable for the high degree of run-time and design-time flexibility they confer to FPGA-based QC LDPC decoder designs. However, now that these have been established, their implementation results advocate the following areas of further research.

Decreased routing hardware requirements

One of the most obvious recurring observations from the results presented in Sections 4.4 and 5.4 was the disproportionately high hardware resource requirements of any decoder supporting all 19 of the different frame lengths in the WiMAX code family. This is caused by the hugely increased complexity of the programmable BSs, which cyclically shift the messages transmitted between the NPUs. These BSs must be capable of supporting any shift amount up to $Z-1$, hence requiring Z inputs and outputs, which does not decrease with the overall decoder parallelism ψ. Furthermore, when supporting a set of PCMs having multiple submatrix sizes z, these programmable BSs must also be able to perform the function of z-sized BSs. Even with the optimisations discussed in Section 4.3.3.3, this results in the large hardware resource utilisation shown previously, which also places limitations on the maximum achievable operating frequency and hence throughput. These factors suggest that this component is the primary candidate for further research and optimisation efforts, as decreasing the hardware requirements of each BS employed in the proposed architectures would have a large impact on its resulting characteristics. For example, it was stated in Section 4.4.6 that the LDPC PCMs in the forthcoming 5G New Radio (NR) standard will support z values that have been specifically selected with highly-flexible and highly-parallel decoder designs in mind. More specifically, the z values are all multiples of powers of 2, which allows hardware-efficient Banyan networks to be used for the implementation of the BSs.

Flexible highly-parallel decoders

It has been stated multiple times throughout this thesis that fully-parallel decoder architectures are not compatible with run-time flexibility due to the large degree of routing required. Simultaneously, it has also been demonstrated that stochastic decoders requiring a large number of DCs must be implemented with higher levels of parallelism than are possible in the proposed stochastic architecture, in order to support competitive processing throughputs. The level of parallelism in the decoder architectures proposed in this thesis is upper-bounded by the maximum quasi-cyclic submatrix size z, in order to avoid data clashes in memory accesses. Further research into strategies to increase this parallelism limit, without incurring significant performance or flexibility penalties, would therefore be beneficial to continued performance improvements.

Decreased DC requirements

In addition to increasing the parallelism and reducing the hardware resources (hence decreasing the critical path and increasing the clock frequency), the throughput of the proposed stochastic decoder could be improved by reducing the number of DCs it requires per frame. There are already several techniques of doing so reported in the current literature [91, 94, 95], which would require further investigation to determine their compatibility with the proposed partially-parallel architecture.

Low processing energy consumption

It is unfortunate that the majority of the designs reviewed in Chapter 3 did not present any information about the decoder's energy consumption. As with all electronic devices, low energy consumption is a key figure of merit in communication systems, since it dictates how long mobile devices can function between battery recharges, as well as dictating the cost and environmental impact of operating base station equipment. This provides a motivation to investigate the factors behind energy consumption in FPGA-based LDPC decoders, possibly by implementing some of the published designs and measuring their energy consumption directly. Drawing upon these results, FPGA-based LDPC decoders having low energy consumption could then be designed.

Characterisation of the NDS coefficient

The value chosen for the Noise-Dependent Scaling (NDS) coefficient γ is known to have a large impact on a stochastic decoder's Bit Error Rate (BER) performance, Frame Error Rate (FER) performance, and the average number of DCs required per frame. It has been observed that the behaviours of differing values of γ depend on the coding rate, but not the frame length or the structure of the PCM. However, a full characterisation and mathematical derivation of this coefficient has never been provided, leaving stochastic decoder designers to heuristically choose a value of their own.

Enhanced offline design flow

Due to the fact that the offline design flow is implemented using a high-level programming language, its usefulness could be improved in a variety of ways. For example, greater flexibility over the the fixed-point LLR width W or the degree of parallelism ψ (beyond expressing an integer reduction factor Q) could provide designers with further degrees of freedom to investigate optimal parametrisations for their specific design objectives. Additionally, currently the process of determining whether or not to use the pipelining registers in the proposed stochastic decoder architecture is fully manual, and depends on a user synthesising two competing parametrisations and comparing their characteristics. This element could theoretically be partially optimised, including an automated tool for converting any input PCM to its column-orthogonal form with a minimal number of stall cycles τ.

However, adding further degrees of freedom, such as variable LLR widths and greater flexibility over the degree of parallelism, to the offline design flow could reduce its value as a simple user-friendly tool. Its usefulness could be enhanced further, therefore, by adding the capability for it to recommend optimal values of these flexible parameters based on a user's prioritisation of one decoder characteristic over another. For example, if a user were to prioritise high transmission energy efficiency over low hardware resource requirements, the design flow could recommend a large value of the LLR word width W. In future versions this could be supported by a high-level graphical user interface, which could visually display the effects of various parameter choices to the user at design-time.

7.3 Concluding remarks

This thesis has comprehensively shown that designing high-performance FPGA-based LDPC decoders supporting run-time flexibility is a necessary but incredibly difficult task. However, when a decoder architecture is designed with this in mind from the start, very high levels of run-time flexibility may be achieved subject to some other performance trade-offs. These trade-offs are caused by the requirement for a flexible decoder to support the least hardware-friendly aspects of all of its supported codes simultaneously. Often these unfavourable parameters do not precisely offset each other, leading to inefficiencies in the synthesised hardware and hence reduced performance in one or more characteristics.

It is for these reasons that a highly-flexible LDPC decoder that also achieves high performance in all of its other characteristics remains elusive. Such a decoder is only possible when the supported codes have been designed with flexible hardware representations in mind. This need for combined holistic code-and-decoder design was one of the original catalysts for the quasi-cyclic code structure, which is now prevalent throughout high-performance decoder architectures and state-of-the-art communications standards. It is therefore hoped that the mainstream adoption of the 5G NR LDPC codes may bring about a new generation of codes that have been designed to facilitate straightforward run-time flexibility.

Bibliography

[1] R. G. Gallager, “Low-density parity-check codes,” *IEEE Trans. Inform. Theory*, vol. 8, no. 1, 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, pp. 1645–1646, 1996.

[3] 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.

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

[5] ETSI, “ETSI EN 302 307 v1.3.1 Digital Video Broadcasting (DVB); Second generation,” 2013.

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

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

[8] 3GPP, “R1-1611081 Final Report,” in *3GPP TSG RAN WG1 Meeting 86bis*, Lisbon, Portugal, oct 2016, pp. 83–89.

[9] 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.

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

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

[12] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, "Stochastic Decoding of LDPC Codes over GF(q)," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 939–950, 2013.

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

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

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

[16] L. Sun, H. Song, Z. Keirn, and B. V. K. V. Kumar, "Field programmable gate array (FPGA) for iterative code evaluation," *IEEE Trans. Magn.*, vol. 42, no. 2, pp. 226–231, feb 2006.

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

[18] 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.

[19] 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.

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

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

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

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

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

[25] C. Beuschel and H. Pfleiderer, "FPGA implementation of a flexible decoder for long LDPC codes," in *Int. Conf. Field Programmable Logic Applicat.* Heidelberg, Germany: IEEE, sep 2008, pp. 185–190.

[26] 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.

[27] H. Li, Y. S. Park, and Z. Zhang, "Reconfigurable architecture and automated design flow for rapid FPGA-based LDPC code emulation," in *Int. Symp. Field Programmable Gate Arrays.* Monterey, CA, USA: ACM, feb 2012, pp. 167–170.

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

[29] O. Boncalo, P. F. Mihancea, and A. Amaricai, "Template-based QC-LDPC decoder architecture generation," in *Int. Conf. Inform., Commun. Signal Processing.* Singapore: IEEE, dec 2015, pp. 1–5.

[30] G. Falcao, M. Gomes, J. Gonqalves, P. Faia, and V. Silva, "HDL Library of Processing Units for an Automatic LDPC Decoder Design," in *Conf. Ph.D. Research Microelectronics and Electronics*, Otranto, Italy, jun 2006, pp. 349–352.

[31] A. A. Madi, A. Ahaitouf, and A. Mansouri, "Min-Sum Algorithm based efficient high level methodology for design, simulation and hardware implementation of LDPC decoders," in *Int. Conf. Innovative Computing Technol.*, Casablanca, Morocco, sep 2012, pp. 218–223.

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

[33] 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.

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

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

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

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

[38] 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.

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

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

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

[42] 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.

[43] 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 *Int. Conf. Commun.* Beijing, China: IEEE, may 2008, pp. 1155–1160.

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

[45] 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.

[46] J. Zhang and M. P. C. Fossorier, “Shuffled Iterative Decoding,” *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 209–213, mar 2005.

[47] M. Awais and C. Condo, “Flexible LDPC decoder architectures,” *VLSI Design*, pp. 1–16, 2012.

[48] P. Schläfer, C. Weis, N. Wehn, and M. Alles, “Design space of flexible multigigabit LDPC decoders,” *VLSI Design*, pp. 1–10, 2012.

[49] F. Quaglio, F. Vacca, and G. Masera, “Low Complexity, Flexible LDPC Decoders,” in *IST Mobile Summit*, Dresden, Germany, jun 2005, pp. 1–6.

[50] 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.

[51] A. I. Vila Casado, M. Griot, and R. D. Wesel, “Improving LDPC decoders via informed dynamic scheduling,” in *Inform. Theory Workshop.* Tahoe City, CA, USA: IEEE, sep 2007, pp. 208–213.

[52] C. Healy and R. C. de Lamare, “Knowledge-Aided Informed Dynamic Scheduling for LDPC Decoding,” in *Int. Conf. Commun. Workshop*, no. 1. London, UK: IEEE, jun 2015, pp. 2212–2217.

[53] A. I. Vila Casado, M. Griot, and R. D. Wesel, “LDPC decoders with informed dynamic scheduling,” *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3470–3479, 2010.

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

[55] 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. Papers*, vol. 61, no. 7, pp. 2150–2158, jul 2014.

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

[57] X. Zuo, I. Perez-Andrade, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “Improving the Tolerance of Stochastic LDPC Decoders to Overclocking-Induced Timing Errors: A Tutorial and a Design Example,” *IEEE Access*, vol. 4, pp. 1607–1629, 2016.

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

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

[60] S. S. Tehrani, W. J. Gross, and S. Mannor, “Stochastic decoding of LDPC codes,” *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, 2006.

[61] 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.

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

[63] S. Mhaske, H. Kee, T. Ly, A. Aziz, and P. Spasojevic, "High-Throughput FPGA-Based QC-LDPC Decoder Architecture," in *Veh. Technol. Conf.* Boston, MA, USA: IEEE, sep 2015, pp. 1–5.

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

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

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

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

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

[69] H.-C. Lee, M.-R. Li, J.-K. Hu, P.-C. Chou, and Y.-L. Ueng, "Optimization Techniques for the Efficient Implementation of High-Rate Layered QC-LDPC Decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 2, pp. 457–470, feb 2017.

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

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

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

[73] Z. Cui and Z. Wang, “A 170 Mbps (8176, 7156) quasi-cyclic LDPC decoder implementation with FPGA,” in *Int. Symp. Circuits Syst.* Kos, Greece: IEEE, may 2006, pp. 5095–5098.

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

[75] I. Kuon, R. Tessier, and J. Rose, “FPGA architecture: survey and challenges,” *Found. Trends Electron. Design Automation*, vol. 2, no. 2, pp. 135–253, 2007.

[76] Y. Sun, M. Karkooti, and J. R. Cavallaro, “High throughput, parallel, scalable LDPC encoder/decoder architecture for OFDM systems,” in *Workshop Design, Applicat., Integration Software.* Richardson, TX, USA: IEEE, oct 2006, pp. 39–42.

[77] N. Chen, Y. Dai, and Z. Yan, “Partly parallel overlapped sum-product decoder architectures for quasi-cyclic LDPC codes,” in *Workshop Signal Process. Syst.* Banff, AB, Canada: IEEE, oct 2006, pp. 220–225.

[78] 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.

[79] A. Darabiha, A. C. Carusone, and F. R. Kschischang, “Block-Interlaced LDPC Decoders With Reduced Interconnect Complexity,” *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 55, no. 1, pp. 74–78, jan 2008.

[80] D. E. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *Workshop Signal Process. Syst.*, vol. 5. Austin, TX, USA: IEEE, oct 2004, pp. 107–112.

[81] A. Darabiha, A. C. Carusone, and F. R. Kschischang, “Power reduction techniques for LDPC decoders,” *IEEE J. Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, 2008.

[82] Z. Chen, X. Zhao, X. Peng, D. Zhou, and S. Goto, “An Early Stopping Criterion for Decoding LDPC Codes in WiMAX and WiFi Standards,” in *Int. Symp. Circuits Syst.* Paris, France: IEEE, may 2010, pp. 473–476.

[83] S. Brown and J. Rose, “FPGA and CPLD architectures: a tutorial,” *IEEE Design Test Comput.*, vol. 13, no. 2, pp. 42–57, 1996.

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

[85] J. P. Hayes, "Introduction to Stochastic Computing and its Challenges," in *Annu. Design Automation Conf.*, San Francisco, CA, USA, jun 2015, pp. 2–4.

[86] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Trans. Embedded Computing Syst.*, vol. 12, no. 2s, pp. 1–19, 2013.

[87] B. R. Gaines, "Stochastic computing systems," *Advances inform. syst. sci.*, vol. 2, no. 2, pp. 37–172, 1969.

[88] I. Perez-Andrade, "Timing-Error-Tolerant Iterative Decoders," Ph.D. dissertation, University of Southampton, 2016.

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

[90] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett.*, vol. 39, no. 3, pp. 299–301, 2003.

[91] Y.-L. Ueng, C.-Y. Wang, and M.-R. Li, "An Efficient Combined Bit-Flipping and Stochastic LDPC Decoder Using Improved Probability Tracers," *IEEE Trans. Signal Process.*, vol. 65, no. 20, pp. 5368–5380, 2017.

[92] I. Perez-Andrade, S. Zhong, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "Stochastic Computing Improves the Timing-Error Tolerance and Latency of Turbo Decoders: Design Guidelines and Tradeoffs," *IEEE Access*, vol. 4, pp. 1008–1038, 2016.

[93] Q. Zhang, Y. Chen, D. Wu, X. Zeng, and Y.-l. Ueng, "An area-efficient architecture for stochastic LDPC decoder," in *Int. Conf. Digital Signal Processing (DSP)*. Singapore: IEEE, jul 2015, pp. 244–247.

[94] D. Wu, Y. Chen, Q. Zhang, Y.-l. Ueng, and X. Zeng, "Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 63, no. 9, pp. 873–877, sep 2016.

[95] S. S. Tehrani, A. Naderi, G.-A. Kamendje, and S. Hemati, "Majority-Based Tracking Forecast Memories for Stochastic LDPC Decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, 2010.

[96] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic Implementation of LDPC Decoders," in *Asilomar Conf. Signals Syst. Comput.* IEEE, 2005, pp. 713–717.

[97] X.-R. Lee, C.-L. Chen, H.-C. Chang, and C.-Y. Lee, "A 7.92 Gb/s 437.2 mW Stochastic LDPC Decoder Chip for IEEE 802.15.3c Applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 2, pp. 507–516, 2014.

[98] I. Perez-Andrade, X. Zuo, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, “Analysis of voltage- and clock-scaling-induced timing errors in stochastic LDPC decoders,” in *Wireless Commun. Networking Conf.* Shanghai, China: IEEE, apr 2013, pp. 4293–4298.

[99] C. Winstead, V. C. Gaudet, A. Rapley, and C. Schlegel, “Stochastic iterative decoders,” in *Int. Symp. Inform. Theory.* Adelaide, Australia: IEEE, sep 2005, pp. 1116–1120.

[100] S. S. Tehrani, “Stochastic Decoding of Low-Density Parity-Check Codes,” Ph.D. dissertation, McGill University, Montreal, Canada, 2011.

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

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

[103] K. Wang, N. Liu, B. Sun, and H. Sun, “A configurable FPGA implementation of PEG-based PS-LDPC decoder,” in *Int. Conf. Pervasive Computing Signal Process. Applicat.* Harbin, China: IEEE, sep 2010, pp. 670–674.

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

[105] F. Demangel, N. Fau, N. Drabik, F. Charot, and C. Wolinski, “A generic architecture of CCSDS low density parity check decoder for near-earth applications,” in *Conf. Design Automation Test Europe.* Nice, France: European Design and Automation Association, apr 2009, pp. 1242–1245.

[106] P. Saunders and A. Fagan, “A high speed, low memory FPGA based LDPC decoder architecture for quasi-cyclic LDPC codes,” in *Int. Conf. Field Programmable Logic Applicat.* Madrid, Spain: IEEE, aug 2006, pp. 1–6.

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

[108] Z. Cao, J. Kang, and P. Fan, “An FPGA implementation of a structured irregular LDPC decoder,” in *Int. Symp. Microwave Antenna Propagation EMC Technol. Wireless Commun.*, vol. 1. Beijing, China: IEEE, aug 2005, pp. 1050–1053.

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

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

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

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

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

[114] V. A. Chandrasetty and S. M. Aziz, "FPGA implementation of high performance LDPC decoder using modified 2-bit min-sum algorithm," in *Int. Conf. Comput. Research Develop.* Kuala Lumpur, Malaysia: IEEE, may 2010, pp. 881–885.

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

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

[117] R. Zarubica, S. G. Wilson, and E. Hall, "Multi-Gbps FPGA-based low density parity check (LDPC) decoder design," in *Global Telecommun. Conf.*, no. 1. Washington, DC, USA: IEEE, nov 2007, pp. 548–552.

[118] Y. Dai, Z. Yan, and N. Chen, "Optimal overlapped message passing decoding of quasi-cyclic LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 5, pp. 565–578, 2008.

[119] C. Spagnol, W. Marnane, and E. Popovici, "Reduced complexity, FPGA implementation of quasi-cyclic LDPC decoder," in *Eur. Conf. Circ. Theory Design*, vol. 1. Cork, Ireland: IEEE, aug 2005, pp. 289–292.

[120] M. Karkooti and J. R. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," in *Int. Conf. Inform. Technol. Coding Computing.* Las Vegas, NV, USA: IEEE, apr 2004, pp. 579–585.

[121] L. Xiong, Z. Tan, and D. Yao, "The moderate-throughput and memory-efficient LDPC decoder," in *Int. Conf. Signal Process.* Beijing, China: IEEE, nov 2006, pp. 1–4.

[122] Softjin Technologies, "LDPC decoder for DVB-S2." [Online]. Available: http://www.softjin.com/IP_Datasheet_PDF_version/LDPC_Decoder_datasheet.pdf

[123] Unicore Systems Ltd, "CCSDS C2 LDPC encoder/decoder IP cores," 2011. [Online]. Available: http://unicore.co.ua/uploads/File/CCSDS_XX_user_manual(netlist).pdf

[124] ——, "IEEE 802.16e (WiMAX) LDPC decoder IP core," 2009. [Online]. Available: http://unicore.co.ua/uploads/File/ldpc_dec_brief.pdf

[125] Blue Rum Consulting Limited, "802.11n/802.11ac LDPC decoder," 2013. [Online]. Available: http://www.bluerum.co.uk/consulting/datasheets/BRC008_LdpcDecRtlDs.pdf

[126] Turbo Concept, "ITU G.hn LDPC decoder." [Online]. Available: http://www.turboconcept.com/prod_tc4400.php

[127] Creonic GmbH, "IEEE 802.11ad WiGig LDPC decoder product brief," 2014. [Online]. Available: http://www.creonic.com/images/product_briefs/PB_Creonic_IEEE_802_11ad_WiGig_LDPC_Decoder_IP.pdf

[128] IPrium Ltd., "I.6 LDPC encoder/decoder IP core short description," 2013. [Online]. Available: https://www.iprium.com/bins/pdf/iprium_ug_i6_ldpc_codec.pdf

[129] TrellisWare Technologies, "Flexible low-density parity-check (F-LDPC)," 2014. [Online]. Available: http://www.trellisware.com/products/fec-products/f-ldpc/

[130] Logic Fruit Technologies, "LDPC decoder IP specification," 2010. [Online]. Available: http://www.logic-fruit.com/resource/LDPCDecoderIP.pdf

[131] L. Hanzo, S. X. Ng, T. Keller, and W. Webb, *Quadrature Amplitude Modulation*. Chichester: Wiley-IEEE Press, 2004.

[132] 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. Papers*, vol. 57, no. 1, pp. 116–124, jan 2010.

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

[134] 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.

[135] 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," 2012.

[136] 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.

[137] 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.

[138] 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. Consumer Electron.*, vol. 59, no. 3, pp. 467–475, 2013.

[139] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, “Efficient implementations of the sum-product algorithm for decoding LDPC codes,” in *Global Telecommun. Conf.*, vol. 2. San Antonio, TX, USA: IEEE, nov 2001, pp. 1–6.

[140] 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.

[141] ZTE, “R1-1701598 Further consideration of Flexibility of LDPC Codes for NR,” in *3GPP TSG RAN WG1 Meeting 88*, Athens, Greece, feb 2017.

[142] C. Roth, P. Meinerzhagen, C. Studer, and A. Burg, “A 15.8 pJ/bit/iter quasi-cyclic LDPC decoder for IEEE 802.11n in 90 nm CMOS,” in *Asian Solid-State Circuits Conf.* Beijing, China: IEEE, nov 2010, pp. 313–316.

[143] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, “A Survey and Evaluation of FPGA High-Level Synthesis Tools,” *IEEE Trans. Comput.-Aided Design Integrated Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, 2016.

[144] J. Andrade, G. Falcao, and V. Silva, “Flexible design of wide-pipeline-based WiMAX QC-LDPC decoder architectures on FPGAs using high-level synthesis,” *Electron. Lett.*, vol. 50, no. 11, pp. 839–840, 2014.

[145] J. Andrade, N. George, K. Karras, D. Novo, F. Pratas, L. Sousa, P. Ienne, G. Falcao, and V. Silva, “Design Space Exploration of LDPC Decoders using High-Level Synthesis,” *IEEE Access*, vol. X, no. JULY, pp. 1–14, 2017.