Efficient Decoding Approaches for Rate-Compatible LDPC Codes of 5G-Enabled IoT Networks

A Thesis

Submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

by

SIVARAMA PRASAD TERA

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY

GUWAHATI -781039, ASSAM, INDIA.

Acceptance Certificate

This document serves to certify that the thesis titled 'Efficient Decoding Approaches for Rate-Compatible LDPC Codes of 5G-Enabled IoT Networks', authored by Sivarama Prasad Tera, a research scholar at the Department of Electronics and Electrical Engineering, Indian Institute of Technology, Guwahati - 781039, Assam, India, has been submitted for the completion of the Doctor of Philosophy degree. This work is entirely original, conducted under my dedicated supervision and guidance. It has comprehensively fulfilled all institutional requirements according to the academic standards and regulations of the institute. The work contained within this thesis has not been submitted elsewhere for any other degree or professional qualification and fulfills the criteria established by the academy for the award of the degree of Doctor of Philosophy.

Date:

Signature

(Dr.A. RAJESH)

Acknowledgments

First and foremost, I extend my heartfelt gratitude to my advisor, Dr. A. Rajesh, whose guidance and support have been instrumental throughout my Ph.D. journey. His relentless dedication and unwavering positivity have been pivotal in driving the progress of my work.

I am also immensely grateful to my doctoral committee members, Professor Prabin Kumar Bora, Professor Palathinkal Paily Roy, and Dr. Tony Jacob. Their profound knowledge and invaluable insights have greatly enriched my research, providing critical perspectives on various strategic questions that shaped the direction of my study.

Lastly, I must express profound thanks to my family, Dr. C. Ravi Kumar, Dr. M. Suresh, and friends, whose endless support and encouragement have played a crucial role in my achievements. Their unwavering love and support continue to inspire and sustain me every day. This acknowledgment reflects not only my appreciation but also the collective contributions that have fueled my academic journey.

Declaration

I, SIVARAMA PRASAD TERA, solemnly affirm that the information provided in this thesis is accurate, correct, and authentic. I am fully responsible for the content of this thesis and confirm that it represents my own work. I acknowledge that any misrepresentation of facts or plagiarism within this thesis may result in disciplinary actions as deemed appropriate by the examining authority. I am prepared to present supporting documents and evidence as required to substantiate the claims and data included in this thesis. This declaration is made in good faith, with the full understanding of the academic standards and integrity policies of the institution.

SIVARAMA PRASAD TERA
(146102033.)

Date: April 30, 2025

List of Abbreviations

5G Fifth Generation

AI Artificial Intelligence

Adam Adaptive Moment Estimation

AM Adaptive Moment Estimation

AR Auto-Regressive

ASIC Application-Specific Integrated Circuit

AWGN Additive White Gaussian Noise

BER Bit Error Rate

BGM Base Graph Matrices

BPSK Binary Phase Shift Keying

BRAM Block RAM

BS Barrel Shifters

CLB Configurable Logic Block

CMS Combined Min-Sum

CND Check Node Decoder

CN Check Node

CNPUs Check Node Processing Units

CMEM Check Node Memory

CNN Convolutional Neural Network

DC Decoding Cycles

DSP Digital Signal Processing

EDA Electronic Design Automation

ELBs Equivalent Logic Blocks

FEC Forward Error Correction

FPGA Field-Programmable Gate Array

HDL Hardware Description Language

HLS High-Level Synthesis

IoT Internet of Things

LDPC Low-Density Parity-Check

LLR Logarithmic-Likelihood Ratio

LUTs Look-Up Tables

mMTC Massive Machine-Type Communication

MAE Mean Absolute Error

ML Machine Learning

MSA Min-Sum Algorithm

MSE Mean Squared Error

NR New Radio

PCM Parity Check Matrix

QC Quasi-Cyclic

ROM Read-Only Memory

SD Stochastic Decoding

SD-CNN Stochastic Decoding-Convolutional Neural Network

SCN Stochastic Check Node

SVN Stochastic Variable Node

SPA Sum Product Algorithm

SNR Signal-to-Noise Ratio

SVND Stochastic Variable Node Decoder

SVPUs Stochastic Variable Node Processing Units

URLLC Ultra Reliable Low-Latency Communication

VND Variable Node Decoder

VNs Variable Nodes

VNPUs Variable Node Processing Units

Abstract

The introduction of 5G technology has significantly advanced the Internet of Things (IoT), enhancing performance and reliability across a range of applications, including smart cities and industrial automation. One of the critical elements driving these advancements is the adoption of Low-Density Parity-Check (LDPC) codes, which are highly regarded for their superior error-correction capabilities and efficiency in handling high data throughput. This progress has spurred an increasing demand for adaptable coding rates that can optimize responsiveness to diverse data requirements and varying environmental conditions, ranging from controlled settings to more dynamic scenarios. The 5G New Radio (NR) specifications encompass multiple LDPC coding rates and block lengths, ensuring broad compatibility across various devices and network configurations.

An FPGA-based fixed-point decoder was developed to support a wide range of LDPC code rates as specified by the 5G standard. This versatile decoder is applicable to various settings, ranging from the stable environments of smart factories to the fluctuating conditions encountered in remote sensing. Implementation results demonstrate the decoder's efficiency, showcasing its ability to deliver high throughput and low latency while optimizing hardware resource utilization. This FPGA-based fixed-point decoder provides a robust, adaptable, and high-performance solution for modern 5G communication systems, effectively meeting the stringent requirements of 5G enhanced Mobile Broadband (eMBB) applications.

The 5G NR specifications introduce various Base Graph Matrices (BGMs) to support different code rates, presenting significant challenges for traditional FPGA-based fixed-point LDPC decoders due to their complexity. To address these challenges, a flexible FPGA-based stochastic decoder was developed for rate-compatible 5G LDPC codes. This

approach converts channel probability values into stochastic bit sequences, which simplifies inter-node routing and reduces the number of interconnects compared to conventional fixed-point decoders. The proposed FPGA-based stochastic decoder offers a flexible, efficient, and scalable solution for 5G LDPC decoding, capable of managing a wide range of code rates with reduced hardware complexity and enhanced energy efficiency, providing a viable alternative to traditional fixed-point decoders.

With the growing application of Machine learning techniques in digital communication, this research explores the integration of deep learning with traditional stochastic decoding to improve the performance of 5G LDPC code decoding. The proposed Stochastic Decoding-Convolutional Neural Network (SD-CNN) architecture addresses the challenges posed by colored noise in 5G-enabled IoT applications, ensuring reliable and efficient communication across a wide range of real-world scenarios.

Contents

List of Figures

List of Tables

1 Introduction

The Internet of Things (IoT) refers to an interconnected network of devices that communicate and share data seamlessly, transforming various industries and everyday life [1]. The advent of 5G technology has significantly amplified the potential of IoT applications due to its superior capabilities, including higher data rates, reduced latency, and increased connectivity density [2]. These advancements enable a new era of IoT, characterized by enhanced performance and reliability, which are crucial for applications ranging from smart cities to industrial automation [3] [4]. One of the key factors driving these improvements is the implementation of advanced channel coding techniques, notably Low-Density Parity-Check (LDPC) codes. LDPC codes are highly regarded for their exceptional error-correction abilities and efficiency in handling high data throughput, which is critical for maximizing bandwidth utilization and achieving performance close to the Shannon limit, especially in

scenarios where bandwidth is limited [5]. Originating from the pioneering work of Robert G. Gallager in the 1960s, LDPC codes have evolved to become essential for robust and efficient communication [6].

LDPC codes are integral to several communication standards, including:

- IEEE 802.11 (WiFi): Enhances wireless local area network (WLAN) performance by improving error correction and increasing data reliability [7].
- IEEE 802.16e (WiMax): Provides wireless broadband access and benefits from LDPC's ability to manage high data rates and ensure reliable communication over long distances [8].
- LTE (Long-Term Evolution): Utilizes LDPC codes to enhance mobile communication by providing higher data rates and more reliable connections, essential for modern cellular networks [9].
- Emerging 5G technologies: Leverage LDPC codes to meet the rigorous demands of next-generation mobile networks, offering unprecedented speeds, capacity, and reliability [10].

In summary, the integration of LDPC codes in communication standards underscores their critical role in advancing IoT applications and realizing the full potential of 5G technology. This synergy is driving the transformation of communication infrastructures, ensuring they are robust, efficient, and capable of supporting the complex demands of modern digital systems.

1.1 The Critical Role of LDPC Codes in 5G-Enabled IoT

LDPC codes, originally introduced by Robert Gallager and gaining significant attention in the 1990s, play a crucial role in powering the IoT by supporting various applications and services. [11] [12]. These codes are celebrated for their capacity-approaching performance and their ability to facilitate efficient decoding algorithms, such as the Belief Propagation (BP) algorithm, due to their sparse parity-check matrix structure [13, 14, 15] .

In IoT networks, devices frequently operate in environments with substantial interference and noise, making robust error correction essential. LDPC codes provide the necessary error correction to maintain data integrity and minimize re-transmissions, thus conserving power—a critical factor for battery-powered IoT devices. These codes support the diverse requirements of 5G use cases, such as enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Machine-Type Communication (mMTC) [16]. The following attributes make LDPC codes ideal for 5G-enabled IoT applications:

- High Reliability: LDPC codes offer excellent error correction performance, ensuring reliable communication where data integrity is paramount, such as in healthcare and industrial IoT.
- Low Latency: The decoding process, particularly when implemented on hardware like FPGAs, can be highly parallelized, resulting in low latency. This is crucial for real-time communication applications, such as autonomous vehicles and industrial automation.
- High Throughput: LDPC codes support high data rates necessary for applications involving large data transmissions, such as video surveillance and smart city infrastructures.
- Energy Efficiency: Iterative decoding algorithms can be optimized for low power consumption, making LDPC codes suitable for battery-powered IoT devices.
- Enhanced Flexibility: LDPC codes support a wide range of code rates and block sizes, providing the flexibility to meet various 5G use case requirements.
- Scalability: The decoding complexity of LDPC codes can be adjusted, making them suitable for a variety of device types, from high-performance smartphones to low-power IoT sensors.

LDPC codes play a pivotal role in numerous 5G-enabled IoT applications, including:

- Smart Cities: Ensuring reliable communication among numerous interconnected sensors and devices for managing infrastructure, traffic, and utilities, thereby enabling efficient data collection and analysis.
- Healthcare: Enhancing the reliability of remote monitoring and telemedicine applications by ensuring the accurate transmission of critical medical data.
- Industrial IoT (IIoT): Supporting high-reliability, low-latency communication required for real-time monitoring and control of industrial processes.
- Autonomous Vehicles: Providing ultra-reliable and low-latency communication necessary for safe navigation and real-time decision-making.

The global standard for 5G, known as 5G New Radio (NR), incorporates LDPC codes for data channels due to their superior performance characteristics [17]. LDPC codes offer improved error correction compared to previous generations Turbo codes, especially at higher code rates. They also provide enhanced flexibility by supporting a wide range of code rates and block sizes, making them suitable for various device types, from high-performance smartphones to low-power IoT sensors.

1.2 5G Technology and QC LDPC Codes

The advent of 5G technology has revolutionized wireless communications by offering ultra-reliable, high data throughput with minimal latency, catering to a variety of applications. At the core of achieving such high reliability and efficiency in 5G networks are Quasi-Cyclic (QC) LDPC codes [18]. The quasi-cyclic structure of QC LDPC codes simplifies hardware implementation for encoding and decoding processes, making them especially advantageous for 5G applications due to their robust performance and adaptability to varying data rates and channel conditions.

Structure and benefits of QC LDPC Codes: The 5G NR specifications [19] incorporate QC LDPC codes to meet the high throughput and low latency demands of contemporary communication systems. QC LDPC codes utilize two Base Graph Matrices (BGMs), H_{b1}

and H_{b2}, and fifty-one possible lifting sizes or expansion factors z, designed to support a wide range of code rates. The characteristics of these BGMs include:

- BGM1: This base graph, larger in size with 46 block rows and 68 block columns, is typically used to generate LDPC codes with higher code rates ranging from 1/3 to 8/9. This allows adaptation for scenarios where less robust error correction is sufficient due to better channel conditions or where higher throughput is necessary.
- BGM2: Slightly smaller, with 42 block rows and 52 block columns, this base graph supports lower code rates ranging from 1/5 to 2/3. It is suited for scenarios that demand more robust error correction capabilities, appropriate for poorer channel conditions.

Role of Lifting Sizes: The lifting sizes or expansion factors z are crucial for the scalability and adaptability of QC LDPC codes. With fifty-one different z values ranging from 2 to 384, these parameters allow fine-tuning of the Parity Check Matrix (PCM) in terms of size and complexity. This tuning is essential for Adjusting the Block Length. The size of z directly influences the block size of the code, impacting both the latency and throughput of the communication system. Larger block sizes generally improve throughput but may increase latency and decoding complexity. Selecting an appropriate z allows optimization for specific channel conditions. Lower z values might be used in less noisy conditions where lower latency is prioritized, while higher values could be employed in noisier conditions to ensure robust error correction.

The use of 102 different PCMs derived from these two base graphs allows 5G NR to effectively address a broad range of service requirements:

- Enhanced Mobile Broadband (eMBB): Utilizes higher code rates for high throughput in urban areas or scenarios with good signal quality.
- Ultra-Reliable Low-Latency Communications (URLLC): Employs lower code rates with more robust error correction to ensure reliability and low latency, crucial for applications like autonomous driving or remote surgery.

- Massive Machine Type Communications (mMTC): Can use various code rates depending on device density and network load.

The 5G NR standard implements a highly flexible coding scheme using LDPC codes, which are adapted for different communication scenarios through a variety of PCMs and BGMs. This use of 102 different PCMs derived from the two base graphs enables 5G NR to efficiently address a diverse range of service requirements. This flexibility allows 5G networks to dynamically adapt to changing channel conditions and user demands, optimizing both robustness and efficiency in data transmission across various deployment scenarios.

1.3 Significance of various approaches

1.3.1 Need for FPGA Prototyping

The design of LDPC decoders encompasses a variety of algorithms and levels of parallelization, offering designers multiple pathways to achieve the desired performance. However, the development of a comprehensive LDPC decoder is complex, as system characteristics such as throughput, latency, hardware resource requirements, error correction capability, energy and bandwidth efficiency, and flexibility are all interrelated. These factors are shaped by elements such as architecture, LDPC code, algorithm, and the number of decoding iterations. The intricate interaction among these parameters makes the design and implementation of an effective LDPC decoder both challenging and time-consuming. To assess achievable performance under realistic conditions, designers typically prototype their designs at different stages.

Field-Programmable Gate Arrays (FPGAs) are ideal for rapid prototyping and high-speed parallel logic processing. FPGA-based LDPC decoders are valuable in research settings due to their fast and parallel logic resources, enabling the measurement of Bit Error Rate (BER) performance for various codes [20, 21, 22]. Custom FPGA implementations can complete simulations in hours, reducing the time required compared to traditional computing methods. However, most FPGA-based LDPC decoder designs are specific to one code, necessitating additional design work to adapt the architecture for new LDPC codes

[23, 24]. This adaptation requires re-synthesis to produce a new FPGA configuration file, which must be downloaded onto the FPGA before decoding the new code.

Utilizing an FPGA for prototyping 5G New Radio (NR) LDPC decoders offers significant benefits due to the FPGA's flexibility and high processing capabilities. Here's how FPGAs can be advantageous in this context:

- Rapid Prototyping: FPGAs enable quick design iteration and testing, accelerating experimentation and allowing designers to explore various configurations and optimizations without extensive retooling or fabrication.
- Parallel Processing: LDPC decoding is computationally intensive. FPGAs' configurable logic blocks and interconnects facilitate parallel processing, enabling the simultaneous processing of multiple data streams or decoding algorithms.
- Testing in Realistic Conditions: FPGAs can test LDPC decoders in real-world environments, providing valuable feedback on robustness and efficiency, thereby refining their design.
- Adaptability to Standards Evolution: FPGAs offer a flexible platform for adapting to changes in 5G technology standards, allowing for reprogramming to accommodate these changes and protecting investment in development hardware.
- Cost Efficiency: Despite initial costs, FPGAs can be more cost-effective over time by reducing the need for multiple ASIC designs, which are more expensive and time-consuming to fabricate.

Overall, using FPGAs in developing and testing 5G NR LDPC decoders not only accelerates the design process but also enhances the ability to achieve high-performance, adaptable solutions ready for next-generation wireless networks.

1.3.2 Significance of Stochastic Decoding

The 5G NR specifications indicate that the BGM H_{b1} can support a wide range of code rates from 1/3 to 8/9. The irregular degrees and diverse connections within these BGMs create a

complex routing network when using traditional FPGA-based fixed-point LDPC decoders that rely on message passing algorithms like the Sum Product Algorithm (SPA) [25] and the Min-Sum Algorithm (MSA) [26]. This complexity is further exacerbated by the use of multi-bit, wide-channel Logarithmic-Likelihood Ratio (LLR) messages, which demand high FPGA resources. An alternative solution is converting the channel probability values corresponding to their respective channel LLRs into stochastic bit sequence representation in stochastic decoding (SD) [27, 28]. This approach enables each stochastic bit sequence to utilize only one wire for the bit-wise transmission of extrinsic values between nodes, in contrast to the multiple wires required for broad LLRs in standard decoders, thereby considerably simplifying inter-node routing. The bit-wise operations in Stochastic Variable Node (SVN) and Stochastic Check Node (SCN) units are facilitated by basic logic units. These stochastic LDPC decoders match the error correction performance of conventional fixed-point decoders [29].

The number of node-interconnects (n_I) of a decoder, which determines routing complexity, is given by:

$$n_I = 2 \times N \times e_l \times d_v. \tag{1.1}$$

Where N represents the length of the codeword, e_l denotes the length of the extrinsic message, and d_v refers to the degree of variable nodes. Consider a 1/3-rate LDPC code with a codeword length $N = 3808$ and an average variable node degree $d_v = 4.56$. The required number of node-interconnects for SPA and SD are 138,916 and 34,729, respectively. The node-interconnects for SD are reduced by a factor of four due to an extrinsic message length $e_l = 1$ in SD, in contrast to a minimum $e_l = 4$ in fixed-point LDPC decoders utilizing SPA.

To address the interconnection network problem, stochastic computation was proposed by Gaudet et al. in 2003 as an alternative to SPA [27, 30, 31, 32, 33].

1.3.3 Significance of CNN based approach to decoding

In communication systems, noise correlation frequently stems from factors like filtering, oversampling, and device noise. This is particularly noticeable in digital systems, where

pink noise results from clock jitter and phase noise [34]. The decoding performance of LDPC codes deteriorates under colored noise due to its complexity. A possible remedy is whitening, which transforms colored noise into white noise. However, this method requires numerous matrix multiplications, making it cumbersome for long codes.

Deep learning techniques are being investigated to tackle digital communication challenges, such as channel coding. An innovative design, Iterative BP-CNN, has been introduced to enhance the signal-to-noise ratio (SNR) per bit (E_b/N_o) in environments with correlated noise [35]. This research led to the development of an SD-CNN architecture aimed at optimizing the decoding performance of 5G LDPC codes [36]. This architecture integrates a trained Convolutional Neural Network (CNN) with Stochastic Decoders (SD), processing received symbols iteratively between the two. The stochastic decoder initially processes the received symbols to produce preliminary decoding results, including an estimate of the channel noise. This noise estimate is then refined by the CNN to minimize estimation errors and improve noise estimation. By iterating between the SD and CNN, the SNR per bit is progressively enhanced, leading to improved decoding performance.

1.4 Structure of the thesis

This thesis is divided into several chapters, each building on the previous one to provide a thorough understanding of LDPC decoding techniques within the framework of 5G-enabled IoT applications.

- Chapter 2: This chapter provides a comprehensive tutorial-style introduction to 5G NR LDPC codes. It covers fundamental concepts and highlights the importance of LDPC codes in the 5G technology landscape, establishing a solid foundation for the discussions in the following chapters.

- Chapter 3: This chapter explores the architecture of a FPGA-based fixed-point 5G Rate-compatible LDPC decoder in detail. It describes each component of the decoder architecture, including the degree of parallelism, the memory management scheme, the flexible data path, the node processing units, and the control unit. The

chapter also presents various optimizations to enhance the implementation characteristics of the proposed architecture, ensuring the design is not confined to any specific code set.

- Chapter 4: Building on insights from the previous chapter, this chapter presents an alternative, generalized flexible FPGA-based LDPC decoder architecture that uses a stochastic approach. This design aims to reduce hardware resource usage compared to its fixed-point counterpart. The proposed structure and decoding algorithm are novel, representing the first stochastic LDPC decoder to support run-time flexibility over multiple code rate sets of the 5G NR standard.
- Chapter 5: This chapter introduces the CNN-based approach to LDPC decoding. It explains the SD-CNN architecture, which enhances the E_b/N_o in the presence of correlated noise. This innovative architecture combines a trained CNN with SD to improve the decoding capability through iterative processing between the SD and CNN.
- Chapter 6: The final chapter summarizes the key findings and offers suggestions for future research directions. It concludes with remarks on the significance of the work and highlights potential areas for further exploration in the field of LDPC decoding and 5G technology.

Each chapter is carefully structured to build upon the knowledge gained in the previous ones, culminating in a comprehensive understanding of advanced LDPC decoding techniques and their application in 5G-enabled IoT systems.

1.5 Novel contributions of the thesis

This thesis introduces several groundbreaking contributions to the field of LDPC decoding for 5G technology:

1. Generalized Architecture for Flexible FPGA-Based 5G Rate-compatible LDPC Decoder: One of the major contributions is the development of a generalized architec-

ture for a flexible, parameterizable FPGA-based fixed-point LDPC decoder. This architecture supports 5G LDPC codes and offers both design-time and single-clock-cycle run-time flexibility for a variety of 5G LDPC codes. This flexibility enables efficient and adaptable decoding solutions tailored to the diverse needs of different 5G applications.

2. First Stochastic LDPC Decoder for 5G Codes with Partially-Parallel Architecture: This thesis introduces the first stochastic LDPC decoder specifically designed for 5G LDPC codes, featuring a partially-parallel architecture. Traditional stochastic processing elements have been adapted to ensure compatibility with 5G LDPC codes, significantly enhancing the decoding process by utilizing the benefits of stochastic decoding.

3. Stochastic LDPC Decoder with Run-Time Flexibility: Another innovative contribution is the creation of the first stochastic LDPC decoder that offers run-time flexibility across multiple code rate families within the 5G NR standard. This feature allows the decoder to dynamically adapt to different code rates during operation, providing greater flexibility and efficiency in various communication scenarios.

4. Integration of Stochastic Decoding and Convolutional Neural Networks (SD-CNN): The thesis presents a novel SD-CNN structure aimed at improving the decoding performance of 5G LDPC codes in the presence of correlated channel noise. By integrating stochastic decoding with convolutional neural networks, this approach enhances the signal-to-noise ratio (SNR) per bit, resulting in more robust and reliable decoding in noisy environments. This represents a significant advancement in applying deep learning to decoding in digital communications.

2

Fifth-Generation (5G) New Radio (NR) LDPC codes

2.1 Foundational Overview

2.1.1 Fundamentals of a Basic digital Communication System

A digital communication system, depicted in Figure 2.1, is designed to process message signals provided by the source for efficient and reliable information delivery from origin to destination [37]. This involves several key stages, each incorporating mathematical principles to ensure reliable communication: source encoding, channel encoding, modulation, transmission through a communication channel, demodulation, channel decoding, and source decoding.

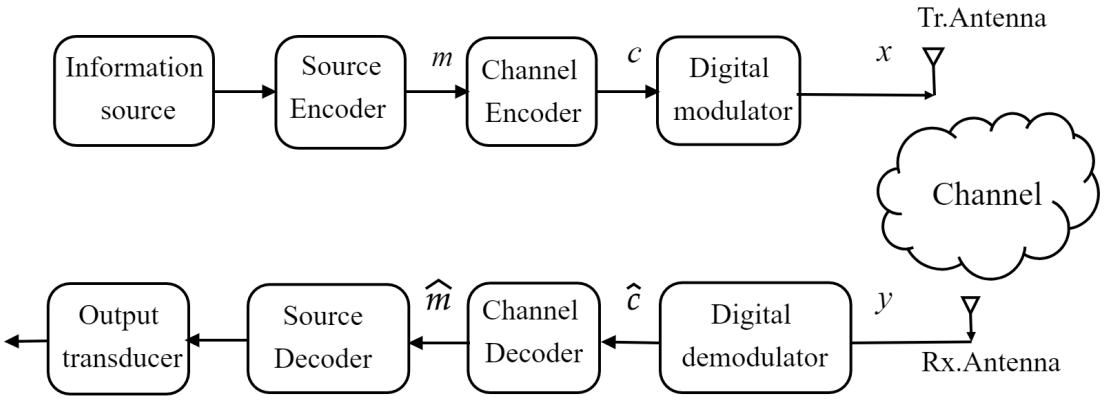


Figure 2.1: Schematic of a digital communication system.

1. Source Encoding: The aim of source encoding is to transform the source output into a sequence of binary digits, reducing redundancy while preserving essential information. This process is represented by the sequence:

$$\mathbf{m} = [m_1, m_2, \ldots, m_K], \tag{2.1}$$

where **m** is the K-bit message word. Although primarily qualitative, source encoding condenses the message into a more compact form.

2. Channel Encoding: Channel encoding introduces redundancy to the binary sequence to facilitate error detection and correction during transmission. The encoded message, or codeword, is expressed as:

$$\mathbf{c} = [c_1, c_2, \ldots, c_N], \tag{2.2}$$

where $N > K$. The relationship between the original message and the encoded message is defined by the coding rate R:

$$R = \frac{K}{N} = \frac{N-M}{N}, \tag{2.3}$$

where $M = N - K$ represents the number of parity bits added. The parity-check matrix H ensures that the codeword **c** satisfies:

$$H \times \mathbf{c}^T = 0. \tag{2.4}$$

3. Digital Modulation: The encoded binary sequence is converted into signal waveforms for transmission through modulation. For Binary Phase-Shift Keying (BPSK), modulation is defined as:

$$x_j = \begin{cases} +\sqrt{E_s} & \text{if } c_j = 0, \\ -\sqrt{E_s} & \text{if } c_j = 1, \end{cases} \tag{2.5}$$

where E_s is the energy per symbol. The modulated signal vector $\mathbf{x}$ is then transmitted through the channel.

4. The Communication Channel: The channel introduces noise to the transmitted signal. The received signal $\mathbf{y}$ can be modeled as:

$$\mathbf{y} = \mathbf{x} + \mathbf{n}, \tag{2.6}$$

where $\mathbf{n}$ represents noise added by the channel. In an Additive White Gaussian Noise (AWGN) channel, $\mathbf{n}$ follows a normal distribution:

$$\mathbf{n} \sim \mathcal{N}(0, N_0). \tag{2.7}$$

The Signal-to-Noise Ratio (SNR) per bit is given by:

$$\frac{E_b}{N_0} = \frac{1}{R} \times \frac{E_s}{N_0}. \tag{2.8}$$

5. Digital Demodulation: At the receiver, the received signal $\mathbf{y}$ is demodulated to estimate the transmitted binary sequence. Soft decisions are made using the Logarithmic-

Likelihood Ratio (LLR):

$$LLR_i = \log\left(\frac{P(c_i = 0 \mid y_i)}{P(c_i = 1 \mid y_i)}\right). \tag{2.9}$$

For BPSK in an AWGN channel, the LLR simplifies to:

$$LLR_i = 4 \times R \times \frac{E_b}{N_0} \times \mathrm{Re}(y_i). \tag{2.10}$$

6. Channel Decoding: The channel decoder corrects errors using the received sequence and the redundancy introduced by the channel encoder [38]. The decoder seeks the codeword $\hat{\mathbf{c}}$ that satisfies:

$$H \times \hat{\mathbf{c}}^T = 0. \tag{2.11}$$

The decoder's performance is measured by the Bit-error rate (BER), the probability of a bit error in the decoded sequence.

7. Source Decoding: The final step reconstructs the original source message from the decoded binary sequence. The source decoder approximates the original message using knowledge of the encoding method:

$$\hat{\mathbf{m}} = \mathrm{SourceDecoder}(\hat{\mathbf{c}}). \tag{2.12}$$

The reconstruction's accuracy depends on the efficiency of the encoding and decoding processes throughout the communication system.

In summary, a basic communication system encompasses a series of processes that transform a source message into a transmitted signal and back into a received message, minimizing errors. Mathematical models and equations are essential in each stage, helping to quantify and mitigate noise and interference, ensuring reliable communication.

2.2 Channel coding and LDPC Codes

Channel coding adds redundancy to a message to protect against errors from noise and other impairments in the transmission channel. This redundancy enables the detection and correction of errors, thereby preserving the integrity of the data being received. Channel codes such as block codes, convolutional codes, turbo codes, and LDPC codes are used for this purpose [39]. LDPC codes represent a category of linear block codes that are defined by a sparse PCM [40]. This matrix is characterized by its sparsity, having a high number of zeros and comparatively few ones. LDPC codes can be visualized using Tanner graphs [41], which are bipartite graphs with two types of nodes.

2.2.1 Basics of LDPC PCM

The PCM is a crucial component in the design and function of LDPC codes, underpinning their strong error-correcting abilities. Represented as H, the PCM is a binary matrix consisting of 0s and 1s. Its dimensions are $M \times N$, where M signifies the number of parity-check equations, and N represents the length of the codeword. A key feature of the PCM is its sparsity, meaning it contains far more 0s than 1s. This sparse nature is vital for the efficiency of LDPC codes, allowing the use of powerful decoding algorithms. For instance, a simple LDPC code with a PCM H comprising 3 parity-check equations and a codeword length of 7 is shown below:

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

In this matrix, each row corresponds to a parity-check equation, and each column represents a bit in the codeword. The sparsity of the PCM is essential to the efficiency of LDPC codes. LDPC codes are generally divided into two categories based on their PCM structure: Regular LDPC Codes and Irregular LDPC Codes [42, 43, 44, 45]. Understanding this distinction is important for grasping the design and performance traits of these codes. Regular LDPC codes have a uniform structure, making them easier to design and

implement. On the other hand, irregular LDPC codes provide greater flexibility and can be optimized for specific performance goals, although they are more complex to design [46].

- Regular LDPC Codes: In these codes, each row of the PCM contains the same number of 1s (denoted by d_c) and each column also contains the same number of 1s (denoted by d_v). For example, a regular LDPC code with d_v ones per column and d_c ones per row is termed a (d_v, d_c)-regular code. In such codes, each column has exactly d_v non-zero elements, and each row has $d_c = d_v \times N/(N-K)$ non-zero elements, with $d_c \ll (N-K)$.
- Irregular LDPC Codes: These codes have a PCM where the number of 1s per row and column can vary. Irregular LDPC codes typically exhibit superior error correction performance because their distribution of 1s can be optimized for particular requirements.

2.2.2 Tanner Graph Representation

A Tanner graph, depicted in Figure 2.2, is a graphical representation used to illustrate the relationships between the codeword bits and parity-check equations in the PCM H of LDPC codes [41]. This representation is essential for understanding and implementing LDPC decoding algorithms. The Tanner graph is a bipartite graph, meaning it consists of two distinct sets of nodes: variable nodes (VNs) and check nodes (CNs). Variable Nodes (VNs) are nodes that represent the bits of the codeword. Check Nodes (CNs) are nodes that represent the parity-check equations.

Edges in the graph connect variable nodes to check nodes based on the positions of 1s in the PCM H. Consider the following PCM H:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.14}$$

In this example, the matrix H has dimensions 3×7, meaning it consists of 3 parity-check equations and 7 codeword bits. An edge exists between a variable node VN_i and a

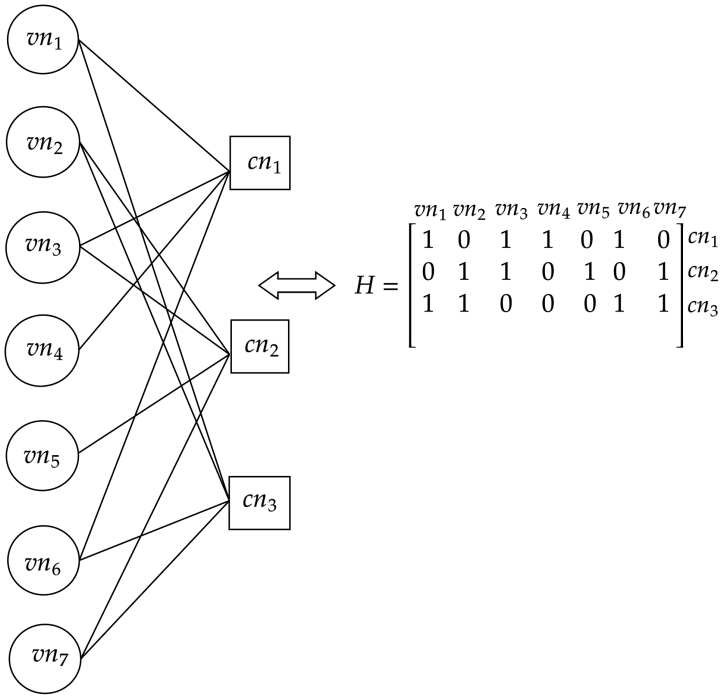


Figure 2.2: An example of a Tanner graph for a (7,4) LDPC code.

check node CN_j whenever $H_{ji} = 1$. This connection illustrates the relationship established by the parity-check matrix, linking each codeword bit to its corresponding parity-check equation. This construction process visually maps out the connections between the bits of the codeword and the parity-check equations, providing a clear framework for the iterative decoding algorithms used in LDPC codes.

2.2.3 Encoding of LDPC Codes

Encoding of a LDPC code involves creating a codeword **c** that adheres to the parity-check constraints defined by the PCM H [47]. This process introduces redundancy into the transmitted data, facilitating error correction at the receiver's end. The systematic encoding procedure entails deriving a generator matrix G from the PCM H. Systematic Encoding Procedure:

1. Construct the PCM H: The PCM H is designed to be sparse, with dimensions $M \times N$, where M represents the number of parity-check equations and N is the length of the codeword.

2. Form the Generator Matrix G: The generator matrix G is derived from the parity-check matrix H. For systematic codes, G takes the form $G = [I_K|P]$, where I_K is a $K \times K$ identity matrix and P is a $K \times (N-K)$ matrix derived from H. This structure ensures that the initial K bits of the codeword are the information bits, followed by the parity bits.

3. Generate the Codeword: The codeword $\mathbf{c}$ is generated by multiplying the vector of information bits $\mathbf{m}$ by the generator matrix G:

$$\mathbf{c} = \mathbf{m} \times G. \tag{2.15}$$

Here, $\mathbf{m}$ is the vector of K information bits.

This systematic encoding process guarantees that the original information bits are embedded directly within the codeword, with the necessary parity bits appended subsequently.

2.2.4 Decoding of LDPC Codes

As illustrated in Figure 2.3, decoding of the LDPC codes generally involves iterative algorithms that exchange messages between variable nodes and check nodes within the Tanner graph, updating probability estimates for each bit [48, 49]. Two commonly used iterative decoding algorithms are the Sum-Product Algorithm (SPA) [25] and the Min-Sum Algorithm (MSA) [26].

Sum-Product Algorithm (SPA): The SPA, also known as Belief Propagation, is a widely used iterative decoding method for LDPC codes [25]. SPA operates on the Tanner graph, with messages exchanged between VNs and CNs to update the probability estimates of codeword bits.

1. Initialization: Each variable node v_i is initialized with the log-likelihood ratio (LLR) based on the received signal y_i:

$$LLR_i = \log \frac{P(y_i|c_i = 0)}{P(y_i|c_i = 1)}. \tag{2.16}$$

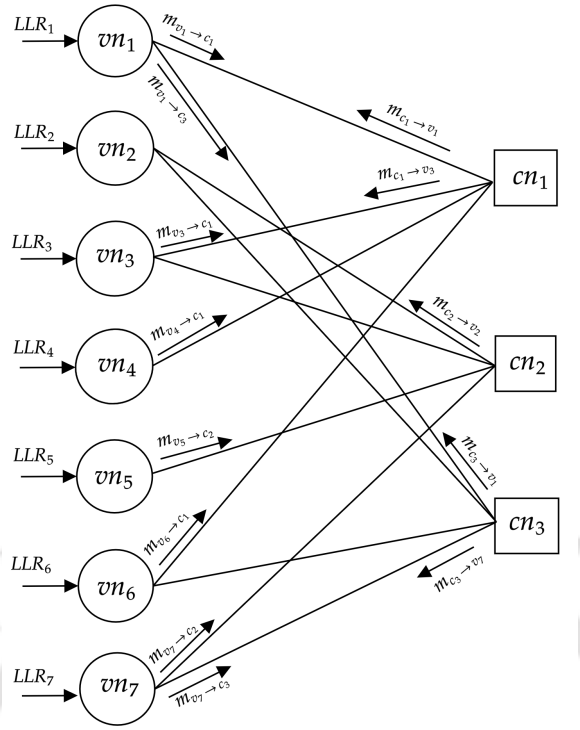


Figure 2.3: An example of node interactions in a (7,4) LDPC Tanner Graph.

2. Message Passing: The algorithm iteratively exchanges messages between variable nodes and check nodes. A message from a variable node to a check node represents the likelihood that the variable node's bit is either 0 or 1. Similarly, a message from a check node to a variable node represents how well the parity-check equation is satisfied given the incoming messages from other variable nodes.

   - Variable Node Update: Each variable node sends a message to a connected check node based on its LLR and incoming messages from other check nodes.

$$m_{v_i \to c_j} = LLR_i + \sum_{c_k \in N(v_i) \setminus c_j} m_{c_k \to v_i}, \tag{2.17}$$

     where $N(v_i) \setminus c_j$ represents all check nodes connected to variable node v_i except c_j.

   - Check Node Update: Each check node sends a message to a connected variable

node based on the incoming messages from other variable nodes.

$$m_{c_j \to v_i} = 2\tanh^{-1}\left(\prod_{v_k \in N(c_j)\setminus v_i} \tanh\left(\frac{m_{v_k \to c_j}}{2}\right)\right), \tag{2.18}$$

where $N(c_j)\setminus v_i$ represents all variable nodes connected to check node c_j except v_i.

3. Bit Decision: After a set number of iterations or upon convergence, the final LLR for each variable node is computed, and a hard decision is made for each bit.

$$LLR_{\text{final},i} = LLR_i + \sum_{c_j \in N(v_i)} m_{c_j \to v_i}. \tag{2.19}$$

$$\hat{c}_i = \begin{cases} 0 & \text{if } LLR_{\text{final},i} \geq 0, \\ 1 & \text{if } LLR_{\text{final},i} < 0. \end{cases} \tag{2.20}$$

Min-Sum Algorithm (MSA): The MSA is a simplified version of the SPA, designed to reduce computational complexity by approximating the sum-product operations with minimum operations [26]. Although this approximation can result in some performance loss, the MSA is more computationally efficient.

1. Initialization: Similar to SPA, each variable node v_i is initialized with the LLR based on the received signal y_i:

$$LLR_i = \log\frac{P(y_i|c_i=0)}{P(y_i|c_i=1)}. \tag{2.21}$$

2. Message Passing: MSA simplifies the message-passing process between variable nodes and check nodes.

   - Variable Node Update: Each variable node sends a message to a connected check node based on its LLR and the incoming messages from other check

nodes.

$$m_{v_i \to c_j} = LLR_i + \sum_{c_k \in N(v_i) \setminus c_j} m_{c_k \to v_i}. \tag{2.22}$$

- Check Node Update: Each check node sends a message to a connected variable node based on the minimum of the incoming messages from other variable nodes.

$$m_{c_j \to v_i} = \left(\prod_{v_k \in N(c_j) \setminus v_i} \text{sign}(m_{v_k \to c_j}) \right) \times \min_{v_k \in N(c_j) \setminus v_i} |m_{v_k \to c_j}|. \tag{2.23}$$

3. Bit Decision: After a set number of iterations or upon convergence, the final LLR for each variable node is computed, and a hard decision is made for each bit.

$$LLR_{\text{final},i} = LLR_i + \sum_{c_j \in N(v_i)} m_{c_j \to v_i}. \tag{2.24}$$

$$\hat{c}_i = \begin{cases} 0 & \text{if } LLR_{\text{final},i} \geq 0, \\ 1 & \text{if } LLR_{\text{final},i} < 0. \end{cases} \tag{2.25}$$

In summary, the decoding process uses the PCM H to iteratively correct errors in the received codeword, ensuring that $H \times \hat{\mathbf{c}}^T = 0$ [50]. The iterative message-passing algorithms leverage the sparse structure of H to update probability estimates and reconstruct the original information vector.

2.2.5 Quasi-Cyclic LDPC Codes

QC LDPC codes are a specific type of LDPC codes that provide structured versions of the general, unstructured LDPC codes [51, 52, 53]. These codes are defined by their PCM H, which consists of a pattern of sparse 1s and 0s that specify the code's error-correcting properties. The PCM of a QC LDPC code, denoted by H, has dimension $M \times N$. Below is an example of an irregular PCM H with $N = 18$ and $M = 9$:

$$H = \left[\begin{array}{cccccccccccccccccc}
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0\\
0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&0\\
0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&1&0&0\\
0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&1\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0\\
1&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&1&0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0
\end{array}\right]. \quad (2.26)$$

In this example, the average row degree d_c is 4.33, and the average column degree d_v is 2.17. When decoding LDPC codes, the computations are based on the rows and columns of the PCM. For each parity-check equation (row), the calculations use inputs from the set of codeword bits (columns) connected by non-zero elements in H. These computations update LLRs, which provide soft information about the likelihood of each bit being 0 or 1. The process iterates until convergence.

Base Matrix Representation

A QC LDPC code can be represented by a base matrix or exponent matrix H_b, where each element of H_b corresponds to a submatrix in H with dimensions $z \times z$ [54, 55, 56]. The base matrix H_b has dimensions $m_b \times n_b$, where $n_b \times z = N$ and $m_b \times z = M$. The coding rate is $R = 1 - \frac{m_b}{n_b}$. For instance, the base matrix H_b for $z = 3$, $n_b = 6$, and $m_b = 3$ can be:

$$H_b = \begin{bmatrix} -1 & 1 & -1 & 0 & 2 & 1 \\ -1 & 2 & 0 & 0 & -1 & 0 \\ 2 & -1 & 0 & -1 & 2 & 1 \end{bmatrix}. \quad (2.27)$$

In H_b: -1 indicates a null submatrix, Non-negative values represent the shift of the identity matrix.

The semi-structured nature of QC LDPC codes results in several beneficial properties:

- All columns and rows within a block-row or block-column have the same degree.
- No two rows or columns within a block have non-zero elements in the same positions.
- Shift values of submatrices allow cyclically shifting LLRs for efficient message-passing during decoding.

Quasi-Cyclic LDPC codes provide a structured approach to LDPC coding, offering advantages in hardware implementation and decoding efficiency. Their structured PCMs and shift properties enable effective and reliable error correction, making them well-suited for modern communication systems.

2.3 Protograph structure of LDPC Codes in 5G NR

A protograph is a small bipartite graph that can be expanded to form a larger Tanner graph. This structure aids in designing LDPC codes with desirable characteristics, such as low error floors and good convergence behavior. 5G LDPC codes support various code rates through rate matching techniques. These techniques include puncturing (removing specific bits) and repetition (duplicating certain bits).

2.3.1 Protograph structure

A protograph is a fundamental component of LDPC codes and is crucial for the 5G communication standard [57, 58, 59]. It is a small bipartite graph consisting of two sets of nodes: variable nodes (VNs) and check nodes (CNs). The edges between these nodes define the relationships and interactions essential for error correction. Key features of protographs in 5G LDPC codes include:

- Simplicity and Regularity: Protographs offer a simple and consistent method for designing LDPC codes. By replicating and permuting the basic protograph structure, larger LDPC codes can be constructed. These larger codes retain the advantageous properties of the original protograph, ensuring efficiency and reliability.

- Flexibility in Design: Protographs provide significant design flexibility, allowing LDPC codes to be customized to meet various requirements, including different code rates and block lengths. This versatility is particularly important for the diverse and demanding applications of 5G technology.
- Efficient Decoding: LDPC codes derived from protographs are known for their efficient decoding capabilities. The structured nature of protographs supports the use of iterative decoding algorithms, such as the Belief Propagation (BP) algorithm, which are both computationally efficient and effective in correcting errors.

Construction Process of LDPC Codes Using Protographs [60, 61]:

1. Designing the Base Protograph: The process starts with the creation of a base protograph, which includes a small number of variable and check nodes. The edges connecting these nodes are carefully selected to ensure strong error-correcting performance.
2. Lifting the Protograph: The base protograph is then ”lifted” to generate a larger LDPC code. Lifting involves replicating the base protograph multiple times and interconnecting these replicas in a specific pattern. This results in a larger bipartite graph that retains the beneficial properties of the original protograph.
3. Permuting the Edges: To avoid the formation of short cycles in the resulting LDPC code, which can negatively impact performance, the edges in the lifted protograph are permuted. This permutation is crucial for enhancing the code's error-correcting capabilities.

In 5G NR (New Radio), two primary base graphs are used: Base Graph 1 (BG1) and Base Graph 2 (BG2). These base graphs are predefined and are essential for constructing the parity-check matrix used in LDPC coding. The specifications and applications of these base graphs are detailed in the NR standard specification TS 38.212.

- BG1: Designed for larger block lengths $(500 \leq K \leq 8448)$ and higher rates $(1/3 \leq R \leq 8/9)$.

- BG2: Designed for smaller block lengths $(40 \leq K \leq 2560)$ and lower rates $(1/5 \leq R \leq 2/3)$.

Table 2.1: 5G NR LDPC base matrix parameters.

Parameter	**Base Matrix 1**	**Base Matrix 2**
Minimum design code rate	1/3	1/5
Base matrix size	46×68	42×52
Number of systematic columns	22	10
Maximum information block size K	8,448 (= 22×384)	3,840 (= 10×384)
Number of nonzero elements	316	197

2.3.2 Base Graph 1 (BG1): Detailed Explanation

Base Graph 1 (BG1) is integral to the 5G NR standard for channel coding, particularly for LDPC codes. BG1 is optimized for higher code rates (greater than or equal to 1/3) and larger block sizes, ensuring robust performance across various 5G applications, including down-link and up-link shared channels. The structure of BG1 is designed for scalability, allowing it to adapt to different block lengths and code rates required by the system. The base graph's structure is divided into several parts, which can be represented as follows:

$$H_b = \begin{pmatrix} A & E & O \\ B & C & I \end{pmatrix}. \tag{2.28}$$

The submatrices within the base graph have specific roles:

- Sub-matrix A: Represents the information bits,
- Sub-matrix E: Corresponds to a dual-diagonal matrix,
- Sub-matrix O: An all-zero matrix,
- Sub-matrices B and C: Correspond to extension check matrices,
- Sub-matrix I: Represents the identity matrix.

Lifting Size and Shift Coefficients: The shift coefficient designs for both BG1 and BG2 are categorized into eight sets based on the parameter a, used in defining the lifting size $z = a \times 2^j$. Each set of lift sizes supports specific shift sizes $z = a \times 2^j$, with parameters

a and j. The base graph that supports $K_{\max}$ must accommodate the lift sizes as shown in the Table 2.2 and Table 2.3 below:

Table 2.2: Relationship between exponent matrices and sets of lifting size.

Exponent Matrix	Lifting Size Set
Set 1	$z = 2 \times 2^j,\ j = 0,1,2,3,4,5,6,7$
Set 2	$z = 3 \times 2^j,\ j = 0,1,2,3,4,5,6,7$
Set 3	$z = 5 \times 2^j,\ j = 0,1,2,3,4,5,6$
Set 4	$z = 7 \times 2^j,\ j = 0,1,2,3,4,5$
Set 5	$z = 9 \times 2^j,\ j = 0,1,2,3,4,5$
Set 6	$z = 11 \times 2^j,\ j = 0,1,2,3,4,5$
Set 7	$z = 13 \times 2^j,\ j = 0,1,2,3,4$
Set 8	$z = 15 \times 2^j,\ j = 0,1,2,3,4$

Table 2.3: Lift sizes z where $z = a \times 2^j$ for $a \in \{2,3,5,7,9,11,13,15\}$ and $0 \leq j \leq 7$.

j	a							
	2	3	5	7	9	11	13	15
0	2	3	5	7	9	11	13	15
1	4	6	10	14	18	22	26	30
2	8	12	20	28	36	44	52	60
3	16	24	40	56	72	88	104	120
4	32	48	80	112	144	176	208	240
5	64	96	160	224	288	352		
6	128	192	320					
7	256	384						

The values $V_{i,j}$ are obtained from Tables 5.3.2-2 and 5.3.2-3 of the 5G NR standard specification 3GPP TS 38.212 based on the selected BGM and the set index i_{LS}. The shift value $s_{i,j}$ is calculated using the formula $s_{i,j} = f(V_{i,j}, z)$, where $V_{i,j}$ is the shift coefficient of the (i,j) element. The function f is defined as:

$$s_{i,j} = f(V_{i,j}, z) = \begin{cases} -1 & \text{if } V_{i,j} = -1, \\ \mod(V_{i,j}, z) & \text{else.} \end{cases} \tag{2.29}$$

The Base Graph Matrix (BGM) H_b is used to construct the Parity Check Matrix (PCM) H for 5G LDPC codes. The dimensions of H and H_b are $M \times N$ and $m_b \times n_b$, respectively, with $N = n_b \times z$ and $M = m_b \times z$. The lifting size z expands the entries of H_b into H using a $z \times z$ square sub-matrix.

In H_b, entries are expanded in H as follows:

- A '-1' entry in H_b becomes a zero matrix of size $z \times z$ in H.
- A '0' entry in H_b becomes an identity matrix of size $z \times z$ in H.
- A non '-1' entry in H_b becomes a circulant permutation matrix $I(s_{i,j})$ in H, with shift value $s_{i,j}$ ranging from 1 to z.

For instance, a circulant permutation matrix $I(1)$ of size $z \times z$ is represented as:

$$I(1) = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{2.30}$$

The general form of the QC-LDPC parity-check matrix H is expressed as an $m_b \times n_b$ array of $z \times z$ circulants over GF(2):

$$H = \begin{pmatrix} I(s_{1,1}) & I(s_{1,2}) & \cdots & I(s_{1,n_b}) \\ I(s_{2,1}) & Q(s_{2,2}) & \cdots & I(s_{2,n_b}) \\ \vdots & \vdots & \ddots & \vdots \\ I(s_{m_b,1}) & I(s_{m_b,2}) & \cdots & I(s_{m_b,n_b}) \end{pmatrix}. \tag{2.31}$$

Each $I(s_{i,j})$ is a circulant permutation matrix corresponding to the shift value $s_{i,j}$. The Base graph or exponent matrix $E(H)$ of the PCM H is defined by:

$$E(H) = H_b = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,n_b} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,n_b} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m_b,1} & s_{m_b,2} & \cdots & s_{m_b,n_b} \end{pmatrix}. \tag{2.32}$$

Each entry $s_{i,j}$ represents a shift value. The PCM H is constructed by expanding the exponent matrix $E(H)$, known as protograph construction.

2.3.3 Construction of BGM in 5G NR Standard

To construct the BGM H_b for an (N,K) LDPC code with a code rate $R = K/N$, follow these five steps:

1. Select the BGM: According to the 3GPP TS 38.212 specification, it is a need to verify the following conditions:
   - If $K \leq 3824$ and $R \leq 0.67$, then BG2 is chosen.
   - If $K \leq 292$, then BG2 is chosen.
   - If $R \leq 0.25$, then BG2 is chosen.
   - Otherwise, BG1 is chosen.
2. Calculate k_b: According to the 3GPP TS 38.212 specification, the values are as follows:
   - For LDPC BG1, $K_b = 22$.
   - For LDPC BG2, if $K > 640$, then $K_b = 10$.
   - For LDPC BG2, if $560 < K \leq 640$, then $K_b = 9$.
   - For LDPC BG2, if $192 < K \leq 560$, then $K_b = 8$.
   - For LDPC BG2, if $K \leq 192$, then $K_b = 6$.
3. Determine the Expansion Factor or Lifting Size z: Choose the smallest z from Table 5.3.2-1 such that $k_b \times z \geq K$.
4. Select the Set Index i_{LS}: Once z is determined, choose the appropriate shift coefficient matrix set from Table 5.3.2-1.
5. Calculate the BGM Entry Values: Utilize the function $S_{i,j} = f(V_{i,j}, z)$ to determine the entry values $S_{i,j}$ with the modular z operation.

Finally, construct the PCM H by replacing each BGM entry with the corresponding circulant permutation matrix or zero matrix of size $z \times z$.

An example:

To construct the BGM H_{b1} for a (N,K) LDPC code with a message block length $K = 1232$ and a code rate $R = K/N = 1/3$ from given steps:

1. Select the Base Graph Matrix:

   According to the 3GPP TS 38.212 specification, the selection of the BGM depends on specific conditions related to the message block length K and the code rate R. Given that $K = 1232$, which is less than 3824, and $R = 1/3$, which falls between 0.25 and 0.67, the appropriate BGM needs to be identified.

   Moreover, the criteria stipulate that if $K \leq 3824$ and $R \leq 0.67$, BG2 should be considered. However, for our case, $R = 1/3$, which meets the condition of being greater than 0.25 but less than 0.67. Based on this, BG1 is ultimately selected as the Base Graph Matrix for constructing the given LDPC code.

2. Calculate k_b: For LDPC BG1, the value of K_b is 22.

3. Determine the Expansion Factor or Lifting Size z: To find z, we use the condition $k_b \times z \geq K$. Given $K = 1232$ and $K_b = 22$, we calculate z as:

$$z \geq \frac{1232}{22} \approx 56$$

   Thus, $z = 56$ is chosen.

4. Select the Set Index i_{LS}: For $z = 56$, refer the Table 5.3.2-1 in the 3GPP TS 38.212 specification to find the appropriate shift coefficient matrix set index. The set index i_{LS} for $z = 56$ is 3.

5. Calculate the BGM Entry Values: Using the function $S_{i,j} = f(V_{i,j}, z)$, where $f(V_{i,j}, z)$ is a modular operation, we determine the entry values $S_{i,j}$.

To construct the PCM H, replace each entry $S_{i,j}$ in the BGM with the corresponding circulant permutation matrix or a zero matrix of size $z \times z$.

Starting from Step 5, the first entry $h_{0,0}$ of H_{b1} is determined using the equation mod $(V_{0,0}, z)$, where mod $(223, 56) = 55$. The selected value $V_{0,0} = 223$ is obtained from Table 5.3.2-2 [19], corresponding to row index $i = 0$, column index $j = 0$, under set index $i_{LS} = 3$. Similarly, the remaining entries are calculated using the function (2.29) and the $V_{i,j}$

values derived from Table 5.3.2-2 [19], based on their respective row and column indices. Table 2.4 displays the calculated values for a few entries of H_{b1}. The constructed H_{b1} has dimensions of 46×68, and the corresponding BGM H_{b1} is shown in Table 2.6.

Table 2.4: Calculation of the entry values of H_{b1}.

Entry of H_{b1}	$V_{i,j}$	$mod(V_{i,j}, z_c)$	Corresponding values
$h_{0,0}$	$V_{0,0} = 223$	$mod(223, 56)$	55
$h_{0,1}$	$V_{0,1} = 16$	$mod(16, 56)$	16
$h_{0,2}$	$V_{0,2} = 94$	$mod(94, 56)$	38
$h_{0,3}$	$V_{0,3} = 91$	$mod(91, 56)$	35
$h_{0,4}$	$V_{0,4} = -1$	$mod(-1, 56)$	-1
$h_{0,5}$	$V_{0,5} = 74$	$mod(74, 56)$	18
$h_{0,6}$	$V_{0,6} = 10$	$mod(10, 56)$	10
$h_{0,7}$	$V_{0,7} = -1$	$mod(-1, 56)$	-1
$h_{0,8}$	$V_{0,8} = -1$	$mod(-1, 56)$	-1
$h_{0,9}$	$V_{0,9} = 0$	$mod(0, 56)$	0

To obtain the desired information lengths and facilitate rate adaptation in 5G NR QC-LDPC codes, a process involving shortening and puncturing is performed. The properties of BGM1 are detailed in Table 2.5.

Table 2.5: Parameters of BGM1.

Characteristics	BGM1 (H_{b1})
Number block columns (n_b)	68
Number block rows (m_b)	46
Number edges	316
Column weights (w_v)	1 to 30
Row weights (w_c)	3 to 19
Base code rate	1/3

2.4 Rate Matching Operation of 5G LDPC Codes

Rate matching in 5G LDPC codes is a critical process that adjusts the encoded data rate to align with the required transmission rate [62, 63]. This process is essential for adapting the LDPC encoded data to the varying bandwidth and data rate requirements of 5G communication systems. The primary operations involved in rate matching are puncturing, shortening, and repetition. These methods ensure that the transmitted code fits the available channel resources effectively.

Table 2.6: Base Graph Matrix.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	.....	67
0	55	16	38	35	-1	18	10	-1	-1	0	37	48	21	47	-1	14	14	-1	29	30	48	25	1	0	-1	-1	-1	.....	-1
1	29	-1	45	39	46	7	-1	45	21	31	-1	38	37	-1	23	9	6	26	-1	31	-1	19	0	0	0	-1	-1	.....	-1
2	39	35	31	-1	8	12	18	39	41	9	14	-1	-1	21	46	21	-1	30	5	55	34	-1	-1	-1	0	0	-1	.....	-1
3	33	18	-1	53	5	-1	45	30	16	-1	34	43	45	35	13	-1	40	18	43	-1	30	46	1	-1	-1	0	-1	.....	-1
4	2	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	.....	-1
5	52	3	-1	30	-1	-1	-1	-1	-1	-1	-1	-1	24	-1	-1	-1	14	-1	-1	-1	-1	18	41	-1	-1	-1	-1	.....	-1
6	46	-1	-1	-1	-1	-1	7	-1	-1	-1	-1	21	-1	7	-1	-1	-1	51	24	-1	4	-1	-1	-1	-1	-1	-1	.....	-1
7	17	20	-1	-1	48	-1	-1	44	38	-1	-1	-1	-1	-1	46	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
8	33	39	-1	4	-1	-1	-1	-1	-1	-1	-1	-1	49	-1	-1	-1	36	-1	-1	39	-1	2	44	-1	33	-1	-1	.....	-1
9	9	37	-1	-1	-1	-1	-1	-1	-1	-1	45	49	-1	33	-1	-1	-1	17	53	-1	50	-1	-1	-1	-1	-1	-1	.....	-1
10	-1	26	53	-1	6	-1	-1	19	26	-1	-1	-1	-1	-1	47	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
11	52	11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	35	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
12	30	7	-1	-1	-1	-1	-1	-1	-1	-1	24	3	-1	28	-1	-1	-1	-1	14	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
13	25	-1	-1	0	-1	-1	-1	16	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	49	-1	-1	22	-1	-1	-1	.....	-1
14	14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	7	-1	-1	43	23	51	-1	-1	-1	43	-1	-1	-1	-1	-1	.....	-1
15	34	8	-1	-1	-1	-1	-1	-1	-1	-1	19	-1	-1	41	-1	-1	-1	-1	41	-1	-1	-1	-1	-1	-1	25	-1	.....	-1
16	-1	42	-1	52	-1	-1	-1	-1	-1	-1	-1	43	-1	-1	-1	-1	-1	-1	-1	-1	21	-1	45	-1	-1	-1	-1	.....	-1
17	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	54	-1	32	7	-1	-1	-1	4	-1	-1	-1	-1	-1	.....	-1
18	-1	31	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	54	32	-1	-1	-1	-1	31	18	-1	-1	-1	-1	-1	-1	.....	-1
19	8	6	-1	-1	-1	-1	-1	47	30	-1	8	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
20	49	-1	-1	42	-1	-1	-1	-1	-1	9	-1	46	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	15	-1	-1	-1	-1	.....	-1
21	-1	24	-1	-1	-1	19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	52	-1	-1	-1	50	50	-1	-1	-1	-1	-1	.....	-1
22	53	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	3	-1	-1	-1	36	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
23	-1	32	35	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	10	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
24	49	-1	-1	45	8	-1	-1	-1	-1	-1	-1	25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	12	-1	-1	-1	-1	.....	-1
25	-1	1	-1	-1	-1	-1	54	9	-1	-1	-1	-1	-1	-1	25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
26	51	-1	8	-1	44	-1	-1	-1	-1	15	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
27	-1	40	-1	-1	-1	-1	29	-1	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
28	34	-1	-1	-1	41	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	49	-1	2	-1	-1	-1	-1	-1	.....	-1
29	-1	38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	11	-1	-1	-1	53	-1	-1	2	-1	-1	-1	12	-1	.....	-1
30	34	-1	-1	-1	-1	-1	-1	-1	-1	-1	18	-1	-1	42	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
31	-1	7	-1	-1	-1	-1	-1	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9	-1	-1	16	-1	.....	-1
32	24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	30	-1	-1	.....	-1
33	-1	2	49	-1	-1	-1	-1	-1	-1	-1	-1	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	25	-1	-1	-1	-1	-1	.....	-1
34	26	-1	-1	-1	-1	-1	-1	18	-1	-1	-1	-1	-1	-1	-1	-1	-1	38	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
35	-1	24	-1	-1	-1	-1	5	-1	-1	-1	-1	-1	26	-1	-1	-1	-1	-1	-1	-1	-1	-1	19	-1	-1	-1	-1	.....	-1
36	54	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	45	0	-1	-1	6	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
37	-1	25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	27	-1	-1	-1	-1	-1	-1	-1	-1	-1	26	-1	-1	-1	.....	-1
38	11	-1	-1	-1	-1	-1	-1	-1	-1	34	17	-1	10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
39	-1	12	-1	21	-1	-1	-1	49	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	-1	-1	-1	.....	-1
40	11	-1	-1	-1	-1	-1	-1	-1	45	-1	-1	-1	-1	-1	-1	-1	-1	40	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
41	-1	23	-1	47	-1	-1	-1	-1	-1	4	-1	-1	-1	-1	-1	-1	-1	-1	55	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
42	2	-1	-1	-1	35	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	22	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1
43	-1	38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	22	-1	22	-1	-1	-1	-1	-1	-1	49	-1	.....	-1
44	28	-1	-1	-1	-1	-1	-1	4	-1	9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	12	-1	-1	-1	-1	.....	-1
45	-1	16	-1	-1	-1	-1	9	-1	-1	-1	29	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	.....	-1

Key Operations in Rate Matching:

1. Puncturing: Puncturing involves removing certain encoded bits from the LDPC codeword before transmission, effectively increasing the code rate by reducing the codeword length [64, 65, 66]. This is done by following a predefined puncturing pattern that specifies which bits to omit. For instance, given an original codeword $[c_1, c_2, c_3, c_4, c_5, c_6]$, if the pattern dictates removing c_2 and c_5, the transmitted codeword becomes $[c_1, c_3, c_4, c_6]$. This selective omission helps the encoded data fit

within the available transmission resources while maintaining sufficient error correction performance.

2. Shortening: Shortening sets specific information bits to zero and excludes them from transmission, thereby reducing the length of the codeword and adjusting the code rate [67, 68]. This involves following a predefined shortening pattern to determine which bits to set to zero. For example, if an information sequence is $[m_1, m_2, m_3, m_4, m_5, m_6]$ and the pattern specifies shortening m_3 and m_6, the sequence becomes $[m_1, m_2, 0, m_4, m_5, 0]$. This method ensures the codeword length matches the desired code rate, optimizing bandwidth usage while preserving the integrity of the data.

3. Repetition: Repetition involves duplicating certain bits of the encoded data to achieve the desired codeword length [69, 70, 71]. This operation is used when the encoded data is shorter than required, filling up the available transmission slots. Following a specified repetition pattern, certain bits are repeated. For instance, if the original codeword is $[c_1, c_2, c_3]$ and the pattern indicates repeating all bits, the transmitted codeword becomes $[c_1, c_2, c_3, c_1, c_2, c_3]$. Repetition enhances the robustness of the transmitted signal, improving its resilience against errors.

4. Combining Operations: Rate matching often involves a combination of puncturing, shortening, and repetition to achieve the desired code rate. For example, if an LDPC code has an original codeword length of 12 bits, after puncturing 2 bits and shortening 2 bits, repetition might be applied to fill the remaining slots to meet the target length.

Detailed Example:

- Original Codeword: $[c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}]$
- Puncturing Pattern: Remove c_2 and c_5, resulting in $[c_1, c_3, c_4, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}]$.
- Shortening Pattern: Shorten c_8 and c_{12}, resulting in $[c_1, c_3, c_4, c_6, c_7, 0, c_9, c_{10}, c_{11}, 0]$.

- Repetition Pattern: Repeat c_1 and c_3 to fill up the codeword, resulting in $[c_1, c_3, c_4, c_6, c_7, 0, c_9, c_{10}, c_{11}, 0, c_1, c_3]$.

By following these steps, the rate matching process ensures that LDPC encoded data is efficiently transmitted over the 5G network, adapting to varying channel conditions and resource allocations.

2.5 Conclusion

This chapter thoroughly examines the key components of digital communication systems, with a particular focus on LDPC codes and their implementation in 5G NR. Channel coding and the specifics of LDPC codes were then delved into, emphasizing their structure, error-correcting capabilities, and the advantages of their sparse PCMs. The concept of Tanner graphs, which provide a graphical representation of LDPC codes, was explored to facilitate the understanding and implementation of decoding algorithms.

The encoding and decoding processes of LDPC codes were discussed in detail, including the systematic encoding procedure and the iterative decoding algorithms such as the SPA and the MSA. These algorithms leverage the sparse structure of LDPC codes to efficiently correct errors and reconstruct the original information. Additionally, the quasi-cyclic nature of LDPC codes was covered, explaining how their structured approach benefits hardware implementation and decoding efficiency. The protograph structure, crucial for 5G LDPC codes, was introduced, showcasing its simplicity, regularity, and flexibility in design, which are essential for the diverse requirements of 5G technology.

The construction of BGM in 5G NR was detailed, providing a step-by-step guide on selecting the appropriate base graph, determining the expansion factor, and calculating the BGM entry values. This section included a practical example to illustrate the construction process. Finally, rate matching operations in 5G LDPC codes were explored, discussing the techniques of puncturing, shortening, and repetition. These methods are vital for adjusting the encoded data rate to fit the available transmission resources, ensuring efficient use of the radio spectrum and optimizing data transmission.

In conclusion, this chapter provides a comprehensive overview of LDPC codes in the context of 5G NR enabled IoT networks, highlighting their importance in modern communication systems. The detailed explanations of encoding, decoding, and rate matching processes offer valuable insights into the robust error correction and adaptability that LDPC codes bring to 5G networks. This understanding is crucial for anyone involved in the design, implementation, and optimization of next-generation communication systems.

3

A FPGA-based fixed- point Decoder for 5G LDPC codes

3.1 Introduction

Technical advancements in Enhanced Mobile Broadband (eMBB) for IoT applications within 5G networks rely heavily on LDPC decoding. Maintaining the performance and reliability of communication in complex IoT environments hinges on efficient data transmission and dependability, which LDPC codes provide. Effective error correction ensures improved signal reception and higher quality of service (QoS), crucial for eMBB applications demanding high data rates and consistent performance via LDPC decoding. LDPC codes, prevalent in 5G networks, offer a robust error-correcting mechanism. Their de-

sign flexibility enhances adaptability to varying data requirements and constraints across a spectrum of IoT applications, which is especially significant given the range of controlled environments (e.g., smart factories) to highly unpredictable ones (e.g., remote sensing) [72]. Flexible coding rates enable systems to adjust according to different environmental conditions, thus maximizing reliability and performance. The 5G NR specifications outline a range of LDPC coding rates and block lengths to ensure compatibility across numerous devices and network configurations [73].

The 5G QC LDPC code's PCM H is defined and represented by its BGM H_b. According to the 5G NR specifications, H_{b1} supports a variety of coding rates from 1/3 to 8/9, while H_{b2} supports code rates from 1/5 to 2/3. Each base graph matrix has 51 defined PCMs corresponding to sub-block sizes or lifting size z values between 2 and 384, resulting in information blocks of varying sizes. For decoding messages associated with different PCMs of various code rates at runtime, it is advantageous for the decoder to possess runtime flexibility [74, 75, 76]. Consequently, the decoder can choose from a range of LDPC coding rates supported by various PCMs. This flexibility is beneficial in practical commercial applications, such as automatically adjusting the code rate to enhance performance based on channel conditions [22, 77], without needing the additional time for FPGA reprogramming. Avoiding FPGA re-synthesis for performance testing across multiple code rates is another potential application. Hence, it is preferable to design a decoder capable of runtime flexibility [78, 79], encompassing a broad spectrum of LDPC code rates within the 5G standard specifications.

3.2 Overview of Key Concepts

3.2.1 Enhanced Mobile Broadband (eMBB)

eMBB stands as one of the key applications of 5G technology [80] , it addresses the increasing need for high-bandwidth applications [81], including:

- High-definition video streaming: Services like 4K/8K and virtual reality (VR) content necessitate substantial bandwidth and minimal latency to ensure a smooth user

experience.

- Broadband access in all areas: eMBB aims to provide high-speed internet access universally, including urban and rural locales, potentially serving as a replacement for traditional wired broadband.
- Connectivity in high-density environments: By boosting traffic capacity and speed, eMBB enhances connectivity in crowded places such as public events and venues, improving overall network performance.

Base Graph 1 is frequently employed in eMBB contexts to manage high data rates and bandwidth, enabling network operators to optimize performance according to specific conditions and service demands.

3.2.2 Base graph matrix (BGM) of 5G NR LDPC code

Table 3.1: An Example of the Constructed BGM-1 and its Parameters.

Parameters of BGM-1	Values
Base code rate	1/3
Edges	316
Lifting size	384
Block columns (n_b)	68
Column weights (w_v)	1 to 30
Block rows (m_b)	46
Row weights (w_c)	3 to 19

Table 3.2: The Parameters of Code rate set of N = 3808.

Code rate (R)	1/3	2/5	1/2	2/3	3/4	5/6	8/9
Sub matrix (z)	56	56	56	56	56	56	56
No. of block columns (n_b)	68	68	68	68	68	68	68
No. of block rows (m_b)	46	41	34	24	17	12	9
No.of. columns (N)	3808	3808	3808	3808	3808	3808	3808
No.of. rows (M)	2576	2296	1904	1344	952	672	840
Information bits (K)	1232	1512	1904	2464	2856	3136	2968

The BGM H_b is used to represent and construct the PCM H for the 5G LDPC code. The dimensions of H and H_b are $M \times N$ and $m_b \times n_b$, respectively. These dimensions are linked by $N = n_b \times z$ and $M = m_b \times z$. The lifting or sub-matrix size z is used to expand the

Table 3.3: The Parameters of 1/3- rate (3808, 1232) LDPC code.

Parameter	value
Expansion factor (or) lifting size	56
Dimension of Base matrix (H_b)	46×68
Number of block-rows (m_b)	46
Number of block-columns (n_b)	68
Dimension of PCM H	2576×3808
Check node degree (d_c)	3 to 19
Average CN degree	6.86
VN degree (d_v)	1 to 30
Average VN degree	4.52

entries of H_b into H by employing a $z \times z$ square sub-matrix. There are three types of entry values in H_b: '-1', '0', and non '-1' values, which are processed as follows:

- Zero Matrix Replacement: A '-1' entry in H_b is replaced with a zero matrix of size $z \times z$ in H.
- Identity Matrix Replacement: A '0' entry in H_b is replaced with an identity matrix of size $z \times z$ in H.
- Circulant Permutation Matrix Replacement: A non '-1' entry in H_b is replaced with a circulant permutation matrix $I(s_{i,j})$ in H.

The non '-1' entry in H_b is called the shift value $s_{i,j}$, which ranges from 1 to z. The indices of the entries in H_b are denoted by i and j, representing the row and column positions, respectively. Each row of the identity matrix is right-shifted by $s_{i,j}$ positions to form the sub-matrix $I(s_{i,j})$.

Based on the chosen BGM and the set index i_{LS}, the shift value $s_{i,j}$ is obtained from Tables 5.3.2-2 and 5.3.2-3 of the 5G-NR standards 3GPP TS 38.212 [19]. The BGM-1 has dimensions of 46×68. For this investigation, a message block length of 1232, a code word length of 3808, and a code rate of 1/3 are used to generate H_b and H for the LDPC code. The parameters of BGM-1 as per the 5G standard and their values are detailed in Table 3.1. The parameters of the constructed H_b and H for the LDPC code rate set are presented in Table 3.2 and Table 3.3.

3.2.3 Normalized Min Sum (NMS) Algorithm

The NMS algorithm [82, 83] is an improved variant of the MSA. It enhances the accuracy of decoding by incorporating a normalization factor that adjusts the messages exchanged between CNs and VNs, thereby reducing the likelihood of overestimating the reliability of these messages. The normalization factor α of check node update is typically determined through empirical studies and simulations. Common values range from 0.5 to 0.8. Lower values of α (closer to 0) decrease the impact of the messages, potentially slowing convergence. Higher values of α (closer to 1) make the NMS algorithm [84, 85] resemble the basic MSA, with possible overestimation issues.

3.3 Suggested Design Framework

This chapter provides a comprehensive evaluation of the FPGA-based LDPC decoder architecture, highlighting its inherent flexibility in both run-time and design time for any coding rate set in the 5G standard. The proposed design is a decoder framework capable of decoding messages from any combination of QC-LDPC PCMs compliant with the 5G standard. Essentially, the architecture is partially parallel [86], which offers considerable flexibility during the design phase, enabling a balance between processing throughput and the required hardware resources. As illustrated in Figure 3.1, the architecture comprises various fundamental modules, including the Variable Node Decoder (VND), Check Node Decoder (CND), a routing network connecting the VND and CND modules, Variable Node Memory (VMEM), Check Node Memory (CMEM), and Read-Only Memory (ROM) for the BGM. The following sections will delve into these design modules in detail.

3.3.1 Parallelism in 5G NR LDPC Code Decoders

The degree of parallelism in the proposed decoder architecture is a crucial factor influencing overall decoding speed and efficiency. This architecture leverages parallelism based on the lifting size z of the sub-matrices in the target QC PCMs, using a modified flooding schedule [87]. This schedule processes every VN and CN once per iteration, ensuring op-

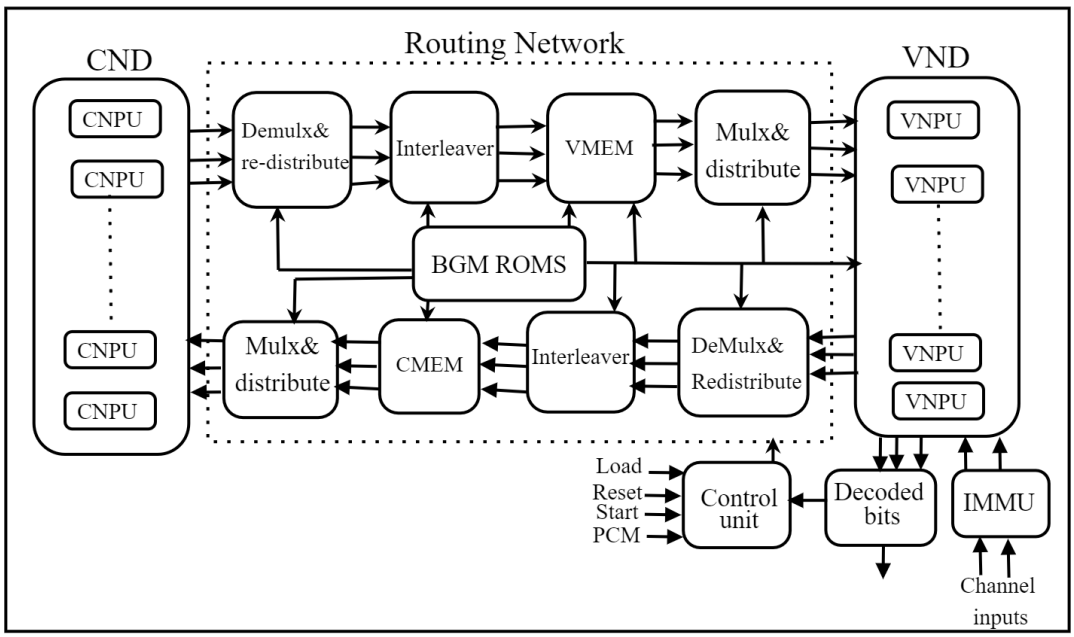


Figure 3.1: A Top level block diagram.

timal resource utilization. The decoder is composed of two primary components: the VND and the CND.

VN Parallelism Factor (Γ_v)

The VND consists of Γ_v Variable Node Processing Units (VNPUs). Key parameters determining the parallelism factor include:

- Z : The largest possible submatrix size from the code rate supported set PCMs.
- F : A parallelism reduction factor, F, must satisfy $1 \leq F \leq Z$. F can be chosen during the design phase to balance processing performance and hardware resource utilization according to specific requirements.

The VN parallelism factor is calculated as follows:

$$\Gamma_v = \left\lceil \frac{Z}{F} \right\rceil . \tag{3.1}$$

This equation ensures that Γ_v VNPUs are available to process Γ_v columns of the PCM simultaneously in each clock cycle.

Impact of Parallelism Reduction Factor F with $Z = 56$:

- Higher F Value: When $F = 56$, the parallelism factor Γ_v becomes 1, making the

decoder architecture fully serial.

- Moderate F Value: When $F = 4$, parallelism is reduced by a factor of 4. Only $\Gamma_v = 14$ VNPUs are utilized, processing 14 columns of the PCM per clock cycle. Thus, the complete block-column of 56 columns requires 4 clock cycles for processing, balancing hardware resource use and processing speed.
- Lower F Value: When $F = 1$, there is no reduction in parallelism. All $\Gamma_v = 56$ VNPUs are utilized to process 56 columns of the PCM simultaneously in one clock cycle, optimizing processing speed by allowing the entire block-column to be processed in a single clock cycle.

CN Parallelism Factor (Γ_c)

Similarly, the CND comprises Γ_c Check Node Processing Units (CNPUs), where Γ_c is less than or equal to Γ_v. The number of CNPUs is optimized by considering $Y = \frac{n_b}{m_b}$, the ratio of the number of block-columns n_b to the number of block-rows m_b. The parameter Y_{min}, which denotes the minimum value of Y among a given set of supported PCMs, influences the reduction in CNPUs. When Y_{min} is greater than or equal to 2, the number of CNPUs required in the CND can be reduced as follows:

$$\Gamma_c = \left\lceil \frac{\Gamma_v}{Y_{min}} \right\rceil. \tag{3.2}$$

Each CNPU processes one row from any of the supported PCMs in a single clock cycle, denoted as $t_c = 1$. The proposed decoder operates by running the Γ_c CNPUs and Γ_v VNPUs in parallel. During each decoding iteration, all CNs and VNs within the factor graph are activated simultaneously. Extensive simulations have demonstrated that processing the CND and VND concurrently does not significantly impact the decoder's BER performance compared to more traditional, serial decoding schedules. This parallel processing approach effectively maintains the decoder's efficiency without compromising its error-correcting capabilities.

3.3.2 ROM Configuration in the Architecture

The proposed design achieves run-time flexibility by utilizing multiple Read-Only Memories (ROM) blocks, which store the values of each base PCM or BGM in a hardware-optimized format. These ROMs play a critical role in the datapaths of both the VND and CND, facilitating the routing and shifting of a priori and extrinsic LLRs between memory units and NPUs. The architecture includes three distinct types of ROMs, each serving a unique function in BGM processing:

1. LOCATION ROM: This ROM stores the indices of the non-null sub-matrices of the BGM. When the VND or CND requires the position of a non-null sub-matrix, it retrieves the relevant index from the LOCATION ROM. This index then guides the data flow, ensuring that operations target the correct sub-matrices during decoding.
2. ADJUST ROM: For each non-null sub-matrix indexed in the LOCATION ROM, the corresponding shift value is stored in the ADJUST ROM. During processing, the VND or CND retrieves the shift value for a specific non-null sub-matrix. These shift values are used to adjust the positions of the elements within the sub-matrix, usually involving a cyclic shift operation. This alignment is essential for the subsequent processing stages.
3. STATUS ROM: This ROM stores a binary value for each entry in the LOCATION and ADJUST ROMs. A value of '1' indicates that the entry corresponds to a non-null sub-matrix, while '0' denotes a null sub-matrix. Before processing a sub-matrix, the VND or CND checks the STATUS ROM to verify if the current entry is non-null. If the STATUS ROM indicates a non-null sub-matrix (value of '1'), the processing continues using data from the LOCATION and ADJUST ROMs. If the indicator is '0', the system skips processing for that entry, optimizing resource usage and computational efficiency.

This configuration allows the architecture to dynamically adapt to various PCMs by accurately controlling the data flow and processing within the VND and CND. By separating the indices, shift values, and status indicators into distinct ROMs, the system ensures effi-

cient management of each aspect of the PCM data, enabling quick and reliable adjustments during the decoding process. The hardware-optimized format of the ROMs enhances the overall efficiency and speed of the LDPC decoding process, making it well-suited for the high demands of 5G communication systems.

3.3.3 Decoding Schedule in the proposed decoder

The decoding schedule in the proposed architecture is designed to enhance parallelism and reduce processing latency. Here is a detailed, step-by-step description of the decoding schedule:

1. Initialization: The decoder begins by initializing the processing units and memory banks. The BGM and other necessary parameters are loaded into their respective ROMs.
2. Parallel Processing Setup: The VNPUs and CNPUs are configured to operate in parallel. The degree of parallelism Γ_v for VNPUs and Γ_c for CNPUs is determined based on the maximum submatrix size Z and the parallelism reduction factor F.
3. Block-Column Processing: The VND processes columns within a block-column for t_{vb} clock cycles before moving to the next block-column. Each VNPU handles one column of the PCM per clock cycle ($t_v = 1$). For instance, if $\Gamma_v = 56$ and $F = 1$, the VND can process a complete block-column of 56 columns within $t_{vb} = 1$ clock cycle.
4. Block-Row Processing: The CND processes rows within a block-row for t_{cb} clock cycles before proceeding to the next block-row. Each CNPU manages one row of the PCM per clock cycle ($t_c = 1$).
5. Simultaneous Activation: Both the VND and CND operate concurrently, processing block-columns and block-rows in parallel. This simultaneous activation ensures efficient processing of both VNs and CNs within each decoding iteration.
6. Iteration Completion: An iteration of the decoding process is completed once all block-columns and block-rows have been processed. The total number of clock cy-

cles required per iteration is given by $t_i = t_{vb} \times n_b$.

7. Repetition: The process is repeated for the necessary number of iterations to ensure accurate decoding. Early stopping mechanisms can be employed to terminate the decoding process once a valid codeword is detected.

3.3.4 MEMORY ORGANIZATION

The architecture employs FPGA Block RAMs (BRAMs) to store extrinsic LLRs, achieving efficient memory management and high performance. This is done by dividing memory into VMEM and CMEM sections, managing data transfers, and optimizing BRAM usage through a non-linear storage pattern [76]. This design effectively supports high bandwidth and ensures compatibility with various PCM configurations. The FPGA's built-in BRAMs are used to store extrinsic LLRs calculated by the VND until needed by the CND, and vice versa. This configuration ensures efficient data storage and retrieval, which is crucial for maintaining high processing speeds. All LLRs are encoded as $W = 4$-bit two's complement integers.

The memory is divided into two main sections: VMEM, which stores LLRs for the VND to read and process, and CMEM, which stores LLRs for the CND to read and process. During each clock cycle, the VND reads up to $\Gamma_v \times D_v$ W-bit values from VMEM and writes an equal number to CMEM. Concurrently, CMEM reads up to $\Gamma_c \times D_c$ W-bit values and writes the same amount back to VMEM. This continuous data transfer ensures that both the VND and CND have timely access to the required LLRs.

To accommodate various PCM values, VMEM and CMEM are organized to handle the maximum values of n_b and m_b within the supported set of PCMs. Both memory sections consist of $\mathscr{N}_{\mathscr{B}}$ BRAMs, each containing $\mathscr{M}_{\mathscr{B}} \times F$ address locations, with each address holding $\Gamma_v \times W$ bit values. This setup is essential because the FPGA's BRAM resources are fewer than the rows and columns in a PCM, necessitating the grouping of multiple LLRs into one BRAM word.

The number of dual-port BRAMs required at any given time equals the maximum number of PCM sub-matrices, denoted as D_{max}. Therefore, $\mathscr{N}_{\mathscr{B}}$ BRAMs are used in both

VMEM and CMEM, ensuring compatibility with any QC PCM, as $\mathscr{N}_{\mathscr{B}}$ is greater than $\mathscr{M}_{\mathscr{B}}$.

Although using fewer BRAMs for PCMs with $D_c < \mathscr{N}_{\mathscr{B}}$ could theoretically be achieved by implementing switchable routing networks, this approach would increase logic resource consumption and path lengths, reducing clock frequency and throughput. Hence, the design opts for $\mathscr{N}_{\mathscr{B}}$ BRAMs in each memory section to minimize hardware resource use and critical path length.

A non-linear storage pattern is used to avoid requiring more than one address from any BRAM simultaneously. LLRs for a block-row C_{BI} are stored in address I of all $\mathscr{N}_{\mathscr{B}}$ BRAMs. LLRs for a block-column V_{BJ} are stored in addresses 1 to $\mathscr{M}_{\mathscr{B}}$ of BRAMs $(J, J+1, ..., J+(\mathscr{M}_{\mathscr{B}}-1)) \mod \mathscr{N}_{\mathscr{B}}$. When LLRs are read from VMEM, the output includes LLRs from all $\mathscr{N}_{\mathscr{B}}$ BRAMs, even though only $\mathscr{M}_{\mathscr{B}}$ BRAMs contain the LLRs for the active block-column. The VND datapath then selects the appropriate BRAM outputs for the active block-column.

3.3.5 Datapaths of the Proposed LDPC Decoder Architecture

The datapath in the proposed architecture for an FPGA-based LDPC decoder plays a crucial role in ensuring efficient and high-performance decoding. The datapath comprises various components that work together to execute the decoding algorithm by processing LLRs through the VND and CND. Key Components of the Datapath:

- VND Datapath: It is responsible for updating the LLRs corresponding to variable nodes. It reads the LLRs from VMEM, processes them, and then writes the updated LLRs to CMEM.

- CND Datapath: It is responsible for updating the LLRs corresponding to check nodes. It reads the LLRs from CMEM, processes them, and then writes the updated LLRs to VMEM.

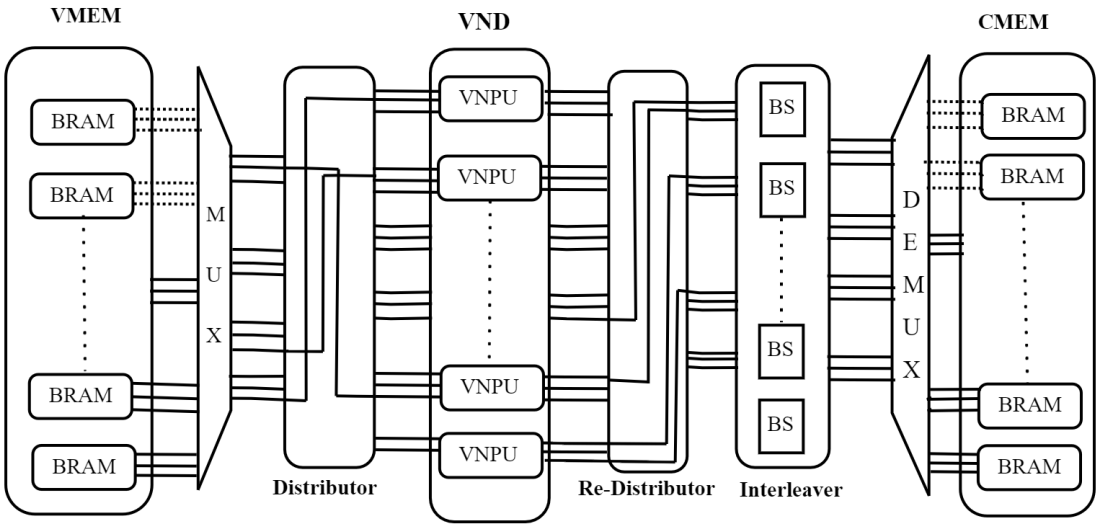


Figure 3.2: An example of the VND Data Path.

3.3.5.1 VND Data Path

As shown in Figure 3.2, the VND data path processes data through multiple stages, updating variable nodes based on received messages before passing them to the next stage for check node processing. The detailed data flow steps are as follows:

The VND data path begins by receiving input from $\mathscr{N}_{\mathscr{B}}$ groups of Γ_v W-bit LLRs stored in VMEM. Each group includes multiple columns of LLR values, each column corresponding to a block-column of the PCM being processed. These LLR groups are routed to Γ_v VNPUs, with each VNPU having input channels for the highest column degree D_V. The routing mechanism ensures that each VNPU receives the correct LLR inputs from VMEM, involving complex wiring to match the columns of LLRs to the appropriate VNPU inputs.

Within the VND, each VNPU processes its respective inputs, producing Γ_v groups of D_V outputs. These outputs are then consolidated, allowing the VND to handle multiple processing tasks simultaneously, maintaining high throughput and efficiency. After processing, the outputs from the VND are rerouted back into $\mathscr{N}_{\mathscr{B}}$ groups of Γ_v messages for storage in CMEM. This redistribution process requires careful coordination to ensure each output message is correctly placed in the appropriate memory location within CMEM.

The VMEM and CMEM memory organization ensures only relevant data is processed.

Guided by the LOCATION ROM, the multiplexer (Mux) isolates non-null submatrices, allowing only meaningful data to pass through. If a submatrix is null (indicated by a '0' in the STATUS ROM), the Mux outputs a value that effectively "turns off" the input, preventing unnecessary data from entering the VND.

Processed LLRs are allocated to Γ_v groups of D_V LLRs by the Distributor. This allocation balances the load across multiple VNPUs, ensuring each VNPU receives the correct subset of LLRs for further processing. Each VNPU processes its allocated LLRs, performing necessary computations such as error correction and parity checks according to the decoding algorithm.

After the VNPUs process the messages, the Re-distributor re-organizes them, ensuring the processed messages are correctly reassembled into their original D_V groups, maintaining data integrity as it moves to the next stage. Before the messages are written into CMEM, they undergo cyclic left-shifting by the D_V programmable Barrel Shifters (BSs) within the Interleaver. The shift values are stored in the SHIFTS ROM to ensure the data is correctly aligned and distributed within CMEM.

The final step involves a programmable de-multiplexer (De-mux), which reverses the initial multiplexer operation. Using indices from the ADJUST ROM, the De-mux places the output values at the correct input ports for the D_V BRAMs within CMEM, ensuring each piece of data is stored in its designated location, ready for the next decoding cycle.

3.3.5.2 CND Datapath

The CND data path in the decoder is essential for updating check node messages, ensuring precise error correction. It operates similarly to the VND data path but involves additional complexities due to the nature of check nodes, as illustrated in Figure 3.3. The CND processes Γ_v rows over $t_{cp} = Y_{min}$ clock cycles using Γ_c CNPUs, with D-type Flip-Flops (DFFs) and extra multiplexers between the CMEM and CND. De-multiplexers and additional DFFs are employed between the CND and VMEM to ensure all Γ_v messages are ready for writing back into VMEM within the same clock cycle. This involves splitting the outputs from the CND and routing them correctly, which is vital for maintaining data

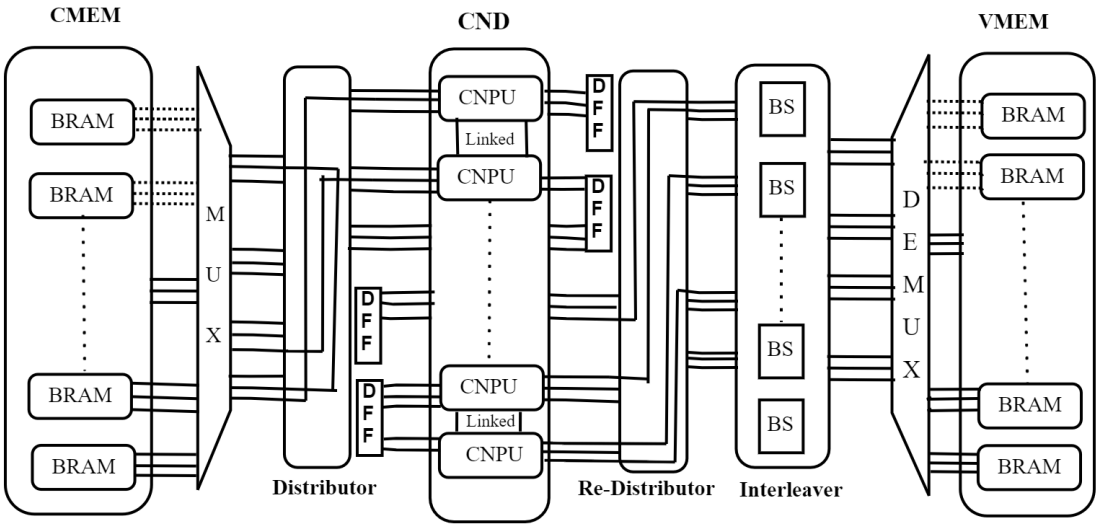


Figure 3.3: An example of the CND Data Path.

integrity.

To optimize hardware resources, the architecture minimizes the number of required decoders, offering flexibility across a variety of PCMs with different coding rates R. By varying m_b while keeping n_b constant, hardware demands are reduced. This is achieved by combining low-degree CNPUs for high-degree rows and linking CNPUs when necessary. This balance reduces the number of high-degree rows in high-rate codes while accommodating many low-degree rows in low-rate codes, optimizing resource usage for decoders needing runtime flexibility.

In scenarios involving high-rate codes, low-degree CNPUs can combine to manage high-degree rows, effectively doubling the clock cycles t_{cb} per block-row. This reduces the overall number of block-rows m_b required, balancing resource usage. To support this flexibility, an additional boolean parameter L is introduced for each PCM, along with a boolean decoder parameter F. The parameter L_p indicates whether rows should be processed using linked CNPUs. The value $L_p = 1$ is determined if $Y_p \geq Y_{min} \times 2$.

Flexible CNPUs are synthesized at design-time if at least one PCM has $L_p = 1$, necessitating the use of linked CNPUs. Otherwise, flexible CNPUs are not synthesized to avoid unnecessary hardware resource consumption.

The Distributor in the CND data path functions similarly to the VND, distributing Γ_v output groups of LLRs. After distribution, the first set of DFFs and multiplexers reduce

Γ_v groups to Γ_c groups by halving the number of groups processed in each cycle. The re-distributor then reassembles these processed groups, ensuring they are correctly organized before writing them back into CMEM. The messages are cyclically left-shifted by D_V programmable Barrel Shifters (BSs) in the Interleaver, as stored in the ADJUST ROM, before reaching CMEM. This ensures accurate data alignment.

For both configurations, null submatrices are managed by setting the multiplexer outputs to off values when the block-row degree $d_c < D_C$. This is controlled by the STATUS ROM, ensuring efficient data handling by the CNPUs, processing only relevant data for storage.

3.3.6 Node Processing Units (NPU) DESIGNS

The design of VNPUs and CNPUs significantly influences the overall hardware resource usage of a decoder. Efficient designs can minimize resource consumption, making the decoder more cost-effective and power-efficient. The critical path refers to the longest path a signal must travel through the circuit. A shorter critical path allows for higher maximum clock frequencies f_{max}, leading to better performance. Higher f_{max} enables faster processing speeds, increasing throughput and reducing latency. Efficient NPU designs enhance throughput and reduce latency, improving overall decoder performance. VNPU design and CNPU designs are created and optimized in HDL.

3.3.6.1 VNPU Architecture

One intrinsic LLR input supplied by the channel and D_V priori LLR inputs are components of the VNPU architecture. It utilizes one posteriori LLR output and D_V extrinsic LLR outputs as the foundation for the determination of the corresponding LDPC-encoded bit. There is a corresponding D_V extrinsic LLR output for each of the D_V a priori LLR inputs. The VNPU calculates the total of the intrinsic LLR input and all a priori LLR inputs except the corresponding one for each D_V extrinsic LLR output. The total of all a priori LLR inputs and the intrinsic LLR input is the a posteriori LLR output. When a VNPU handles a PCM column with $d_v < D_V$, inputs that aren't being used can be turned off by giving them

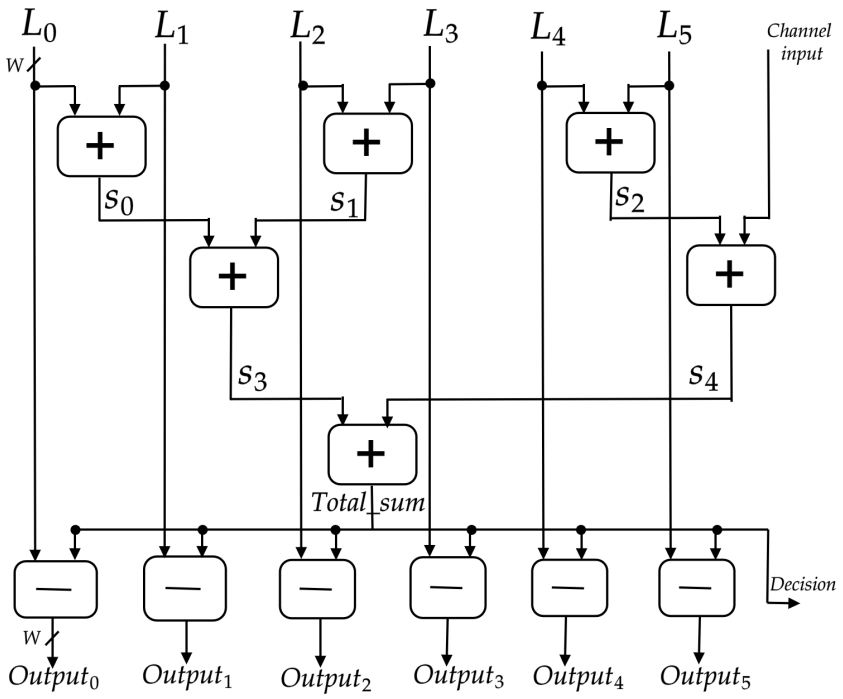


Figure 3.4: Tree Sum-Minus VNPU structure for $D_V = 6$.

a value of 0. This doesn't change the sums. Tree SM (Sum-minus) architecture was chosen for its low hardware resource usage and high f_{max} from its short critical path.

Description of Tree SM Architecture: The VNPU primarily performs addition, with subtraction as its inverse operation. The Tree SM architecture [76] efficiently sums all $D_V + 1$ W-bit LLR inputs using a tree structure, minimizing the critical path length. After calculating the total sum, the architecture uses the inverse operation (subtraction) to compute each output as the total sum minus its corresponding input. To prevent overflow, each addition result expands by one bit, and each output is then saturated back to W bits. The VNPU architecture includes D_V a priori LLR inputs, one intrinsic LLR input, D_V extrinsic LLR outputs, and one a posteriori LLR output. It sums the LLR inputs and adjusts unused inputs by setting them to zero. Various VNPU designs were evaluated for hardware efficiency and f_{max}, with the Tree SM architecture chosen for its optimal performance and minimal critical path length, leveraging a tree structure for summation and subtraction for output calculation.

The Tree Sum-Minus (SM) architecture for a VNPU is designed to efficiently calculate the sum of multiple inputs and then derive individual outputs by using an inverse operation.

Here's a detailed explanation and step-by-step explanation for an example VNPU with $D_V = 6$:

The Tree SM architecture for a VNPU in Figure 3.4, which efficiently computes the sum of several inputs and then derives individual outputs using an inverse operation. In this architecture, for $D_V = 6$, there are seven W-bit wide Log-Likelihood Ratio (LLR) inputs labeled as $L_0, L_1, L_2, L_3, L_4, L_5$, and $channelinput$. The architecture employs a tree structure to generate the total sum of these inputs, ensuring the shortest possible critical path.

The calculation process involves multiple levels of addition, where each level halves the number of nodes. The number of levels required to sum X inputs can be expressed as $\lceil \log_2(X) \rceil$. For $X = D_V + 1 = 7$, three levels of additions are needed. In the first level, inputs are paired and added together, yielding three sums: $S_0 = L_0 + L_1$, $S_1 = L_2 + L_3$, and $S_2 = L_4 + L_5$. The remaining input $channelinput$ moves to the next level unchanged.

In the second level, these sums are further added: $S_3 = S_0 + S_1$ and $S_4 = S_2 + channelinput$. Finally, in the third level, the results from the second level are added to get the total sum: $Total\_Sum = S_3 + S_4$. Once the total sum is calculated, the inverse (subtraction) operation is used to compute each output by subtracting the corresponding input from the total sum, resulting in $Output_0 = Total\_Sum - L_0$, $Output_1 = Total\_Sum - L_1$, and so on until $Output_5 = Total\_Sum - L_5$.

Each addition operation increases the bit width by one bit to avoid overflow, and the outputs are subsequently saturated back down to W bits to match the input bit width. The Tree SM architecture provides a balanced solution, offering a short critical path and making it efficient for FPGA implementation.

3.3.6.2 CNPU Architecture

It uses D_C a priori LLR inputs to compute D_C extrinsic LLR outputs based on the min-sum algorithm. This involves calculating the minimum of all a priori LLR inputs except the corresponding one and multiplying it by the sign of their cumulative product. Unused inputs are turned off by assigning them the maximum positive value representable by a

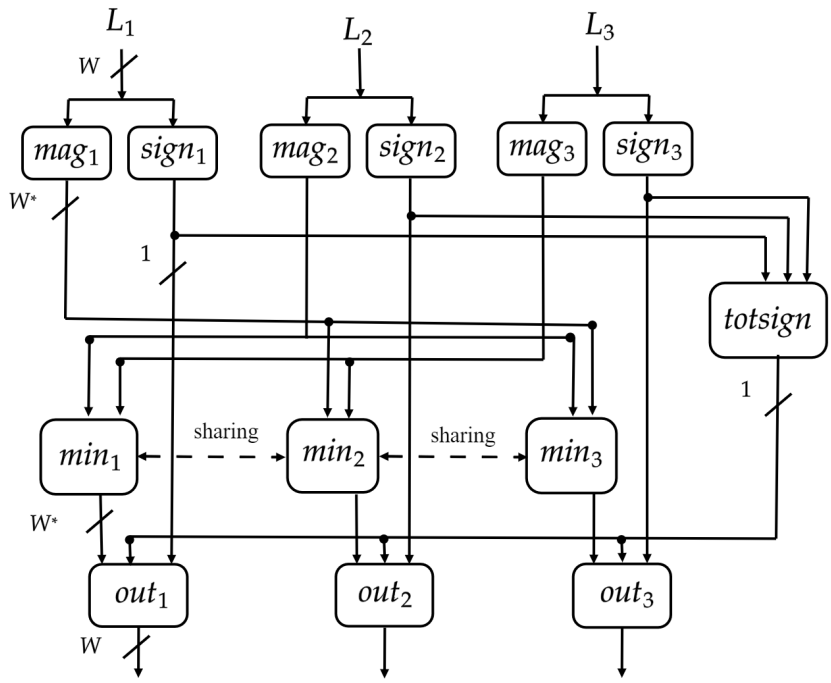


Figure 3.5: Min Tree architecture for $D_C = 3$.

W-bit two's complement integer, ensuring they do not affect the output calculations. This approach simplifies the decoding hardware while maintaining effective performance. Min Tree CNPU structure: The Min Tree architecture [76] is designed to calculate each output efficiently by re-using intermediate results. The Figure 3.5 illustrates an example of Min Tree CNPU structure with a D_C of 3, commonly used in decoders. This structure is designed to find the minimum values among multiple inputs while ensuring that no node has more than three connections. By maximizing the re-use of calculated minimums between tree structures, this approach enhances computational efficiency.

Each input LLR is divided into its magnitude and sign components. For example, the inputs L_1, L_2, and L_3 are processed through magnitude extraction blocks ('mag_1', 'mag_2', and 'mag_3'), which extract the absolute values $|L_1|, |L_2|$, and $|L_3|$. Simultaneously, the sign extraction blocks ('$sign_1$', '$sign_2$', and '$sign_3$') extract the sign bits of each input, determining whether the inputs are positive or negative.

To find the minimum values, the CNPU creates tree structures that compute the minimum of each combination of $D_C - 1$ inputs. With $D_C = 3$, this involves calculating the minimum of each pair of inputs. For instance, the 'min_1' block computes $\min(|L_2|, |L_3|)$,

'min_2' computes $\min(|L_1|, |L_3|)$, and 'min_3' computes $\min(|L_1|, |L_2|)$. These minimum values are then shared across the tree structures to reduce redundancy and improve efficiency.

The total cumulative sign of all inputs is calculated by the '*totsign*' block, which multiplies the signs of L_1, L_2, and L_3. For example, if the signs are Sg_1, Sg_2, and Sg_3, the total sign is $S_{\text{total}} = Sg_1 \times Sg_2 \times Sg_3$.

Each extrinsic LLR output is then calculated using the corresponding minimum value, the total sign, and the original sign of the input. For instance, the output for L_1 is calculated as $\min(|L_2|, |L_3|) \times S_{\text{total}} \times Sg_1$. Similarly, the outputs for L_2 and L_3 are calculated using the same method.

Finally, the outputs are saturated to fit within the range of the signed W-bit output, ensuring they remain within the specified bit range, such as from -2^{W-1} to $2^{W-1} - 1$. This process ensures that the outputs are correctly scaled and within the appropriate numerical limits.

In summary, the Min Tree CNPU structure with $D_C = 3$ efficiently computes extrinsic LLR outputs by re-using intermediate minimum values, maintaining the degree constraint, and ensuring optimal performance in LDPC decoders.

3.3.7 Programmable barrel shifters

Programmable barrel shifters (BSs) are crucial for converting message representations between column-centric and row-centric formats after processing by the VND or CND. This conversion is done by cyclically shifting the messages based on a shift value s. Each BS must handle Z inputs and outputs, requiring additional DFFs when fewer than Z NPUs are used. The design of these BSs can be complex, especially when supporting multiple submatrix sizes at run-time. The chosen design in this work optimally supports a specific set of z values, using a cyclic shift network and tailored HDL descriptions for efficient FPGA implementation.

Once a group of Γ_v messages has been processed by either the VND or CND, it is necessary to convert these messages from a column-centric format to a row-centric format, or vice versa. This conversion is essential to ensure that the messages are correctly ordered

for the next stage of either the CND or VND. The conversion process involves cyclically shifting the messages for each submatrix by a specific shift value s as defined in the matrix H_b. This cyclic shifting is carried out by programmable BSs within the interleaver. The shift value s can be any integer within the range $0 \leq s < Z$. To manage the entire range of potential shift values, each BS must have Z inputs and outputs. When fewer NPUs are used, specifically $\Gamma_v < Z$, due to the parallelism reduction factor F, additional DFFs are necessary. These additional DFFs are placed at the BS inputs to temporarily hold each group of Γ_v NPU outputs until all Z outputs are available. This arrangement ensures that the shifting operation is executed correctly.

This study utilizes a design that is derived from the precise cyclic shift network suggested in [88]. Every BS is tailored to handle a unique set of z values. In a cyclic fashion, the shift value s moves the Z inputs. This cyclic shift is executed consistently, regardless of the value of z. From the shifted input, one can choose each output B_e, where $0 \leq e < Z$. The number of supported z-values bigger than e is represented by u and is used to perform this selection using a u-to-1 multiplexer. The HDL description of the BSs is adjusted for FPGA implementation so it only supports the precise set of z values that the PCM set supports. Software tools are used to accomplish this automatic tailoring. To optimize resource utilization and performance for the specified PCM set, the HDL description of the BSs is adjusted.

3.3.8 Control signals

The proposed decoder architecture utilizes four key control signals to efficiently manage its operations: the Load Signal, the Reset Signal, the PCM Input Signal, and the Start Signal. These signals collectively ensure high throughput and flexibility in the decoder's performance. The following is a detailed explanation of each control signal:

The Load Signal is crucial for ensuring that new sets of intrinsic LLRs are available in the IMMU for decoding. By storing these LLRs in high-speed memory, the decoder can operate at higher frequencies without being bottle-necked by slower external data operations. The IMMU is designed with two locations to allow parallel processing: one location

is used for the current decoding process while the other is loaded with the next frame of LLRs. This overlapping mechanism enables the decoder to handle successive frames efficiently, reducing downtime between frames to a single clock cycle.

The Reset Signal ensures that the decoder can start fresh with a new frame, clearing out any residual data from previous frames. This is essential for maintaining data integrity and ensuring accurate decoding results. The reset signal triggers a switch in the active IMMU location, instantly making the new message available for decoding. Instead of erasing all memory locations, each BRAM location is equipped with a register indicating data validity. During a reset, these registers are set to indicate invalid data, ensuring that null data is read until new data is written.

The PCM Input Signal allows the decoder to switch between different PCMs on the fly, providing the flexibility to handle various coding schemes. This feature is particularly useful in environments with multiple coding schemes. The decoder uses the PCM Input Signal to address ROMs that store pre-characterized PCM information, enabling a quick switch of context and allowing decoding to begin with the new PCM within a single clock cycle, thus maintaining high throughput.

The Start Signal controls the iterative decoding process, allowing it to start, continue, pause, or resume as needed. This flexibility is crucial for managing different decoding scenarios and maintaining synchronization with external control logic. When the start signal is asserted, the decoder begins or continues the iterative decoding process. If the signal is de-asserted, the process pauses, preserving the current state.

These control signals are fundamental to the proposed decoder architecture, facilitating high throughput, flexibility, and efficient memory usage. The Load Signal supports continuous decoding by overlapping the loading of new frames. The Reset Signal ensures the decoder starts fresh with each new frame without lengthy memory erasures. The PCM Input Signal allows rapid switching between different coding rates, and the Start Signal provides precise control over the iterative decoding process.

3.4 Advanced Offline Design Strategy

The proposed offline design flow for LDPC decoders automates the process of generating optimized designs for one or more QC LDPC codes using pre-defined architectures. This comprehensive process involves several key steps: extracting essential parameters and values from the PCMs, generating optimized sub-module structures where necessary, and producing a robust HDL description that can be synthesized into hardware. The design flow

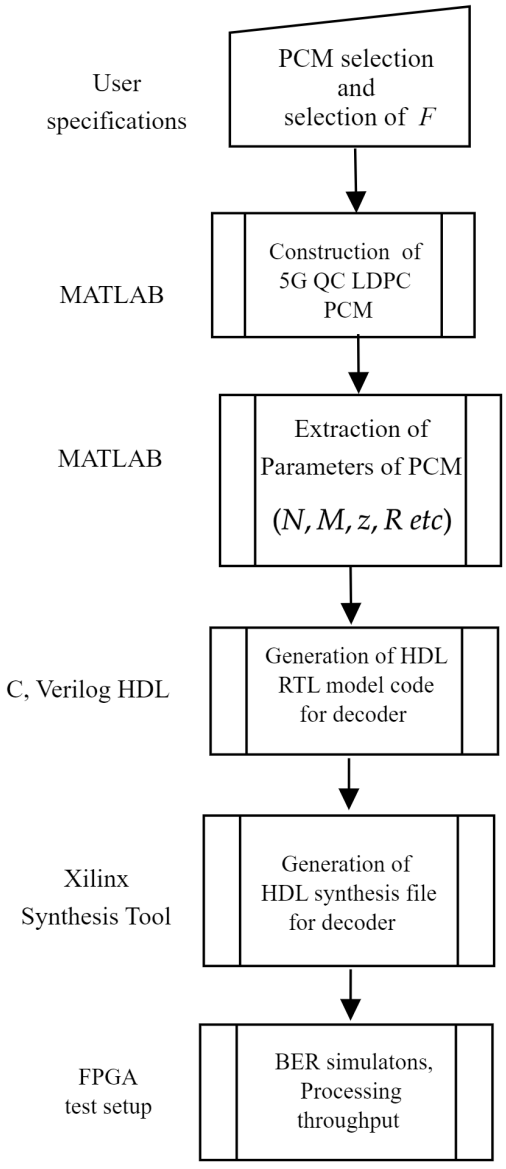


Figure 3.6: Proposed offline design flow.

is visually represented in Figure 3.6, highlighting the sequence of operations, including PCM interpretation, parameter calculation, and automatic HDL generation through cus-

tom C++ applications. This automated approach contrasts with conventional High-Level Synthesis (HLS) tools by allowing software engineers to implement complex algorithms in programmable hardware without requiring deep architectural knowledge. It leverages extensive programming capabilities to generate flexible FPGA-based LDPC decoders that support a wide range of PCM families and parameters.

The primary advantage of this design flow is its ability to rapidly generate and synthesize various decoder configurations, accommodating different PCM sets and degrees of parallelism. This flexibility significantly enhances the efficiency and adaptability of LDPC decoder design, enabling high-performance error correction and processing throughput. By simplifying the implementation process, this design flow makes the generation of robust, optimized hardware designs more accessible, while still meeting specific application requirements. The several detailed steps:

1. User Selection of PCMs: The process begins with the user selecting one or more PCMs. These matrices are critical for defining the error correction capabilities of the LDPC decoder.
2. Initial Selection of F: The user initially selects an integer F within the range $1 \leq F \leq Z$. This value helps determine the level of parallelism and other critical parameters for the decoding process.
3. PCM Interpretation: Using a custom C++ application, the selected PCMs are interpreted. This step involves:
    - Calculating essential decoder parameters such as Γ_v (overall degree of parallelism), N_B , v_e (number of edges per block), M_B, Z , G (girth), D_C, D_V, and I (flexibility indicator).
    - Determining per-PCM parameters, including L (number of layers) and F (parallelism reduction factor), as well as generating ROM values necessary for the hardware implementation.
4. Re-selection of F (if required): Based on the interpreted parameters, the user may need to re-select the value of F to better optimize the decoder's performance and resource usage.

5. HDL Generation: The custom C++ application then generates a SystemVerilog description of the decoder. This HDL code details the structural and functional aspects of the LDPC decoder, tailored to the specific parameters and PCMs selected.
6. HDL Synthesis: The synthesis tool takes the generated HDL description and converts it into an FPGA program object file. During this process, the tool also assesses the hardware resource requirements, ensuring that the design fits within the constraints of the target FPGA device.
7. FPGA Test Setup: The synthesized FPGA program is then deployed onto an FPGA test setup. BER simulations are conducted in this stage to evaluate the decoder's error correction performance and processing throughput. This step is crucial for verifying that the implemented design meets the required specifications and performs efficiently under different conditions.

These stages collectively ensure that the FPGA-based LDPC decoder is designed with run-time flexibility, allowing it to adapt to various PCMs and operational requirements while maintaining high performance and efficiency. The automated and detailed nature of this design flow simplifies the implementation process, enabling the rapid generation and optimization of LDPC decoders for diverse applications.

3.5 Implementation results

3.5.1 Approach

The process begins with identifying a set of 5G NR PCMs and determining an optimal parallelism reduction factor F. This factor is essential for balancing resource efficiency and decoding throughput. An offline design method is employed to create the HDL code for the decoder based on the chosen PCMs and F. This method involves extracting essential parameters such as frame length N, coding rate R, and matrix dimensions from the PCMs. Using these parameters, SystemVerilog code is generated to describe the architecture of the decoder, tailored to meet the specific requirements of the 5G NR PCMs.

The generated SystemVerilog code is then synthesized using Synopsys design tools. The target FPGA for synthesis is selected as the Xilinx Kintex-7-series FPGA board [89]. The synthesis process converts the HDL code into a netlist, representing the hardware implementation of the decoder. During this process, hardware resource usage, including logic elements, memory blocks, and DSP slices, is measured. Additionally, the maximum operating frequency f_{max} of the synthesized design is determined. Both simulation and synthesis stages are performed using Synopsys design tools in conjunction with TSMC 65 nm CMOS standard cell technology. This combination ensures accurate and consistent results, facilitating comprehensive comparative analysis. BER simulations are conducted using C++ to assess the error-correction performance of the synthesized decoder. These simulations involve modeling BPSK transmission over an AWGN channel. The simulations are configured to run up to 15 decoding iterations per frame, with at least 100 frame errors per BER measurement to ensure statistical significance.

3.5.2 Discussion on 5G NR Decoder Performance Metrics

In evaluating the performance of a 5G NR decoder, several key metrics are essential for a comprehensive assessment. These metrics include hardware resource utilization, processing throughput, latency, error correction performance, and energy efficiency. Below is a detailed discussion of each metric and its significance in the context of 5G NR LDPC decoders.

1. Hardware Resource Utilization: Hardware resource utilization refers to the amount of FPGA resources consumed by the decoder, which typically includes logic elements, memory blocks, and DSP slices.

- Logic Elements (LEs): These are basic building blocks of an FPGA, and their usage reflects the complexity of the decoder's design.
- Memory Blocks: These are used to store intermediate data, such as the parity-check matrix, extrinsic information, and other temporary data required during decoding.
- DSP Slices: These are specialized units for performing arithmetic operations efficiently, crucial for the computationally intensive tasks of the decoder.

2. Efficient utilization of these resources is critical for minimizing the cost and maximizing the scalability of the decoder. The Equivalent Logic Blocks (ELBs) metric is often used to standardize resource utilization across different designs, providing a basis for comparison.

3. Processing Throughput: Processing throughput [23] is a measure of how much data the decoder can process per unit time, typically expressed in megabits per second (Mbps).

$$T = \frac{f_{max} \times N \times R}{t_i \times I_a}, \tag{3.3}$$

where f_{max} is the maximum operating frequency, N is the frame length, t_i is the number of clock cycles per decoding iteration, I_a is the average number of decoding iterations.

4. Latency: Latency is the time taken to decode a single frame of data, which is a critical metric for applications requiring real-time processing.

$$\text{Latency} = \frac{k}{T}, \tag{3.4}$$

where k is the message word length (frame length n minus the number of parity bits m), T is the processing throughput. Low latency is crucial for ensuring timely delivery of data in real-time applications, impacting user experience and system responsiveness.

5. Energy Efficiency: Energy efficiency is evaluated based on the energy required to decode a bit of data, often measured by the E_b/N_0 necessary to achieve a specified BER. This metric is significant for mobile and battery-powered devices where power consumption directly affects battery life and device performance.

The performance of a decoder is a balance of these metrics, where improvements in one area may sometimes lead to trade-offs in another. For instance:

- High Throughput vs. Low Latency: Increasing processing throughput can sometimes increase latency if the decoder processes larger frames or more iterations.
- Resource Utilization vs. Performance: Optimizing for minimal resource usage can affect throughput and error correction performance.

In conclusion, a well-designed 5G NR LDPC decoder must efficiently balance these

performance metrics to meet the diverse requirements of 5G applications. Through careful design and optimization, the decoder can achieve high throughput, low latency, excellent error correction, and energy efficiency, making it suitable for the demanding environments of modern wireless communication systems.

3.5.3 Analysis of Results

Table 3.4: Implementation results of different code rate sets.

Decoder	F	PCM N	PCM R	Sub matrix size (z)	t_i	Supported PCMs	ELBs	I_a	T (Mbps)	Latency (µs)	$\frac{Eb}{N0}$ (dB)
5GD-1	1	6528	1/3	96	68	7	74.2k	14.6	3200	0.68	1.9
		6528	2/5		68		74.2k	13.1	4400	0.593	2.2
		6528	1/2		68		74.2k	12.6	5600	0.583	2.4
		6528	2/3		68		74.2k	10.4	7800	0.558	2.5
		6528	3/4		68		74.2k	9.8	9310	0.526	2.6
		6528	5/6		68		74.2k	9.3	10520	0.517	2.8
		6528	8/9		68		74.2k	8.6	12460	0.466	3.1
5GD-2	1	3808	1/3	56	68	7	49.7k	12.8	3550	0.358	2.34
		3808	2/5		68		49.7k	11.3	4760	0.326	2.39
		3808	1/2		68		49.7k	10.2	6380	0.298	2.50
		3808	2/3		68		49.7k	9.6	9265	0.274	2.68
		3808	3/4		68		49.7k	8.4	11125	0.257	2.83
		3808	5/6		68		49.7k	7.8	12843	0.247	3.10
		3808	8/9		68		49.7k	7.3	14365	0.235	3.22
5GD-3	1	3264	1/3	48	68	7	41.4k	11.4	4570	0.238	2.53
		3264	2/5		68		41.4k	10.5	5830	0.224	2.59
		3264	1/2		68		41.4k	9.5	7540	0.216	2.64
		3264	2/3		68		41.4k	8.4	11023	0.197	2.69
		3264	3/4		68		41.4k	8.2	13045	0.188	2.93
		3264	5/6		68		41.4k	8	15450	0.176	3.26
		3264	8/9		68		41.4k	7.5	16980	0.171	3.44
5GD-4	1	1632	1/3	24	68	7	33.6k	9.3	3300	0.165	3.52
		1632	2/5		68		33.6K	8.6	4280	0.152	3.67
		1632	1/2		68		33.6k	8.1	5470	0.149	3.89
		1632	2/3		68		33.6k	7.8	7520	0.145	4.21
		1632	3/4		68		33.6k	7.6	8750	0.140	4.56
		1632	5/6		68		33.6k	7.5	9780	0.139	4.73
		1632	8/9		68		33.6k	7.2	11340	0.128	4.96

The Table 3.4 provides an implementation results of different code rate sets of given 5G LDPC decoder based on several performance metrics. For designs utilizing Xilinx FPGAs, which incorporate complex multi-element slices, we have established a ”utilization” figure of merit. This metric quantifies the average number of Look-Up Tables (LUTs) and D Flip-Flops (D-FFs) employed per slice. By evaluating designs where both the number of slices and the count of LUTs/DFFs are provided, we determined the average utilization rates. The results indicated that approximately 83% of LUTs and 36% of DFFs are used per slice, highlighting that most slices are primarily utilized for their LUTs. When hardware

utilization data is provided only in terms of slices, we assume Equivalent Logic Blocks (ELBs) can be calculated as slices $\times$ 4 LUTs per slice $\times$ 0.83. Here's a summary of the key observations:

1. Hardware Resource Utilization: The 5GD-1 decoder uses the highest number of ELBs at $74.2k$, indicating a more complex design. The 5GD-4 decoder has the lowest ELBs at $33.6k$, making it more resource-efficient compared to the others.
2. Processing Throughput: Throughput (T) varies significantly across decoders and coding rates. For instance, the 5GD-3 decoder achieves the highest throughput of 16,980 Mbps at an 8/9 coding rate. The 5GD-1 decoder, while having higher ELBs, also delivers high throughput, peaking at 12,460 Mbps.
3. Latency: Latency decreases with higher throughput. For example, the 5GD-3 decoder has the lowest latency of 0.171 µs at the highest throughput of 16,980 Mbps.The 5GD-1 decoder, despite its higher throughput, maintains a relatively low latency across different coding rates.
4. Error Correction Performance: The E_b/N_0 values indicate the error correction capability, with lower values being better. The 5GD-1 decoder has the best performance with E_b/N_0 values starting at 1.9 dB. The 5GD-4 decoder, although more resource-efficient, has higher E_b/N_0 values, suggesting a trade-off between resource utilization and error correction performance.
5. Energy Efficiency: Energy efficiency can be inferred from the E_b/N_0 values and throughput. Higher throughput with lower E_b/N_0 indicates better energy efficiency. The 5GD-1 and 5GD-3 decoders show a good balance of high throughput and lower E_b/N_0, suggesting efficient energy usage.

The performance of the decoders is influenced by the balance between hardware resource utilization, processing throughput, latency, error correction performance, and energy efficiency. The 5GD-1 decoder, despite its higher resource utilization, offers excellent throughput and error correction performance, making it suitable for high-demand applications. The 5GD-3 decoder stands out for its high throughput and low latency. The

5GD-4 decoder, with its lower resource utilization, is more cost-effective but with some compromise on performance metrics like E_b/N_0 and throughput. Selecting the right decoder depends on the specific requirements of the 5G application, whether it prioritizes high throughput, low latency, energy efficiency, or minimal hardware resource usage.

3.5.4 Comparative results

For a thorough comparative analysis, the architecture was developed using Verilog HDL. The functionality of this model was then validated through simulation, utilizing test patterns produced by a C++ simulator. Once the design functions were confirmed to be correct, the simulation and synthesis processes made use of Synopsys design tools, which were used in TSMC's 65 nm CMOS standard cell technology.

Table 3.5: Comparative results.

Design	Proposed	[90]	[91]	[92]
Standard	5G-NR	5G-NR	5G-NR	802.16e
Code length	3808	3808	3808	2304
Base code rate	1/3	1/3	1/3	1/2
Decoding algorithm	NMS	SD	CMS	NMS
Scheduling	Flooding	Column-layered	Row-layered	Row- layered
Extrinsic message width	4-bit	1-bit	4-bit	6-bit
Sub-matrix size	56	56	56	96
DCs or Itrs	12	620	10	10
Area (mm^2)	1.32	1.10	1.49	2.9
Throughput (Gbps)	3.55	1.12	3.04	2.20
Power (mW)	230	410	259	870

The provided Table 3.5 compares the proposed model against three other models across various parameters. Here's an in-depth analysis of the comparison:

- Code Length: The code length for the proposed model and the models in [90] and [91] is 3808 bits, which is longer than the 2304 bits used by [92]. The longer code length in the proposed model suggests enhanced error correction capabilities.
- Base Code Rate: The proposed model, along with [90] and [91], uses a base code rate of 1/3, indicating a higher redundancy and better error correction. The model by

[92] has a base code rate of 1/2, suggesting less redundancy and potentially higher efficiency.

- Decoding Algorithm: The proposed model utilizes the NMS algorithm, similar to [92]. The model in [90] uses the SPA, which is more complex, while [91] employs the MS algorithm. NMS and MS are simpler and faster but might be less accurate compared to SPA.
- Scheduling: The proposed model uses flooding scheduling, which differs from the column-layered scheduling in [90] and the row-layered scheduling in [91] and [92]. Flooding scheduling can offer better parallelism but is more complex to implement.
- Extrinsic Message Width: The proposed model has a 4-bit extrinsic message width, which matches [91] and is higher than the 1-bit width used in [90]. [92] uses a 6-bit width, indicating a higher information capacity but also greater computational demand.
- Sub-matrix Size: The sub-matrix size is 56 for the proposed model, [90], and [91], whereas [92] has a larger sub-matrix size of 96. Larger sub-matrix sizes can process more data simultaneously, while smaller sizes can enhance parallelism.
- Decoding Cycles or Iterations (DCs or Itrs): The proposed model requires 12 decoding cycles, significantly fewer than the 620 cycles needed by [90], indicating faster decoding. Models [91] and [92] require 10 cycles, suggesting a balance between speed and accuracy.
- Area (mm²): The proposed model has a chip area of 1.32 mm², which is compact compared to [91] at 1.49 mm² and [92] at 2.9 mm². The model in [90] is the most compact at 1.10 mm², advantageous for small device integration.
- Throughput (Gbps): The proposed model achieves a throughput of 3.55 Gbps, which is higher than [90] at 1.12 Gbps and [92] at 2.20 Gbps. The model in [91] achieves 3.04 Gbps, showing good performance but still lower than the proposed model.
- Power Consumption (mW): The proposed model consumes 230 mW, which is lower than [90] at 410 mW and [91] at 259 mW. However, [92] consumes significantly more power at 870 mW, indicating less energy efficiency.

The proposed model demonstrates a strong balance between performance and efficiency across various parameters. Its adherence to the 5G-NR standard, robust error correction with a 1/3 base code rate, compact design, high throughput, and low power consumption make it a competitive choice for modern communication systems. While it may not lead in every individual metric, the overall profile of the proposed model suggests it is well-suited for applications requiring a mix of performance, efficiency, and compactness.

3.5.5 ASICs Vs FPGAs

ASICs and FPGAs exhibit distinct power consumption characteristics due to their architectural differences. ASICs, being custom-designed, leverage optimized power gating, reduced leakage through tailored transistor sizing, and lower dynamic power via minimized routing. In contrast, FPGAs, being reconfigurable, suffer from higher static power due to unused logic elements and increased dynamic power from extensive routing and switching activity. Clocking and routing further differentiate the two: ASICs employ optimized clock trees for lower power consumption, whereas FPGAs rely on programmable interconnects, leading to higher routing overhead. In FPGA-based LDPC decoders, power consumption is elevated due to LUT usage, routing complexity, and reconfigurable interconnects, whereas ASIC implementations benefit from custom datapaths that significantly reduce both dynamic and leakage power. FPGA power measurements account for static power (unused logic) and dynamic power (switching and routing activity). FPGA implementations consume more power due to increased routing congestion, higher capacitance in configurable logic blocks, and the absence of dedicated low-power optimizations. In contrast, an ASIC implementation would minimize dynamic power by eliminating redundant routing, reduce leakage power through lower-power transistors, and optimize clock gating, leading to a lower energy per decoded bit.

To differentiate FPGA and ASIC power considerations, including a qualitative comparison Table 3.6 outlining expected ASIC power savings. Additionally, an estimated power scaling factor for ASIC extrapolation will be provided based on existing LDPC decoder designs, offering a clearer context for interpreting power results.

Table 3.6: Comparison of Power Consumption Metrics for FPGA and ASIC Implementations.

Metric	FPGA-Based LDPC Decoder	Expected ASIC-Based LDPC Decoder
Static Power	Higher (unused logic)	Lower (only required logic present)
Dynamic Power	Higher (routing overhead)	Lower (custom datapath design)
Clock Power	Higher (programmable clocking)	Lower (optimized clock trees)
Leakage Power	Moderate (depends on FPGA tech)	Significantly lower (custom transistors)
Power per Decoded Bit	1.0x (measured on FPGA)	Estimated 3 to 5 times lower on ASIC

3.6 Conclusion

This chapter has meticulously explored an FPGA-based fixed-point decoder tailored for rate-compatible 5G LDPC codes, focusing on its utility in eMBB applications within 5G networks. We delved into crucial concepts such as eMBB, BGMs, and the NMS algorithm, laying the groundwork for designing LDPC decoders. The architecture of the proposed decoder was thoroughly examined, emphasizing its modular design and the pivotal role of parallelism in boosting decoding efficiency. By employing VNDs and CNDs, along with a versatile memory organization and routing network, the decoder achieves high throughput and low latency. The innovative use of ROMs for storing essential parameters ensures dynamic adaptability to various PCMs, offering runtime flexibility.

The design framework highlights the advantages of a partially parallel architecture, balancing hardware resource use with processing performance. By accommodating a wide array of LDPC code rates within the 5G standard, the decoder meets diverse application requirements, from controlled environments like smart factories to more unpredictable settings such as remote sensing. Implementation results validated the efficacy of the proposed decoder, demonstrating its capability to deliver high throughput and low latency while optimizing hardware resource use. Comparative analysis with other leading decoders showcased its superior performance in terms of power consumption, area, and overall metrics.

The proposed offline design strategy automates the HDL generation process based on

user-selected PCMs and parallelism factors, facilitating rapid development and optimization of LDPC decoders customized to specific needs. This approach not only simplifies implementation but also enhances the overall efficiency and adaptability of the decoder design. In conclusion, the FPGA-based fixed-point decoder for rate-compatible 5G LDPC codes discussed in this chapter offers a robust, flexible, and high-performance solution for modern communication systems. Its dynamic adaptability to varying coding rates and environments, coupled with efficient hardware usage and high throughput, makes it exceptionally suited for the demanding requirements of 5G eMBB applications.

4

A FPGA-Based Stochastic Decoder for 5G LDPC codes

4.1 Introduction

The 5G NR specifications allow H_{b1} to accommodate a wide range of code rates, spanning from 1/3 to 8/9. This diversity in BGMs complexity poses significant challenges for traditional FPGA-based fixed-point LDPC decoders, which usually employ algorithms like the SPA and the MSA. The difficulty is heightened by the need to manage multi-bit, wide-channel LLR messages, which require substantial FPGA resources.

A practical solution is to transform channel probability values into stochastic bit sequence representations in SD. This method simplifies inter-node routing by enabling each

Decoder Type	Code Rate	(N)	(w_v)	(e_l)	(n_I)
SPA	1/3	3808	4.56	4	138,916
SD	1/3	3808	4.56	1	34,729

Table 4.1: Comparison of node interconnects for SPA and SD.

stochastic bit sequence to use a single wire for exchanging extrinsic values between nodes, unlike conventional decoders that require multiple wires for multi-bit wide LLRs. As a result, the single bit-wise computations in Stochastic Variable Node (SVN) and Stochastic Check Node (SCN) units can be implemented using simple logic units. Stochastic LDPC decoders offer error correction performance on par with traditional fixed-point decoders [29]. The routing complexity can be assessed by calculating the number of node interconnects (n_I), using the equation [23]:

$$n_I = 2 \times N \times e_l \times w_v. \tag{4.1}$$

In the context of LDPC codes, where N denotes the codeword length, e_l signifies the length of extrinsic messages, and w_v represents the column weight, a notable comparison arises between two decoding techniques. Consider a 1/3-rate LDPC code with a codeword length of $N = 3808$ and an average column weight of $w_v = 4.56$. When decoded using SD and SPA, the number of node interconnects required are 34,729 and 138,916, respectively. This substantial decrease in node interconnects with SD is attributed to e_l being only 1, as opposed to a minimum of $e_l = 4$ in traditional fixed-point LDPC decoders utilizing SPA. Table 4.1 displays the marked reduction in node interconnects when employing SD over SPA under identical LDPC code configurations.

4.2 Preliminaries

4.2.1 Introduction to Stochastic Computation

Stochastic computation, introduced in the 1960s, revolutionized the approach to complex calculations by utilizing simple circuitry [27, 93, 94]. This method encodes probabilities into streams of random bits, known as Bernoulli sequences, where the statistical charac-

teristics of the bit stream convey the required information [95]. This technique simplifies complex operations, such as multiplication and division, into straightforward bitwise operations manageable by basic stochastic gates [96]. The bit-serial nature of stochastic computation allows for high clock rates while maintaining low hardware complexity.

In this paradigm, probabilities are converted into bit streams called Bernoulli sequences. Each bit in these streams is set to 1 with a probability equal to the value being represented. For instance, a probability of 0.3125 can be represented by various bit streams where the proportion of 1s corresponds to the given probability. To achieve this conversion, a comparator is employed. This device compares the probability value P with a pseudo-random number U at each clock cycle. The output bit is set to 1 if P exceeds U, and 0 otherwise. For a probability of 0.3125, output streams of length 8 could include sequences like '01001001', '01101000', '10010010', or '11001000'. Despite having different bit sequences, these streams consistently reflect the same probability through their proportion of 1s.

By reducing complex operations to simple bitwise manipulations, stochastic computation provides an efficient means to process probabilistic data with minimal hardware requirements. This method supports high-speed processing and is well-suited for various applications where probabilistic representations are crucial.

$$\begin{aligned} P &= 01001001, \\ P &= 01101000, \\ P &= 10010010. \end{aligned} \tag{4.2}$$

4.2.2 Generation of stochastic bit sequences

Gross et al [97] proposed the stochastic approach to LDPC decoding. For a codeword where x_s denotes the s-th symbol, its corresponding LLR can be calculated based on the received symbol y_s at the SVN_s. This LLR, denoted as L_{ch,v_s}, is derived under the assumption of BPSK modulation in an AWGN channel with a mean of zero and a noise variance of σ^2. The formula to compute this channel LLR is given by $L_{ch,v_s} = \frac{2y_s}{\sigma^2}$. The channel

probability is calculated using the formula:

$$P_{ch,v_s} = P(x_s = 1/y_s) = \frac{\exp(L_{ch,v_s})}{\exp(L_{ch,v_s}) + 1}. \tag{4.3}$$

Let exp() represent the exponential function. The binary representation of the value P_{ch,v_s} is evaluated by comparing it with a W-bit Pseudo-Random Number (PRN) $U \in [0,1]$ produced by a comparator [98]. As seen in Figure 4.1, this comparison, which changes with each clock cycle, generates the corresponding stochastic bit sequence of length $f = 8$.

Let $P_{ch,v_s} = 0.3125$, be denoted as $W = 4$-bit fractional binary symbols $P = 0101$. Following each clock cycle, a PRN generator utilizing a Linear Feedback Shift Register (LFSR) produces a fresh PRN $W = 4$-bit PRN U. Provided that the value of P exceeds U, the comparator output for that clock cycle is 1. Otherwise, it is 0. As indicated in Table 4.2, the comparator output after eight clock cycles is 00100101, which represents the stochastic bit sequence associated with $P_{ch,v_s} = 0.3125$. The mean value of this sequence is $\frac{3}{8}$, which serves as an approximation for $P_{ch,v_s} = 0.3125$.

Table 4.2: Example of Stochastic Bit Sequence Generation

Clock Cycle (k)	U	u (Real Value)	Comparator Output ($P > U$)
0	1101	0.8125	0
1	0111	0.4375	0
2	0011	0.1875	1
3	0110	0.375	0
4	1001	0.5625	0
5	0010	0.125	1
6	1100	0.75	0
7	0100	0.25	1

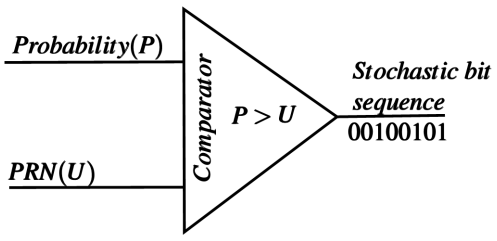


Figure 4.1: Schematic of Comparator

This process effectively translates probabilities into bit sequences that reflect the statistical properties of the original probabilities, crucial for stochastic decoding. These bit

streams are subsequently used for further computations and error correction in LDPC codes [99].

4.2.3 Basic Stochastic Operations in Computation

Stochastic computation enables complex arithmetic operations using simple digital circuits by representing probabilities as streams of random bits. This approach simplifies operations like inversion, multiplication, division, and addition into manageable bitwise processes [100, 101, 102].

1. Inversion: Inversion in stochastic computation flips the bits in a stochastic stream. For a given input stream $a(t)$ with a probability $P_a = \Pr(a(t) = 1)$, the output bit stream $c(t)$ of the inverter will have a probability $P_c = \Pr(c(t) = 1) = 1 - P_a$. For instance, if the input probability P_a is 0.7, the output probability P_c would be calculated as $1 - 0.7 = 0.3$. Given an input bit stream of '0111101', the resulting output bit stream after inversion would be '1000010'.

2. Multiplication: Stochastic multiplication is achieved using an AND gate. For two input stochastic streams $a(t)$ and $b(t)$ with probabilities $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$, the resulting output bit stream $c(t)$ from the AND gate will have a probability $P_c = \Pr(c(t) = 1) = P_a \cdot P_b$. Suppose the input probabilities are $P_a = 0.5$ and $P_b = 0.4$. The output probability P_c would then be $0.5 \times 0.4 = 0.2$. Given the input bit streams $a(t) = 0110001011$ and $b(t) = 1000101001$, the resulting output bit stream $c(t)$ after stochastic multiplication would be 0000001001.

3. Division: Stochastic division can be executed by utilizing a JK flip-flop, with two distinct states: 0 and 1. The transition probabilities between these states are influenced by the input probabilities P_a and P_b. The steady-state probability $P_c = \Pr(c(t) = 1)$ of the output bit being 1 is determined by the equation $P_c = \frac{P_a}{P_a + P_b}$. Consider input probabilities $P_a = 0.4$ and $P_b = 0.6$. The output probability P_c would be calculated as $\frac{0.4}{0.4+0.6} = 0.4$. Given the input bit streams $a(t) = 0101100$ and $b(t) = 1010011$, the JK flip-flop will utilize the Markov model to process these streams and produce the corresponding output stream that reflects the calculated probability.

4. Addition:

Approximate stochastic addition is achieved using an OR gate. When two input stochastic streams $a(t)$ and $b(t)$ have probabilities $P_a = \Pr(a(t) = 1)$ and $P_b = \Pr(b(t) = 1)$, the resulting output bit stream $c(t)$ from the OR gate will exhibit a probability $P_c = \Pr(c(t) = 1)$ given by the formula $P_c = P_a + P_b - P_a \cdot P_b$. Suppose the input probabilities are $P_a = 0.3$ and $P_b = 0.5$. The output probability P_c is calculated as $0.3 + 0.5 - (0.3 \times 0.5) = 0.65$. If the input streams are $a(t) = 0010100$ and $b(t) = 1100010$, the resulting output stream $c(t)$ will be 1110110.

5. Addition with Scaling: In stochastic computation, addition is paired with a scaling operation to maintain the output probability within the interval $[0, 1]$. The scaled sum P_c of N input bit streams $\{a_j(t)\}$ with their respective probabilities $\{P_j\}$ and scaling factors $\{S_j\}$ is calculated as $P_c = \sum_{j=1}^{N} S_j P_j$, where the sum of scaling factors $\sum_{j=1}^{N} S_j$ equals 1.

Consider three input probabilities: $P_1 = 0.2$, $P_2 = 0.3$, and $P_3 = 0.1$. The corresponding scaling factors are $S_1 = 0.3$, $S_2 = 0.5$, and $S_3 = 0.2$. The output probability P_c is computed as $P_c = (0.3 \times 0.2) + (0.5 \times 0.3) + (0.2 \times 0.1) = 0.06 + 0.15 + 0.02 = 0.23$. Given the input streams $a_1(t) = 001000$, $a_2(t) = 011100$, and $a_3(t) = 010000$, a multiplexer combines these streams according to the scaling factors to produce the final output stream.

These stochastic operations enable the execution of complex arithmetic functions using simple digital circuits, suitable for various applications such as neural networks and error-control coding, while maintaining high efficiency and low hardware complexity.

4.2.4 Stochastic decoding algorithmic description

In the process, the channel probability values are represented by a stochastic bit sequence. These values are exchanged iteratively bitwise between the SCNs and the SVNs until the desired codeword is found or the maximum iteration limit is reached [103]. To demonstrate simplification of the decoding procedure, we use a degree-3 SCN and a degree-3 SVN. Both the SCN and SVN components execute the bitwise Modulo-2 arithmetic operations. The steps are shown in the following:

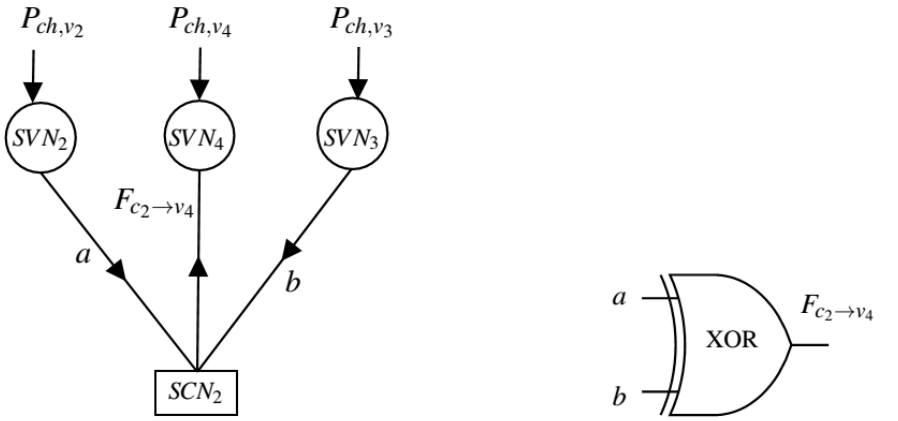
1) Initialization: As a first step, initialize the generated stochastic bit sequence to its

corresponding SVNs vn_j by comparing the W-bit fractional binary equivalent sequence of P_{ch,vn_j} with the W-bit PRN U.

2) SCN update: As shown in Fig. 4.2, the SCN_2 is connected to three SVNs SVN_2, SVN_3, and SVN_4. The arriving SCN to SVN single-bit extrinsic message for node SVN_4 is computed as

$$F_{c_2 \to v_4} = a.(1-b) + b.(1-a). \tag{4.4}$$

In this scenario, a represents a single bit from the stochastic bit sequence $\{a[k]\}$, where k ranges from 0 to $f-1$, associated with P_{ch,v_2}. This bit was transmitted from SVN_2 to SCN_2 during the previous clock cycle. Similarly, b denotes a single bit from the stochastic sequence $\{b[k]\}$, related to P_{ch,v_3}, which was sent from SVN_3 to SCN_2 in the same previous clock cycle.



(a) Degree-3 SCN (b) Schematic of SCN unit

Figure 4.2: SCN and its implementation unit

3) SVN update: In Fig. 4.3, the SVN_4 is connected to three SCNs SCN_1, SCN_2, and SCN_4. The arriving SVN to SCN single-bit extrinsic message for node SCN_2 is computed as

$$F_{v_4 \to c_2} = \frac{e.d}{e.d + (1-e).(1-d)}. \tag{4.5}$$

In this context, e represents a single bit from the stochastic bit sequence $\{e[k]\}$ that was received from SCN_1 to SVN_4 during the previous clock cycle. Similarly, d denotes a single bit from the stochastic bit sequence $\{d[k]\}$, which was received from SCN_4 to SVN_4 in the same previous clock cycle.

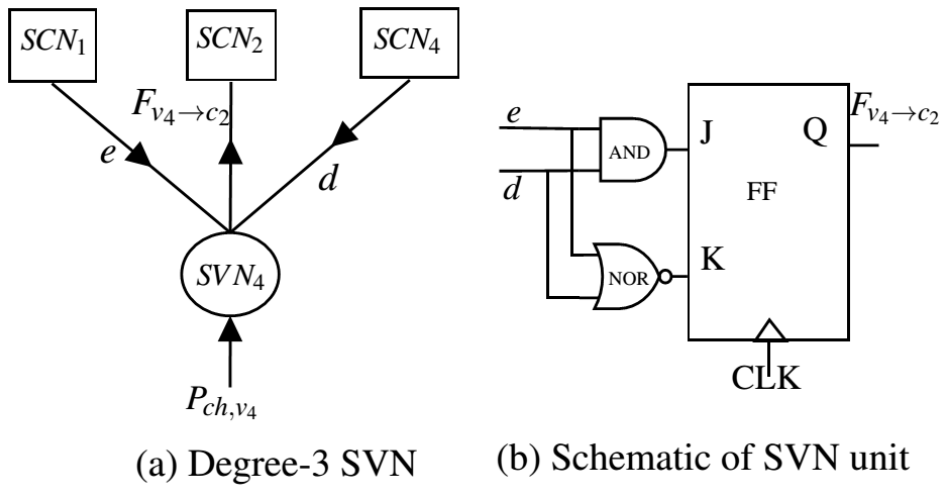


Figure 4.3: SVN and its implementation unit

4) Termination: When the number of decoding cycles (DC) reaches a certain maximum or when all of the parity check equations are met, the iterative procedure will terminate.

4.3 Top Level Design of the FPGA-Based Stochastic Decoder

This section outlines the top-level design of the proposed FPGA-based partially parallel stochastic decoder. The decoder is designed with runtime flexibility, enabling it to decode received messages across any set of code rates specified by the 5G NR standard. The architecture is composed of several fundamental modules, each of which plays a critical role in the decoder's functionality. These modules include: Stochastic Variable Node Decoder (SVND), Stochastic Check Node Decoder (SCND), Base Graph Matrix (BGM) Read-Only Memory (ROM), Controller Unit, Intrinsic Message Memory Unit (IMMU), Routing Network. The SVND and SCND are responsible for processing the variable and check nodes, respectively, using stochastic bit sequences. The BGM ROM stores the structure of the base graph matrix, which is essential for the decoding process. The controller unit manages the overall operation of the decoder, coordinating the activities of all modules. The IMMU stores intrinsic messages, which are probability values converted into stochastic bit sequences. Finally, the routing network facilitates communication between the SVND and

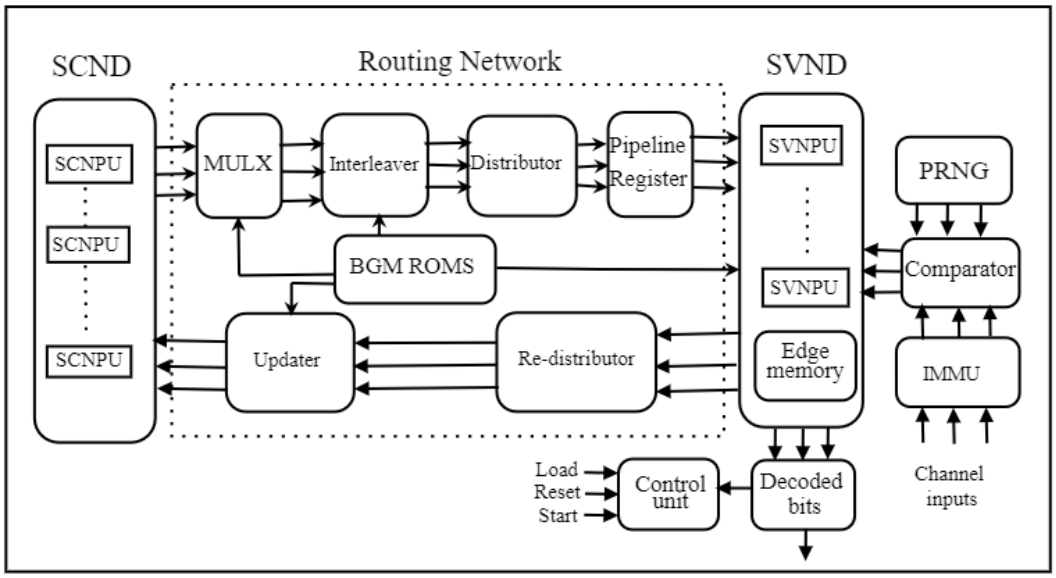


Figure 4.4: Top Level Design

SCND modules, ensuring efficient data exchange. Each of these modules is described in detail in the following subsections, highlighting their individual functions and contributions to the overall decoder architecture. This structured design allows for dynamic adaptability and efficient decoding across varying 5G NR code rates.

4.3.1 Optimized Parallelism and Decoding Strategy

The innovative stochastic flexible LDPC decoder architecture achieves a balance between fully serial and fully parallel decoders through a partially parallel design. This approach enhances processing throughput compared to fully serial decoders while avoiding the complexity associated with routing every edge in fully parallel setups. The level of parallelism, denoted as η_v, is adjustable within the range $1 \leq \eta_v \leq Z$, where Z is the maximum QC expansion factor or sub matrix size z within the supported PCM set. This adaptability allows for a trade-off between hardware resource usage and processing speed, managed by the integer parallelism reduction factor Q. The factor Q subdivides each block-column of the QC PCM, sized z, into q layers, each of size $\eta_v = \frac{Z}{Q}$.

The fixed-point architecture uses a parallelized subdivided flooding schedule, employing both a CND and a VND in parallel, each with distinct datapath and memory structures. This structure allows for the addition of pipelining stages throughout the datapaths, al-

though with significant hardware resource consumption primarily due to routing elements. Conversely, the flexible stochastic architecture adopts a layered decoding schedule to reduce routing complexity by utilizing single-bit stochastic messages and a unified datapath. This architecture features a SCND comprising M D-type Flip-Flops (DFFs) that store the current SCN to SVN values. The flexible routing architecture connects the SCND to the SVND, which includes η_v stochastic Variable Node Processing Units (SVNPUs). Each SVNPU processes a layer of η_v columns of the PCM during each clock cycle, with a single decoding cycle (DC) requiring $t_{DC} = n_b \times q$ clock cycles.

The requirement for message representation within a PCM block to be row-centric is removed by this schedule. Rather of being shifted back, the SVND outputs for each block-row can stay in their current rotation within the SCND. It is possible to incrementally shift each block of values as required by the SVND by utilizing the differences in successive shift values on each block-row of the PCM. By utilizing this strategy, it is possible to utilize a single interleaver module while simultaneously allowing processing parallelism for a maximum of Z columns per clock cycle. This methodology ensures efficient and accurate decoding by leveraging the flexibility and simplicity inherent in the stochastic computation framework within the proposed architecture. By combining partial parallelism with layered scheduling, the architecture optimizes resource usage and processing throughput, making it highly suitable for high-performance LDPC decoding.

Few SVNs and SCNs are instantiated at the same time in partially-parallel designs. The SCND is connected to the SVND by the suggested design, as seen in Figure 4.4. In order to store the updated SCN messages received from the SVND, the SCND uses M D Flip-Flops as memory. In this case, the total number of SCNs are represented by M The design utilizes a shuffled iterative decoding method, which involves splitting the total number of block-columns in the BGM into vertical layers [104]. A single block column from the BGM or η_v columns from the PCM, represented as SVNs in the Tanner graph of the decoded LDPC code, represents each layer. The results of processing each layer's SVN in parallel are used to update the related SCNs before processing the following vertical layer.

Using this method for scheduling, let's look at a QC LDPC code with a codeword

length of N = 3808. This code's PCM is organized into 68 vertical layers, each of which has $\eta_v = 56$ columns. An SVND that consists of 56 SVNPUs is incorporated into the proposed decoder architecture. During each clock cycle, these SVNPUs process one layer of the PCM, which consists of 56 columns, using SVN updates. The updated values are promptly written back to the SCND after they have been processed.

Consequently, there are 68 clock cycles needed to finish a decoding cycle (DC), and each cycle adds one bit to the end of each stochastic stream. This organized method of dividing the PCM into vertical layers and using multiple SVNPUs for parallel processing ensures efficient decoding. The immediate update mechanism further enhances the decoder's performance, enabling it to handle high-throughput requirements effectively.

4.3.2 BGM ROMs

Similar to the fixed-point design, the suggested stochastic decoder architecture also makes use of BGM ROMs. This architecture relies heavily on these ROMs, which store hardware-optimized values that describe each PCM in the supported set. The suggested architecture's runtime flexibility relies on addressing these ROMs with the signal that chooses the current active PCM. The routing network, SVNPUs, and SCNPUs can have their intrinsic and extrinsic message routing and shifting controlled using hardware-optimized ROM blocks that store the location and shift values of non-'-1' entries of each BGM. A key difference from the fixed-point architecture is that only one set of these three ROMs is required for each PCM in the proposed stochastic architecture, rather than separate sets for the VND and the CND of Fixed-point design. This is a benefit of the single datapath architecture, which results from the layered decoding schedule. The single datapath architecture simplifies the design and enhances efficiency. The three ROMs are:

1. ROM1-location: The ROM1-location holds the row indices for all non '-1' entries within each block column of the BGM. This means that if there are non-null submatrices in particular rows of a PCM block, the corresponding row indices are stored in this ROM. For instance, if specific rows in a PCM block contain non-null submatrices, ROM1-location will store these row indices to facilitate accurate processing

during decoding.

2. ROM2-shift: The ROM2-shift records the shift value differences between each non '-1' entry in a block column and the previous non '-1' entry in the same block row. These values are determined during the design phase and stored to streamline the interleaver's complexity. For example, if a submatrix shifts by a specific value relative to its preceding entry, this difference is captured and stored in the ROM2-shift, facilitating efficient data handling during the decoding process.

3. ROM3-weight: The ROM3-weight holds the degree, or weight, of each block column of the BGM. Unlike the NONZERO ROM used in fixed-point architectures, which indicates whether values represent non-null submatrix blocks or empty data, the ROM3-weight directly influences the function of the SVND. For instance, the weight of a block column, which is crucial for the SVND's operation, is stored in this ROM to ensure accurate decoding and efficient processing.

Following the 5G NR standard, the suggested design has three ROMs for a QC LDPC code that has a codeword length of $N = 3808$. Each ROM contains $N_b = 68$ locations, with each location holding $W_v = 30$ values, resulting in a total of $N_b \times W_v = 2040$ locations. The strategic integration of these ROMs allows the stochastic decoder to achieve runtime flexibility and efficient decoding performance. This design optimizes the use of hardware resources, ensuring accurate and effective data processing while adhering to the architectural principles of the decoder. By leveraging the full capacity of these ROMs, the architecture can dynamically adjust to varying code rates and conditions, enhancing overall performance.

4.3.3 Routing network

The routing network is crucial in the stochastic decoder's architecture, facilitating the transfer of single-bit messages from the SCND registers to the η_v SVNPUs in the SVND. Once these messages are updated in the SVND, they are sent back to the SCND. Modules such as the Multiplexer (MULX), Interleaver, Distributor, Re-distributor, and Updater make up

the routing network. Each of these components performs a specific function to ensure the efficient flow of data within the decoder.

The maximum number of message inputs from SCNs to any SVNPU at any given time is W_v. This number corresponds to the highest count of non '-1' entries in any block column within the set of permissible code rates. This configuration supports the necessary flexibility and capacity for efficient decoding in compliance with the 5G NR standard. The detailed functions of each module within the routing network are elaborated in the subsequent sections, providing insight into their roles in maintaining the decoder's operational integrity.

4.3.3.1 Multiplexer

The MULX unit chooses W_v blocks of Z_c bits from the SCND using row-index values from the ROM1-location. A more efficient way is used by the MULX unit instead of directly connecting each input to each output, which requires substantial hardware resources. The requirement for hardware resources is greatly decreased by storing the ROM1-location row index data in ascending order [105].

Every block column in a BGM has a weight w_v, where $w_v \leq W_v$, in actuality. In the ROM1-location, the top w_v block row indices of non '-1' entries in that block column are sorted in ascending order, while the rest $W_v - w_v$ values are left as empty data. This ensures that only the necessary data is processed, thereby reducing hardware complexity.

Using the method proposed in [105], the design's efficiency is significantly improved, as the number of connections needed is reduced by 39%, allowing the multiplexer to support the seven LDPC BGMs of the allowed code rate set of $N = 3808$, with a maximum column weight of $W_v = 30$. This approach not only conserves hardware resources but also simplifies the data selection process, improving the overall performance of the stochastic LDPC decoder.

4.3.3.2 Interleaver

In the Interleaver module, W_v parallel Barrel Shifters cyclically shift the W_v blocks that the MULX unit receives from the SCND. The ROM2-shift stores the shift values that are used for these shifts. It is necessary to convert a set of η_v messages from the submatrix from a row-centric to a column-centric format before the SVND starts reading the data. This transformation is carried out by a Barrel Shifter, which aligns with the corresponding shift value in the BGM H_{b1}.

Since any number between 0 and Z_c can be the shift value, every barrel shifter can accommodate Z_c inputs and outputs. To accommodate the suggested adaptable decoder architecture, which was developed to accommodate the BGMs, the Hardware Description Language (HDL) utilized by these Barrel Shifters has been adapted. The expansion factor Z_c of each BGM affects multiplexer input selection during design. The algorithm described in [88], which is cyclic shift decomposition, guides this selection. This method ensures efficient data reorganization and alignment, facilitating accurate and efficient decoding processes within the flexible decoder architecture.

4.3.3.3 Distributor and Re-distributor

The distributor's primary function is to take W_v blocks of η_v bits from the SCND and rearrange them into η_v blocks of W_v bits. This rearrangement process is essential for preparing the data for further processing by the SVND. To enhance the efficiency of this process, pipeline registers may be used to latch these blocks temporarily before they are processed by the SVND.

After the SVND processes these blocks, the resulting η_v blocks of W_v bits undergo a similar rearrangement process by the Re-distributor. This process converts the η_v blocks back into W_v blocks of η_v bits. This step is crucial for maintaining the proper data structure and ensuring that the information is correctly aligned for subsequent stages of the decoding process.

By systematically rearranging the blocks during both the distribution and re-distribution stages, the architecture ensures optimal data flow and processing efficiency, aligning with

the design principles of the flexible stochastic LDPC decoder.

4.3.4 Pipelining in Stochastic Decoding

Because binary operations are inherent in SD, message updates in each layer can either entirely alter the prior message or keep it unchanged. Therefore, the layered schedule in SD necessitates that all updates from the previous layer are stored before the next layer begins. Delaying a single layer's update can result in a complete decoding failure, as verified through extensive BER simulations. To ensure effective pipelining, each block-column must be processed exactly once per decoding cycle. Some PCMs feature a column-wise permutation, ensuring column orthogonality with neighbouring columns. This arrangement allows a pipelining stage to be added without impacting decoding performance, as updates from each layer are not needed until at least two clock cycles later [105].

If a natural permutation is not present, artificial insertion of a few empty "stall" layers between non-orthogonal columns can achieve the same effect. By systematically following these steps, the architecture ensures efficient and accurate decoding, balancing resource use and processing speed.

1. Identify Block-Columns: Each block-column must be processed exactly once during each decoding cycle to ensure efficient operation. This is crucial for maintaining the integrity of the decoding process. In practice, this means arranging the block-columns so that they exhibit column orthogonality relative to their neighboring columns. This orthogonality ensures that the processing of one block-column does not interfere with the others, thereby facilitating smoother and more reliable decoding.

2. Insert Stall Layers : To maintain orthogonality, τ stall layers are introduced into the processing sequence. This helps in ensuring that the decoding process remains efficient and accurate. For instance, when a PCM (Parity Check Matrix) is reshuffled, these stall layers are inserted strategically, to align with the orthogonal processing requirements. This method enhances the overall decoding performance by preventing

overlap and ensuring that each block-column is processed correctly.

3. Calculate Clock Cycles: To account for the introduction of τ stall layers, it is essential to determine the extra clock cycles required for each decoding cycle. This involves ensuring that the increase in operating frequency compensates for the additional cycles, thereby maintaining or improving the overall processing throughput. The objective is to achieve a balance where the enhanced frequency offsets the extra time introduced by the stall layers, ensuring efficient decoding performance.

4. Evaluate Throughput Improvement: To determine whether the addition of the pipeline stage enhances throughput, apply the processing throughput equation. This assessment will reveal if the increased operating frequency provided by the pipeline stage outweighs the extra clock cycles required, ensuring a net improvement in decoding efficiency.

Evaluate Throughput Improvement: To determine if the addition of a pipeline stage improves throughput, follow these steps:

1. Identify the Current Throughput: To determine the current throughput of the decoder without the pipeline stage, use the following formula:

$$\text{Throughput} = \frac{f_{\max} \times n \times R}{t_{\text{DC}} \times I_a}$$

In this equation, $f_{\max}$ denotes the maximum operating frequency, n is the number of bits in the codeword, R represents the code rate, t_{DC} indicates the number of clock cycles per decoding cycle, and I_a is the number of iterations. Calculating the throughput using these parameters will give you the baseline performance of the decoder.

2. Determine the Impact of the Pipeline Stage: To introduce the pipeline stage, add τ stall layers, resulting in a new number of clock cycles per decoding cycle:

$$t'_{\text{DC}} = t_{\text{DC}} + \tau$$

Next, evaluate the increase in the maximum operating frequency due to the pipeline stage:

$$f'_{\max} > f_{\max}$$

This step assesses how the pipeline stage affects both the clock cycles and the operating frequency of the decoder.

3. Calculate the New Throughput with Pipeline: Determine the new throughput incorporating the pipeline stage by using the revised number of clock cycles and the enhanced maximum operating frequency:

$$\text{Throughput}' = \frac{f'_{\max} \times n \times R}{(t_{\text{DC}} + \tau) \times I_a}$$

This calculation accounts for the increased frequency and additional clock cycles introduced by the pipeline stage, providing a measure of the updated decoding efficiency.

4. Compare Throughputs: Assess whether the new throughput exceeds the original throughput by evaluating the following condition:

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

If this inequality holds true, the pipeline stage results in a net improvement in decoding efficiency.

By systematically introducing a pipeline stage, recalculating the throughput, and comparing the results, you can determine whether the increased operating frequency provided by the pipeline stage results in a net improvement in decoding efficiency.

4.3.4.1 Updater

The Updater module utilizes the values stored in the ROM1-location to transfer the W_v blocks of η_v bits back into the SCND at the correct positions. This ensures that data is accurately placed within the SCND, maintaining the integrity of the decoding process.

Additionally, the ROM3-weight is employed to ensure that only the most recent W_v values are written, effectively filtering out any non '-1' entries or irrelevant "don't care" data. This selective writing process is crucial for maintaining the accuracy and efficiency of the decoder.

Given that D-Flip Flops are used to implement the SCND, each layer can be written immediately, ensuring rapid and efficient data updates. This capability enhances the overall performance of the stochastic LDPC decoder, aligning with its design goals of flexibility and high throughput.

4.3.5 Stochastic variable node decoder (SVND)

The SVND module comprises η_v parallel SVNPUs. Each SVNPU is responsible for processing a single block column from the BGM H_{b1}, which means each SVNPU handles one column of the PCM. The SVNPUs are designed with multiple inputs and outputs, totaling $W_v + 1$, where W_v denotes the highest column weight among the BGM code-rate set. The additional input and output are used for channel-intrinsic messages and the bit estimation, respectively. It is crucial to account for this maximum number of inputs and outputs to ensure that each SVNPU can decode any column with an arbitrary weight within the code-rate set. The SVND module accepts W_v input bits from the connected SCNs and an intrinsic bit from the IMMU. Based on these inputs, the SVND performs the SVN update operation and transmits the updated extrinsic values to the corresponding SCNPUs.

4.3.6 Stochastic Variable Node Processing Unit (SVNPU)

In stochastic architectures, the likelihood of the SVNPU output being in the Hold state increases with the number of SVNPU inputs. The Hold state occurs when the two input bits of the SVNPU are not equal, causing the current output to continuously repeat the previous output. This state hampers the decoder's ability to make accurate decisions, degrading BER performance. To mitigate this, high column-weight SVNPUs are constructed using low-weight sub-nodes with memory blocks, typically employing sub-nodes with weights $w_v \leq 4$.

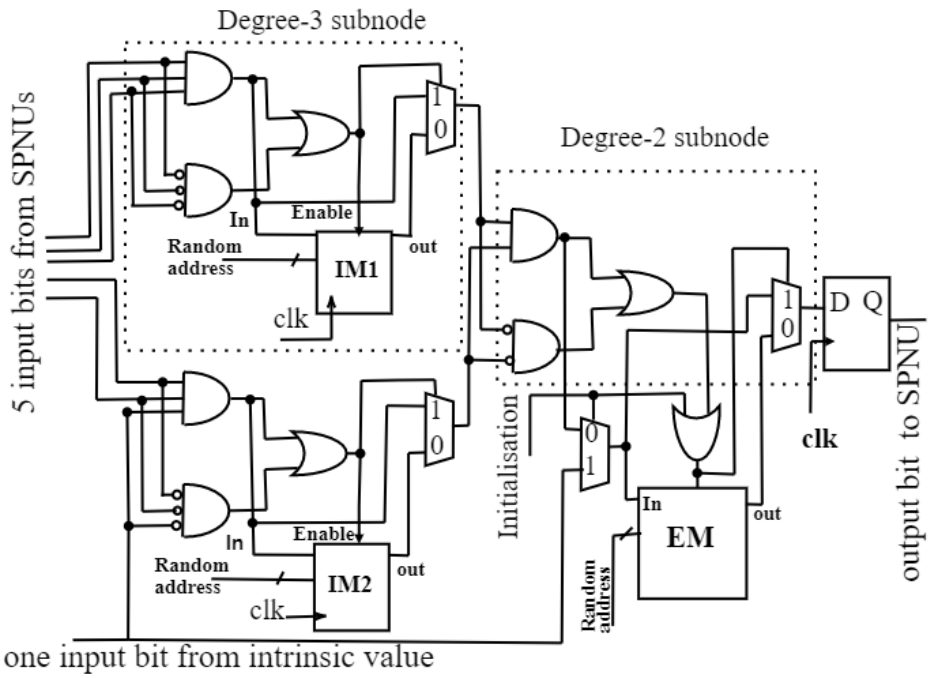


Figure 4.5: Block diagram of SVNPU with high $w_v = 6$.

Construction of High Column-Weight SVNPU : For example, a column-weight $w_v = 6$ SVNPU can be built using two column-weight $w_v = 3$ sub-nodes and one $w_v = 2$ sub-node. Internal Memory (IM) blocks, each with a length of 2 bits, and Edge Memory (EM) blocks, each with a length of 64 bits, are used to minimize correlation in the stochastic sequence caused by the Hold state. A dual-tree design is employed to construct a high column-weight SVNPU, making it flexible enough to handle any number of active inputs and outputs up to a maximum of $\lambda = W_v + 1$, which is a power of 2. For instance, an SVNPU with a column weight $W_v = 30$ will have $\lambda = 32$ inputs and outputs, requiring 90 sub-nodes in total. The dual-tree structure comprises summing and combining sections, each with four stages.

SVNPU architectures assume a "null" value for inputs greater than the degree d_v of the active block column. However, in stochastic bit streams, both 0 and 1 impact the represented probability, necessitating an alternative design. The dual-tree architecture extends to any number Λ of inputs and outputs, comprising $N_{\text{sub}} = 3 \times \Lambda - 6$ sub-node blocks, each with two inputs and one output. This structure is aligned with the forwards-backwards algorithm and minimizes hardware requirements for non-invertible functions.

The proposed flexible stochastic decoder uses these sub-nodes as flexible Edge Memories (EMs) and Internal Memories (IMs) [31]. The generalization of the dual-tree archi-

tecture allows any 2-input 1-output sub-node function, facilitating its use in various architectures. Flexibility is achieved by placing multiplexers at the outputs of key sub-nodes, allowing flexible signal routing and function replication.

The dual-tree topology consists of summing and combining sections [105], each with $Y = \lfloor \log_2(\Lambda) \rfloor - 1$ stages. Each summing stage y combines intermediate signals S_y^x from the previous stage. This continues until only two signals remain. The combining section processes these signals further, doubling the number of intermediate signals in each stage.

Multiplexers enhance flexibility by bypassing unused NPU inputs. When $\lambda_a < \lambda_n^x$, a multiplexer bypasses the sub-node to prevent signal corruption. This ensures that intermediate signals can replace sub-node outputs if inputs are inactive. The value $\lambda_n^{y,x}$ denotes the minimum value of active inputs for bypassing the corresponding sub-node:

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

This detailed explanation outlines the SVNPU architecture, its construction using sub-nodes and memory blocks, the flexibility provided by dual-tree architecture, and the strategic use of multiplexers to optimize performance and minimize hardware requirements.

4.3.7 Control unit

The Control Unit in the proposed decoder architecture oversees both internal and external control signals, including those that manage the iterative decoding process. A key signal is the LOAD signal, which indicates when a new set of $\eta_v \times w$-bit intrinsic probabilities needs to be loaded into the Intrinsic Message Memory Unit (IMMU). When the decoder is set to process a new frame, all modules must be reset to their default states using the RESET signal.

One of the standout features of this architecture is its ability to switch between differ-

ent BGMs during a single clock cycle. This is facilitated by the parameterization of the supporting BGMs in the ROMs at compile-time, providing a high degree of flexibility.

The decoding process is initiated and sustained by the START signal. As long as this signal remains active, the decoder continues to perform iterative decoding. The decoding operation halts when the START signal is deactivated, ensuring efficient and controlled processing. This dynamic control mechanism enables the decoder to adapt quickly to varying decoding tasks, enhancing its operational versatility and effectiveness.

4.4 Design flow

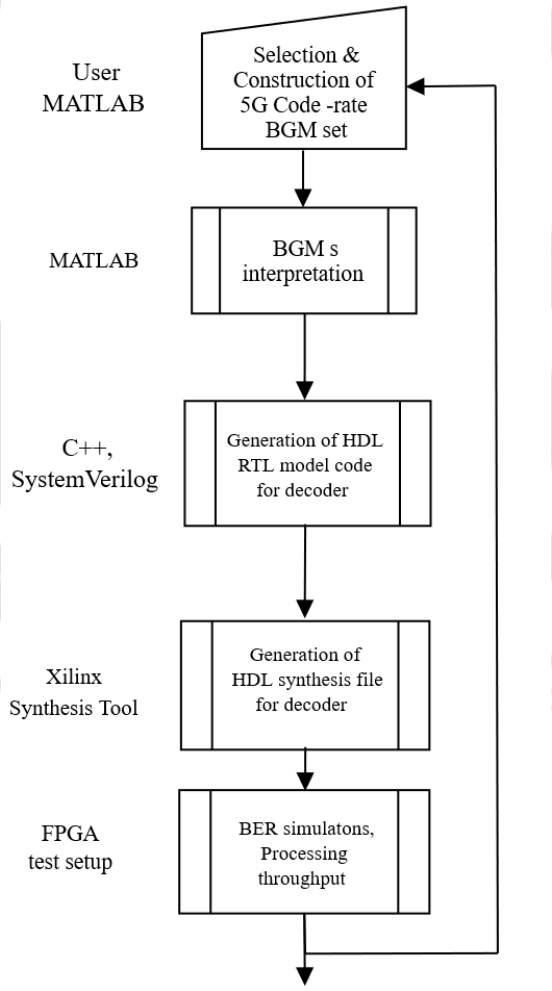


Figure 4.6: Flow chart for Offline design of the proposed decoder.

The flowchart in Figure 4.6 outlines the comprehensive offline design and implementation process for the proposed 5G code-rate LDPC decoder. This process involves several

key stages, each integral to the development of an efficient and effective decoder. Below is a detailed step-by-step description of each stage, accompanied by relevant examples:

1. Selection and Construction of 5G Code-Rate BGM Set:The design process initiates with the selection and construction of BGMs tailored for different 5G code rates. Using MATLAB, users develop and select the appropriate BGMs that meet the 5G NR standard specifications. For instance, a user may choose BGMs such as H_{b1} and H_{b2} for specific code rates like 1/3 and 2/3, respectively, depending on the requirements of their particular application. This step ensures that the chosen matrices are optimally suited for the intended decoding tasks.

2. BGM Interpretation: The next phase involves interpreting the selected BGMs using MATLAB. This step requires understanding and translating the structure of the BGMs to facilitate subsequent design stages. It involves extracting key architectural parameters, such as the maximum matrix dimensions, including the number of block columns N_B, block rows M_B, lifting size Z_c, column-weight W_v, and row-weight W_c. For example, MATLAB scripts can be utilized to read matrix elements and prepare the data for HDL model code generation, ensuring the accurate representation of each non-zero entry and their respective positions within the matrix. This precise interpretation is crucial for developing an effective hardware design.

3. Generation of HDL RTL Model Code for Decoder: In this stage, the RTL model code for the decoder is crafted using programming languages like C++ and SystemVerilog. This code specifies the logical operations and data flow within the decoder. For instance, a SystemVerilog script could be created to define the functionality of a SVND or SCND, outlining how each input bit is processed and routed through the decoder. This step is critical for establishing the decoder's architecture and ensuring its operations are accurately represented in hardware.

4. Generation of HDL Synthesis File for Decoder: The RTL model code is then transformed into a synthesis file, suitable for FPGA implementation. This process is carried out using tools like the Xilinx Synthesis Tool. For example, the synthesis tool

translates the SystemVerilog code into a bit stream file, which can be uploaded to an FPGA board for testing and deployment. This conversion ensures that the logical operations defined in the HDL code are accurately realized in the FPGA hardware, enabling the decoder to perform its intended functions efficiently.

5. FPGA Test Setup: The final synthesized HDL file is uploaded to an FPGA for testing. This phase involves preparing the FPGA test environment and performing simulations to validate the decoder's functionality and performance. For instance, the FPGA might be connected to a test bench to conduct BER simulations and throughput tests, ensuring the decoder operates according to the desired specifications. This setup helps identify and resolve any issues, ensuring the decoder's reliability and efficiency in real-world applications.

6. FPGA Test Setup: The final synthesized HDL file is uploaded to an FPGA for testing. This phase involves preparing the FPGA test environment and performing simulations to validate the decoder's functionality and performance. For instance, the FPGA might be connected to a test bench to conduct BER simulations and throughput tests, ensuring the decoder operates according to the desired specifications. This setup helps identify and resolve any issues, ensuring the decoder's reliability and efficiency in real-world applications.

7. BER Simulations and Processing Throughput: This stage involves executing BER simulations and assessing the processing throughput of the decoder. These evaluations are crucial for understanding the decoder's performance under different conditions. For example, BER simulations might include transmitting a predefined sequence of bits through the decoder and comparing the error rate with the original sequence. Throughput tests could measure the rate at which the decoder processes bits, determining how many bits per second the decoder can handle, thus providing a comprehensive performance analysis.

This structured approach ensures that the proposed decoder is rigorously tested and optimized before deployment, providing a reliable solution for decoding 5G code rates.

4.4.1 An example

In the case of a QC LDPC code with a codeword length of $N = 3808$ and a code rate of $R = 1/3$, which complies with the 5G NR standards, the corresponding BGM H_b has dimension of 46×68. This matrix includes 308 non '-1' entries, equivalent to 68×4.52, reflecting an average variable node degree of 4.52. The BGM H_b serves as the foundation for constructing the PCM H. To compute the number of non-zero entries in PCM, the 308 entries are multiplied by the lifting size or expansion factor $Z_c = 56$, resulting in 17,248 non-zero entries. This multiplication implies a total of double the 17,248 connections between the variable nodes and check nodes in the decoder, highlighting the complexity of the network's architecture.

The task of modeling the numerous interconnections in HDL can consume upwards of thirty minutes, particularly when engaging with a partially parallel architecture, making it a time-intensive process. The activities involved in simulating and debugging the manually coded HDL model present considerable challenges and are notably time-consuming. Additionally, adapting the model to accommodate decoders with varying code lengths tends to be both inefficient and repetitive. Consequently, to eliminate these redundant tasks, adopting an efficient design methodology is crucial. Addressing these challenges, an automation tool was developed utilizing MATLAB and C++ programming language. This tool significantly enhances efficiency by generating connections between decoder nodes in less than a minute.

The proposed design workflow facilitates automated design adaptability, leading to the development of an optimal FPGA-based Stochastic LDPC decoder tailored for a select range of 5G QC LDPC codes. Specifically, for the $N = 3808$ LDPC code-rate set, the parameters are established as $N_b = 68$, $M_b = 46$, $Z_c = 56$, $W_v = 30$, and $W_c = 19$. Additionally, the process involves identifying the positions and shift values of the non '-1' entries within each BGM, aligning them accordingly with the ROMs. Utilizing the parameters identified in the initial stages, the design employs a High-Level Synthesis (HLS) tool [106] to generate the HDL SystemVerilog code from the high-level language C++. Following the RTL modeling of the decoder, it undergoes synthesis using the Xilinx Synthesis Tool to as-

certain the decoder's hardware requirements. BER simulations are subsequently conducted on the FPGA test setup.

4.5 IMPLEMENTATION RESULTS and DISCUSSION

Approach

Following the offline design strategy, HDL code specific to the selected code-rate set is developed. This code is subsequently synthesized and deployed on a Xilinx Kintex-7 XC7K160T series FPGA board [89]. Simulations to measure this efficiency involve at least 100 frame errors per BER metric and permit a maximum of 600 decoding cycles per frame.

The FPGA board's decoder module accepts intrinsic channel LLRs as inputs, which are transferred through the computer's RS232 port. An RS232 transceiver module is specifically designed to enable this communication link with the computer. In this setup, MATLAB, operating on the computer, is utilized to dispatch and subsequently retrieve the same set of decoded LLRs following the decoding operation on the FPGA. Afterward, MATLAB conducts a comparison of the input and output LLRs from the FPGA to determine the BER performance.

This approach ensures a comprehensive evaluation of the synthesized decoder's performance in terms of hardware efficiency, processing speed, and transmission energy efficiency, facilitating reliable and optimized LDPC decoding for 5G applications.

4.5.1 FPGA Implementation Results

The FPGA implementation results, summarized in Table 4.3, offer a detailed comparison between the proposed 5G stochastic decoder and the conventional Min-Sum decoder. This comparison highlights the advantages of the stochastic decoder in terms of hardware efficiency, flexibility, and overall performance. Key Metrics and Comparison :

1. Code Length and Base Code Rate: Both decoders are designed for a code length of 3808 bits with a base code rate of 1/3. This ensures a consistent basis for comparison.

Table 4.3: FPGA Implementation results.

Standard	5G	5G
Code length	3808	3808
Base Code rate	1/3	1/3
Sub-matrix size	56	56
Implementation	Kintex-7 FPGA	Kintex-7 FPGA
Decoding algorithm	Stochastic Decoding	Min-Sum
Scheduling	Column-layered	Row-layered
No.of interconnects	34729	138916
Intrinsic message width	8-bit	4-bit
Extrinsic message width	1-bit serial	4-bit
LUTs	8,278	12,962
Slice Registers	1,767	2,041
DCs or Itrs	$\approx$ 620 DCs	15
Avg. throughput	$\approx$ 953 Mbps	1.5 Gbps
E_b/N_o at BER=10^{-6}	2.65dB	2.57dB

2. Sub-Matrix Size: The sub-matrix size of 56 is used for both decoders, providing a standard framework for analyzing performance.

3. Implementation Platform: The decoders are implemented on the Xilinx Kintex-7 XC7K160T series FPGA [89], ensuring that hardware comparisons are made on an equal footing.

4. Decoding Algorithm: The proposed decoder utilizes stochastic decoding, a bit-serial approach, whereas the Min-Sum decoder employs a traditional Min-Sum algorithm.

5. Scheduling: Stochastic decoding uses a column-layered scheduling approach, processing one column of nodes at a time. The Min-Sum algorithm adopts a row-layered scheduling, processing one row of nodes at a time.

6. Number of Interconnects: The stochastic decoder requires significantly fewer interconnects (34,729) compared to the Min-Sum decoder (138,916). This reduction in interconnects is crucial as it lowers the complexity and potential congestion in the routing network, leading to more efficient data processing.

7. Intrinsic and Extrinsic Message Widths: The intrinsic message width for the stochastic decoder is 8-bit, while the Min-Sum decoder uses 4-bit. For extrinsic messages,

the stochastic decoder uses a 1-bit serial width, whereas the Min-Sum decoder uses 4-bit, which simplifies the hardware requirements for the stochastic decoder.

8. Hardware Utilization: The stochastic decoder utilizes 8278 Look-Up Tables (LUTs) and 1767 slice registers, while the Min-Sum decoder requires 12,962 LUTs and 2041 slice registers. This demonstrates a substantial hardware savings of approximately 37% for the stochastic decoder, making it more resource-efficient.

9. Decoding Cycles (DCs) or Iterations (Itrs): The stochastic decoder needs about 620 decoding cycles per frame, whereas the Min-Sum decoder requires only 15 iterations. Although the stochastic method demands more cycles, its design is optimized to handle these efficiently.

10. Average Throughput: The stochastic decoder achieves an average throughput of approximately 953 Mbps, compared to the Min-Sum decoder's 1.5 Gbps. Despite the lower throughput, the stochastic decoder's efficiency in terms of hardware resource utilization compensates for this difference.

11. Energy Efficiency (E_b/N_o at BER = 10^{-6}): The energy efficiency for the stochastic decoder is measured at 2.65 dB, slightly higher than the Min-Sum decoder's 2.57 dB. This slight difference indicates that while both decoders perform well, the stochastic decoder maintains competitive energy efficiency.

Advantages of the Stochastic Decoder over the Min-Sum Decoder

1. Reduced Hardware Complexity: The stochastic decoder's significant reduction in the number of interconnects (34,729 vs. 138,916) simplifies the routing complexity, making the design more efficient and easier to manage.

2. Lower Hardware Utilization: With fewer LUTs (8278 vs. 12,962) and slice registers (1767 vs. 2041), the stochastic decoder is more resource-efficient, reducing overall FPGA resource consumption.

3. Simplified Message Handling: The 1-bit serial extrinsic message width in the stochastic decoder simplifies the data handling and processing compared to the 4-bit width

in the Min-Sum decoder, leading to streamlined operations and reduced hardware requirements.

4. Energy Efficiency: Despite a slightly higher E_b/N_o value, the stochastic decoder maintains competitive energy efficiency, ensuring reliable performance with minimal energy consumption.

5. Scalability and Flexibility: The column-layered scheduling and single-bit message handling of the stochastic decoder make it highly adaptable and scalable, suitable for various 5G code rates and configurations.

The comparison between the proposed stochastic decoder and the Min-Sum decoder reveals the former's superior hardware efficiency, reduced complexity, and competitive performance metrics. Although the stochastic decoder has a lower processing throughput, its significant hardware savings and effective handling of interconnects make it a viable solution for high-performance 5G LDPC decoding.

4.5.2 BER Performance Analysis

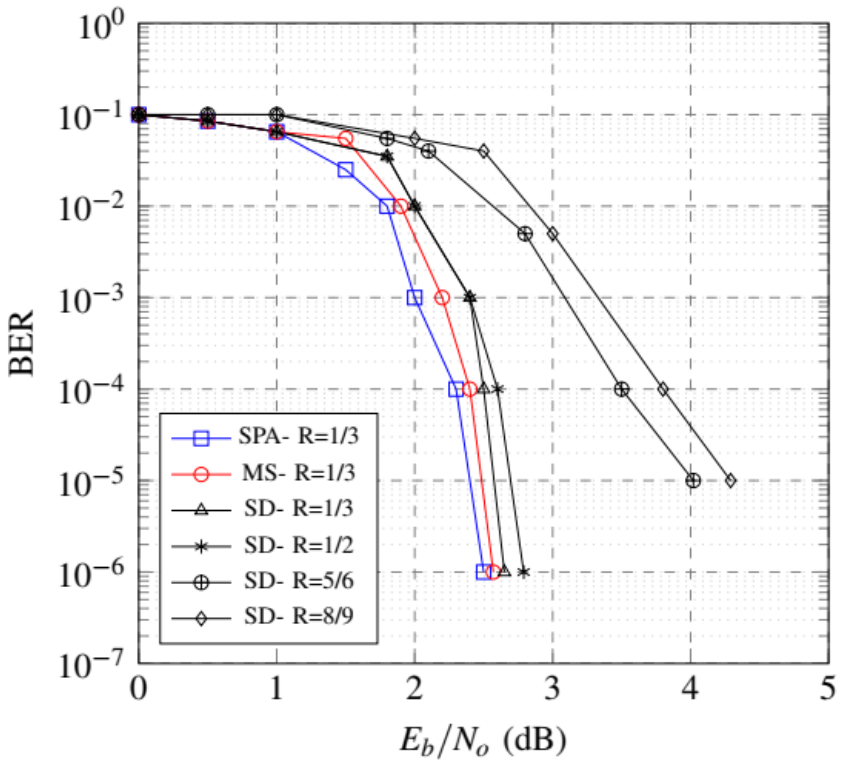


Figure 4.7: BER plot of various algorithms and code rates for N=3808.

Figure 4.7 presents a comparative analysis of the Bit Error Rate (BER) performance across different decoding algorithms and code rates for a codeword length $N = 3808$. The analysis includes the SPA with a code rate $R = 1/3$, the MSA with $R = 1/3$, and the Stochastic Decoding (SD) algorithm with code rates $R = 1/3$, $R = 1/2$, $R = 5/6$, and $R = 8/9$. Key Observations:

1. SPA (R = 1/3) vs. MS (R = 1/3): The SPA and MSA display nearly identical BER performance at a code rate of $R = 1/3$. Their BER curves are so close that they almost overlap, indicating that both algorithms are virtually equivalent in their error correction capabilities for this particular code rate. Despite this overall similarity, SPA demonstrates a slight edge over MSA at very low BER levels. This minor performance advantage suggests that SPA is marginally more effective than MSA in achieving lower BER as the E_b/N_0 ratio increases.

2. SD (R = 1/3) vs. SPA and MS (R = 1/3): The SD algorithm for a code rate of $R = 1/3$ exhibits performance comparable to both the SPA and MSA. The BER curve for SD at $R = 1/3$ closely mirrors those of SPA and MS, particularly when the E_b/N_0 ratio is around 2 dB. This similarity in BER performance demonstrates that the SD algorithm can achieve error correction capabilities on par with traditional SPA and MS algorithms for the same code rate. Moreover, the SD algorithm offers the added advantage of potentially simpler hardware implementation, making it an attractive alternative in practical applications.

3. Performance of SD Across Various Code Rates: For a code rate of $R = 1/2$, the SD algorithm shows improved BER performance compared to $R = 1/3$, achieving lower BER values at the same E_b/N_0 levels. This indicates that the SD algorithm performs effectively at moderate code rates. However, as the code rate increases to $R = 5/6$ and $R = 8/9$, the performance of the SD algorithm diminishes. The BER curves for these higher code rates indicate that a higher E_b/N_0 is needed to reach the same BER level as lower code rates. Additionally, for $R = 5/6$ and $R = 8/9$, the BER deteriorates more quickly as E_b/N_0 decreases, reflecting the greater difficulty

of achieving robust error correction at higher code rates due to reduced redundancy. Furthermore, introducing a noise-dependent scaling factor of 0.86 to the received channel probabilities has been observed to improve performance, especially at lower code rates like 1/3 and 2/5. On the other hand, this technique is less effective at higher code rates, such as 5/6 and 8/9, emphasizing the differing impact of this method across various code rates.

The comparative analysis reveals that while the SD algorithm provides comparable BER performance to SPA and MSA at a lower code rate (R = 1/3), it performs less effectively at higher code rates. Despite requiring more clock cycles for decoding, the SD algorithm's ability to achieve similar error correction with potentially simpler hardware implementation makes it an attractive alternative for certain applications. The performance trade-offs observed at higher code rates indicate the need for careful consideration of redundancy and error correction requirements in practical implementations.

4.5.3 FPGA implementation results for different code rates

Table 4.4: SD results of various code rates of $N = 3808$.

Active Code-rate	No.of clock cycles per DC	LUTs(k)	Slice registers(k)	Throughput(Mbps)	E_b/N_o at BER=10^{-6}	No.of DC per frame
1/3	68	8.2	1.7	953.4	2.65dB	620
2/5	68	8.2	1.7	964.3	2.69dB	530
1/2	68	8.2	1.9	1100.9	2.79dB	450
2/3	68	8.2	1.9	1189.3	3.28dB	430
3/4	68	8.2	1.9	1240.5	3.87dB	400
5/6	68	8.2	1.9	1267.6	4.02dB	360
8/9	68	8.2	1.9	1298.2	4.29dB	330

Table 4.4 provides an insightful comparison of the Stochastic Decoder (SD) performance across various code rates for a codeword length $N = 3808$. The data presented

includes hardware utilization, throughput, and error performance metrics for each code rate.

The table begins by listing the Active Code-Rate, which indicates the different code rates tested (1/3, 2/5, 1/2, 2/3, 3/4, 5/6, and 8/9). For each of these code rates, the Number of Clock Cycles per Decoding Cycle (DC) remains fixed at 68 cycles. This constancy highlights a stable processing requirement across all code rates.

In terms of hardware complexity, the Look-Up Tables (LUTs) and Slice Registers are crucial metrics. The number of LUTs used is consistently 8.2k for all code rates, signifying a steady hardware demand. However, the slice registers, which are essential for storing intermediate values, vary slightly, ranging from 1.7k to 1.9k. This slight increase corresponds to the higher demands of more complex decoding tasks at higher code rates.

The Throughput, measured in megabits per second (Mbps), shows a clear upward trend with higher code rates. Starting from 953.4 Mbps at a code rate of 1/3, the throughput increases progressively, reaching 1298.2 Mbps at a code rate of 8/9. This improvement is a direct result of reduced redundancy, allowing for faster data processing.

The E_b/N_0 at BER = 10^{-6} column indicates the energy efficiency of the decoder, with E_b/N_0 representing the energy per bit to noise power spectral density ratio. Lower code rates like 1/3 require 2.65 dB to achieve a BER of 10^{-6}, while higher code rates such as 8/9 require 4.29 dB. This trend shows that higher code rates necessitate more signal power to maintain the same level of error performance.

Finally, the Number of Decoding Cycles per Frame decreases as the code rate increases. At a code rate of 1/3, 620 decoding cycles are needed per frame, whereas at a code rate of 8/9, only 330 cycles are required. This reduction is due to the decreased redundancy and increased efficiency at higher code rates.

To summarize, Table 4.4 illustrates the performance and efficiency of the SD decoder across various code rates. The fixed number of clock cycles per DC and consistent LUT usage demonstrate stable hardware requirements. The slight increase in slice registers reflects the added complexity of higher code rates. The increasing throughput and E_b/N_0 values highlight the trade-offs between speed and signal power for different code rates,

while the reduced number of decoding cycles per frame indicates improved efficiency at higher rates. This comprehensive analysis underscores the effectiveness of the SD decoder in handling diverse decoding tasks.

4.5.4 Comparative analysis

Table 4.5: Comparative results.

Design	Proposed	[91]	[92]	[107]	[108]
Standard	5G-NR	5G-NR	802.16e	802.15.3c	802.11n
Code length	3808	3808	2304	672	1944
Base code rate	1/3	1/3	1/2	1/2	1/2
Decoding algorithm	SD	CMS	NMS	NMS	MS
Scheduling	Column-layered	Row-layered	Row- layered	Row-layered	Row-layered
Extrinsic message width	1-bit	4-bit	6-bit	4-bit	4-bit
Sub-matrix size	56	56	96	21	81
DCs or Itrs	620	10	10	5	10
Area (mm^2)	1.10	1.49	2.9	2.25	4.88
Throughput (Gbps)	1.12	3.04	2.20	5.28	4.5
Power (mW)	410	259	870	182	523

The comparative analysis of the proposed Stochastic Decoder (SD) architecture against other LDPC decoders, as outlined in Table 4.5, highlights the efficiency and effectiveness of the proposed design through various implementation and performance parameters.

The proposed SD design, adhering to the 5G-NR standard, features a code length of 3808 and a base code rate of 1/3. This is compared against other designs from references [91], [92], [107], and [108], which adhere to different standards like 802.16e, 802.15.3c, and 802.11n, with varying code lengths and base code rates.

The proposed design employs the Stochastic Decoding (SD) algorithm, whereas other designs utilize different algorithms, such as Combined Min-Sum (CMS), Normalized Min-Sum (NMS), and Min-Sum (MS). Moreover, the proposed design implements a column-layered scheduling approach, in contrast to the row-layered scheduling used by the other designs.

In terms of extrinsic message width, the proposed design operates with a 1-bit width, while the comparative designs vary from 4-bit to 6-bit. The sub-matrix size for the proposed design is 56, while other designs have sub-matrix sizes ranging from 21 to 96.

The number of decoding cycles (DCs) or iterations required by the proposed design is approximately 620, significantly higher than the 5 to 10 iterations required by the other designs. However, the proposed design demonstrates significant area efficiency, occupying only 1.10 mm² compared to the 1.49 mm² to 4.88 mm² occupied by the other designs. This represents a 26% improvement compared to [91], 62% compared to [92], 51.11% compared to [107], and 77.46% compared to [108].

In terms of throughput, the proposed design achieves 1.12 Gbps, which, while lower than the 2.20 to 5.28 Gbps achieved by the other designs, balances complexity and performance effectively. The proposed design excels in power consumption, using only 410 mW, significantly lower than the 259 to 870 mW used by the other designs. This indicates a 52% improvement in energy efficiency compared to [92] and a 21.6% improvement compared to [108].

The analysis underscores the proposed SD architecture's superior area efficiency and power consumption, despite slightly lower throughput compared to other designs. This detailed comparison demonstrates the proposed design's robust and optimized approach to 5G LDPC decoding, highlighting significant improvements and efficiency gains over traditional methods.

4.6 Conclusion

In this chapter, we delved into the design and implementation of a flexible FPGA-based stochastic decoder for rate-compatible 5G LDPC codes. The architecture addresses the intricate complexity of BGMs and meets the requirements for efficient error correction across a broad spectrum of code rates.

Our main innovation lies in the stochastic decoding method, which converts channel probability values into stochastic bit sequences. This transformation streamlines inter-node routing and significantly reduces the number of interconnects compared to traditional fixed-point decoders. The stochastic approach not only maintains error correction performance comparable to conventional methods like the SPA and MSA but also boosts efficiency.

Implemented on a Xilinx Kintex-7 FPGA board, our design integrates several key modules, including the SVND, SCND, and BGM ROMs. This setup demonstrates notable hardware efficiency, achieving substantial reductions in both the number of interconnects and overall hardware utilization.

Our performance analysis reveals that the proposed stochastic decoder offers competitive BER performance, particularly at lower code rates, with results comparable to SPA and MS algorithms. Despite requiring more decoding cycles, the decoder achieves a throughput of approximately 1.12 Gbps, effectively balancing performance with reduced hardware complexity. Additionally, the design exhibits improved energy efficiency, consuming less power than conventional decoders.

In a comparative analysis with other state-of-the-art decoders, our design showcases significant advantages in terms of area efficiency and power consumption while maintaining competitive throughput and error correction performance.

Overall, the proposed FPGA-based stochastic decoder presents a flexible, efficient, and scalable solution for 5G LDPC decoding. Its ability to handle a wide range of code rates with reduced hardware complexity and enhanced energy efficiency positions it as a robust alternative to traditional fixed-point decoders. This work establishes a solid foundation for future research and development in high-performance, resource-efficient LDPC decoding for next-generation communication standards.

5 CNN Approach for 5G LDPC decoding

5.1 INTRODUCTION

Fifth-Generation (5G) [3] technology supports Internet of Things (IoT) applications by delivering high data speeds, low latency, and huge connectivity [4]. 5G [16] [109] is built to handle up to one million devices per square kilometer and enables massive machine-type communications (mMTC) [110], which is critical for IoT applications like smart cities, agriculture, and industrial automation [1]. Furthermore, URLLC [111] in 5G supports essential IoT applications such as remote surgery, autonomous driving, and industrial automation, which require extremely low latency and great reliability, with latency as low as 1 millisecond. Enhanced mobile broadband (eMBB) [112] provides faster data rates and more bandwidth, enabling high-definition video streaming and real-time data processing

for IoT applications such as smart surveillance and augmented reality.

Smart cities benefit from 5G-enabled IoT applications such as smart traffic management, surveillance systems, waste management, and environmental monitoring. In the automotive and transportation sectors, 5G's low latency and high reliability enable vehicle-to-everything communication, improving safety and autonomous driving capabilities. Precision farming uses IoT sensors and 5G connectivity for real-time monitoring of soil conditions, crop health, and livestock management. Energy management also benefits from 5G for balancing supply and demand, integrating renewable energy sources, and reducing energy waste.

In 5G supported IoT applications, the utilization of LDPC codes for channel coding guarantees reliable data transmission [113]. They are particularly popular in 5G New Radio (NR) because of their capacity to handle massive data rates while maintaining high reliability [19].

5.1.1 Urgency of dealing with colored noise

Colored noise [34], unlike white noise which has a flat spectral density, exhibits a spectral density that varies with frequency. In the context of 5G enabled IoT networks, colored noise [114] introduces varying levels of interference across different frequency bands, which can significantly degrade the performance of LDPC decoding. This degradation manifests as higher bit error rates and complications in the decoding process due to signal distortion in specific frequency bands. The presence of Colored noise [115] in the communication channel reduces communication system performance and reliability significantly in IoT applications enabled by 5G. Here are a few sources of colored noise:

- Natural sources: Natural sources of colored noise include thermal noise, cosmic background radiation, and atmospheric noise.
- Man-Made Sources: Human activities, industrial machinery, and electronic gadgets all produce electromagnetic interference, which may add colorful noise into a communication channel.

- Electronic Devices: Microwaves, Bluetooth devices, and other wireless equipment can produce colored noise through their operating frequencies.

Our research is driven by several key objectives:

- In the context of smart cities, where many sensors and equipment are used to monitor and control urban infrastructure, the use of effective LDPC decoding techniques ensures that data may flow smoothly even when there is interference from colored noise.
- In the field of industrial automation [116] [117], where the ability to monitor and control processes in real-time is crucial, the robustness of LDPC decoding against colored noise is essential in preventing any disturbances that could result in expensive periods of inactivity.
- In healthcare, effective LDPC decoding enables dependable data transfer for applications such as remote patient monitoring and telemedicine, where fast and accurate data is critical.
- Similarly, in smart agriculture, where sensors monitor soil conditions, crop health, and animals, reducing the effects of colored noise ensures information accuracy and reliability, resulting in better resource utilization and higher yields.

Each of these IoT applications has specific needs for data rates, and dependability. Effective LDPC decoding in the presence of colored noise meets these objectives by ensuring high-quality communication links.

The study is based on the following hypotheses:

- Improved Error Correction and Data Integrity: Artificial intelligence (AI)-driven 5G channel decoding will greatly enhance error correction capabilities, ensuring higher data integrity and dependability for IoT applications, especially in the presence of colored noise.

- Enhanced Adaptive Coding: Based on the channel conditions, adaptive coding approaches optimizes real-time coding rate modifications, resulting in better IoT network performance and efficiency under varied colored noise situations.

Researchers are exploring deep learning techniques on digital communication issues, like channel coding. An innovative iterative BP-CNN design was introduced to improve the channel's E_b/N_o in the presence of correlated noise. This research inspired the developing of an SD-CNN architecture to optimize the decoding capability of 5G LDPC codes. The architecture combines a trained CNN with a Stochastic decoder, with received symbols processed iteratively between the two. The Stochastic decoder processes the received symbols to acquire the initial decoding results, including the estimated channel noise. This noise passes into a CNN to reduce estimation errors and get better noise estimation. Iterating between the SD and CNN gradually enhances the E_b/N_o and results in improved decoding.

5.2 Preliminaries

5.2.1 Colored noise

In an elementary communication system model centered on channel coding, a message block vector $\mathbf{m}$, comprising K bits, is encoded to generate a codeword $\mathbf{c}$. This codeword is a N-bit vector obtained by appending $(N-K)$ parity bits to the initial message block. Utilizing BPSK modulation, the modulator produces a modulated symbol vector $\mathbf{x}$. This vector is then transmitted through the communication channel, resulting in the received signal vector $\mathbf{y} = \mathbf{x} + \eta$, where η represents the noise vector added by the channel, also consisting of N bits.

In environments where noise is correlated, such as in the case of colored noise , an Auto-regressive (AR) process of order 1 is frequently utilized to model the noise characteristics. In this AR(1) model, each noise value in the time series is a linear function of its immediate predecessor, combined with a white noise component. Consequently, this results in a noise spectrum that deviates from a flat (uniform) distribution, exhibiting

frequency-dependent characteristics. Mathematically, the noise value η_s at time step s is given by the equation:

$$\eta_s = \beta \times \eta_{s-1} + w_s. \tag{5.1}$$

In this equation, β is the autoregressive correlation coefficient that determines the influence of the previous noise value η_{s-1} on the current noise value η_s. The term w_s represents the white noise component, which is a random variable with zero mean and constant variance σ_w^2, added to introduce randomness at each time step. This AR(1) process creates a sequence of noise values where each value is correlated with its predecessor, leading to the characteristic non-flat spectral density of colored noise. In this process, if we assume the noise is stationary, the variance σ_n^2 and the covariance between any two samples η_i and η_j can be described as follows:

$$\sigma_n^2 = \frac{\sigma_w^2}{1-\beta^2}, \tag{5.2}$$

$$\text{Cov}(\eta_i, \eta_j) = \beta^{|i-j|} \times \sigma_n^2. \tag{5.3}$$

Here, σ_w^2 represents the variance of the white noise component, and β is the correlation coefficient. The variance σ_n^2 of the noise process is determined by the white noise variance adjusted for the correlation coefficient. The covariance between two noise samples η_i and η_j is influenced by the correlation coefficient β raised to the power of the absolute difference in their indices $|i-j|$, scaled by the noise variance σ_n^2. Therefore, the entire sequence $\eta = [\eta_1, \eta_2, ..., \eta_N]^T$ forms a Gaussian column vector characterized by its mean and covariance matrix.

5.2.1.1 Covariance matrix

The covariance matrix, denoted by $\mathscr{T}$, is defined as:

$$\mathscr{T}_{ij} = \beta^{|i-j|} \times \sigma_n^2. \tag{5.4}$$

The equation (5.4) illustrates that the correlation between any two noise values η_i and η_j decays exponentially with the distance $|i-j|$ between their respective time indices. The correlation is modulated by the correlation coefficient β, indicating how quickly the influence of one noise value diminishes as the separation between samples increases. The structure of the covariance matrix reflects this exponential decay, ensuring that closer samples in time are more strongly correlated than those further apart. To elaborate:

1. Diagonal elements of $\mathscr{T}$ when $(i = j)$:

   $\mathscr{T}_{ii} = \sigma_n^2$ indicates that the variance of each noise sample η_i is σ_n^2. This is because the diagonal elements represent the variance of the noise samples themselves, incorporating the full variance of the white noise component.

2. Off-Diagonal elements of $\mathscr{T}$ when $(i \neq j)$:

   - The off-diagonal terms $\mathscr{T}_{ij}$ illustrate how the correlation between two different noise samples decreases as the distance $|i-j|$ increases. The correlation coefficient β raised to the power of the absolute difference $|i-j|$ determines the strength of this correlation.
   - If β is near 1, the correlations decay slowly, meaning that past noise values have a strong influence on future values, even if they are further apart.
   - Conversely, if β is near 0, the correlations decay rapidly, resulting in noise samples n_i and n_j being almost independent when separated by a larger distance.

This framework shows how the covariance matrix captures both the variance of individual noise samples and the way correlations between different samples decrease with increasing separation, depending on the value of the correlation coefficient β.

The covariance matrix $\mathscr{T}$ has several important properties:

1. Positive Definiteness and Invertibility:

   The covariance matrix $\mathscr{T}$ is positive definite if $|\beta| < 1$. This positive definiteness is essential because it guarantees that $\mathscr{T}$ can be inverted, a requirement for numerous applications in statistics and signal processing, such as LLR calculations.

2. Symmetry:

   $\mathscr{T}$ is symmetric, meaning $\mathscr{T}_{ij} = \mathscr{T}_{ji}$. Symmetry is a fundamental characteristic of covariance matrices, which simplifies many mathematical manipulations and computational techniques.

3. Toeplitz Structure:

   The covariance matrix $\mathscr{T}$ is a Toeplitz matrix because each off-diagonal element depends solely on the distance between indices rather than the specific indices themselves. This means that the elements along each diagonal are constant. The Toeplitz structure is advantageous because it allows for the use of efficient algorithms in matrix computations.

By ensuring positive definiteness, symmetry, and leveraging the Toeplitz structure, the covariance matrix $\mathscr{T}$ supports efficient and effective mathematical and computational operations, enhancing its utility in various applied fields.

For a scenario involving four samples, the covariance matrix $\mathscr{T}_4$ can be expressed as:

$$\mathscr{T}_4 = \begin{bmatrix} \sigma_1^2 & \beta\sigma_1\sigma_2 & \beta^2\sigma_1\sigma_3 & \beta^3\sigma_1\sigma_4 \\ \beta\sigma_1\sigma_2 & \sigma_2^2 & \beta\sigma_2\sigma_3 & \beta^2\sigma_2\sigma_4 \\ \beta^2\sigma_1\sigma_3 & \beta\sigma_2\sigma_3 & \sigma_3^2 & \beta\sigma_3\sigma_4 \\ \beta^3\sigma_1\sigma_4 & \beta^2\sigma_2\sigma_4 & \beta\sigma_3\sigma_4 & \sigma_4^2 \end{bmatrix},$$

where σ_1^2, σ_2^2, σ_3^2, and σ_4^2 are the variances of the four noise samples, and β denotes the correlation coefficient between adjacent samples. If we assume $\beta = 0.6$ and that the variances of all noise samples are equal to one, the covariance matrix simplifies to:

$$\mathscr{T}_4 = \begin{bmatrix} 1 & 0.6 & 0.36 & 0.216 \\ 0.6 & 1 & 0.6 & 0.36 \\ 0.36 & 0.6 & 1 & 0.6 \\ 0.216 & 0.36 & 0.6 & 1 \end{bmatrix}.$$

For an Auto-regressive process of order 1 (AR(1)), the covariance matrix $\mathscr{T}$ takes on a

Toeplitz structure, characterized by constant diagonals. It is given by:

$$\mathscr{T} = \sigma_n^2 \begin{bmatrix} 1 & \beta & \beta^2 & \cdots & \beta^{N-1} \\ \beta & 1 & \beta & \cdots & \beta^{N-2} \\ \beta^2 & \beta & 1 & \cdots & \beta^{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta^{N-1} & \beta^{N-2} & \beta^{N-3} & \cdots & 1 \end{bmatrix},$$

where N represents the length of the noise series. This Toeplitz matrix structure reflects how each element $\mathscr{T}_{ij}$ is determined by the correlation coefficient β raised to the power of the absolute difference between indices $|i-j|$, scaled by the variance σ_n^2.

5.2.2 Calculating LLR when Colored Noise is present

The received signal at time s when transmitting BPSK symbols $x_s \in \{+1,-1\}$ is expressed as:

$$y_s = x_s + \eta_s. \tag{5.5}$$

Here, η_s represents the noise at time s.

To calculate the LLR for the symbol x_s, we use the following formula:

$$L_{v_s} = \log \frac{P(x_s = +1 \mid y_s)}{P(x_s = -1 \mid y_s)}. \tag{5.6}$$

Assuming that the noise η_s follows an AR(1) process, and considering the Gaussian probability distribution functions, we have:

$$\begin{aligned} P(y_s \mid x_s = \pm 1) = & \frac{1}{\sqrt{2\pi \det(\mathscr{T})}} \times \\ & \exp\left(-\frac{1}{2}(y_s \mp 1)^T \mathscr{T}^{-1}(y_s \mp 1)\right). \end{aligned} \tag{5.7}$$

Here, $\mathscr{T}$ is the covariance matrix of the noise.

The LLR calculation then becomes:

$$L_{v_s} = -\frac{1}{2}\left[(y_s - \mathbf{1})^T \mathscr{T}^{-1}(y_s - \mathbf{1})\right] + \frac{1}{2}\left[(y_s + \mathbf{1})^T \mathscr{T}^{-1}(y_s + \mathbf{1})\right]. \tag{5.8}$$

Simplifying this expression, we get:

$$L_{v_s} = -\frac{1}{2}\left[y_s^T \mathscr{T}^{-1} y_s - 2y_s^T \mathscr{T}^{-1}\mathbf{1} + \mathbf{1}^T \mathscr{T}^{-1}\mathbf{1}\right] + \frac{1}{2}\left[y_s^T \mathscr{T}^{-1} y_s + 2y_s^T \mathscr{T}^{-1}\mathbf{1} - \mathbf{1}^T \mathscr{T}^{-1}\mathbf{1}\right]. \tag{5.9}$$

After further simplification, the LLR is:

$$L_{v_s} = 2y_s^T \mathscr{T}^{-1}\mathbf{1}. \tag{5.10}$$

In this final expression, $\mathscr{T}^{-1}$ is the inverse of the covariance matrix of the AR(1) noise process, and $\mathbf{1}$ is a column vector of ones. This formulation allows for the efficient calculation of LLR in the presence of colored noise, leveraging the properties of the covariance matrix.

5.3 Proposed method

The proposed SD-CNN design, as illustrated in Figure 5.1, integrates a Flexible Stochastic Decoder (SD) with a CNN to enhance the decoding process of LDPC codes over noisy channels. This section elaborates on the components and workflow of the proposed design, highlighting the role of each element in the decoding chain. Components and Workflow:

1. LDPC Encoder: The LDPC encoder is responsible for transforming the input message m into a codeword c using the LDPC coding scheme. This process involves adding redundancy to the original data, which is crucial for error correction. By incorporating additional bits, the LDPC encoder ensures that the encoded message

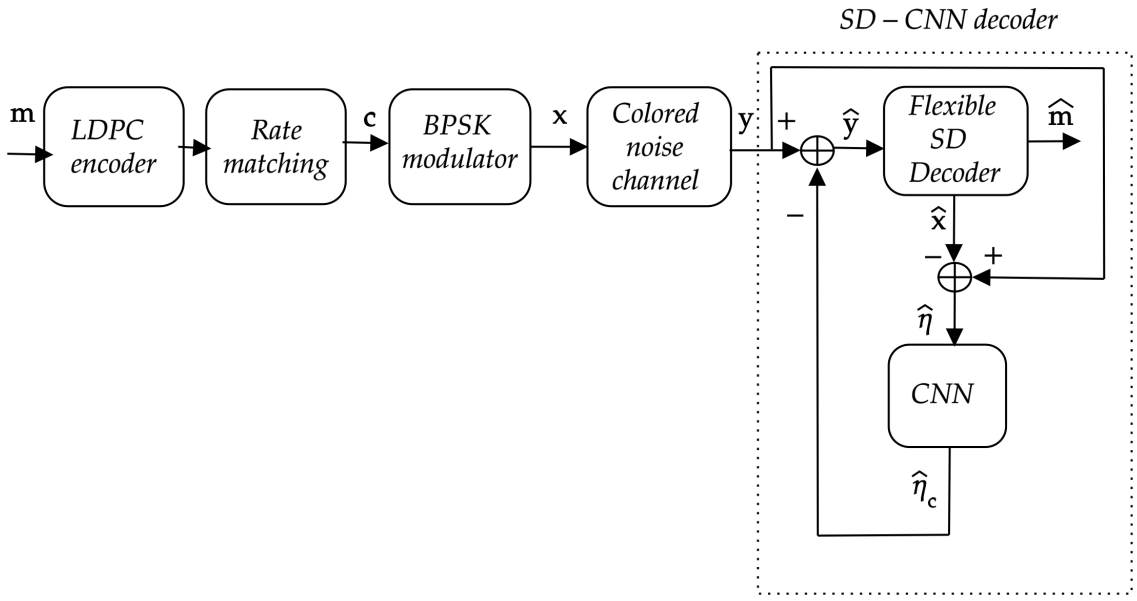


Figure 5.1: Proposed SD-CNN design.

can be accurately reconstructed even if parts of it are corrupted during transmission. This redundancy allows the decoder to identify and correct errors, thus improving the reliability of data communication.

2. Rate Matching: The rate matching block modifies the encoded codeword c to align with the specified transmission rate. This adjustment ensures that the data is transmitted at the correct rate for the given communication channel. For example, the rate matching process might involve puncturing, where certain bits are removed, or repeating, where certain bits are duplicated, to achieve the desired code rate. This step is essential for optimizing the data flow and maintaining efficient use of the available bandwidth during transmission.

3. BPSK Modulation : The BPSK modulation block converts the rate-matched codeword c into a modulated signal x. This step involves encoding the binary data into two distinct phases of a carrier signal, with bits '0' and '1' represented by different phases. For example, a bit '0' might be represented by a 0-degree phase, while a bit '1' is represented by a 180-degree phase. This modulation technique ensures that the data can be effectively transmitted over the communication channel while maintaining robustness against noise.

4. Colored Noise Channel: The colored noise channel transmits the modulated signal x, resulting in the received signal y. This channel is characterized by colored noise, which can introduce specific frequency-dependent distortions to the signal. For example, certain frequencies may be more affected by noise than others, potentially altering the signal's integrity. This distortion necessitates robust error correction by the decoder to accurately recover the original data.

5. SD-CNN Decoder: The SD-CNN decoder integrates two key components to enhance decoding accuracy: the flexible stochastic decoder and the Convolutional Neural Network (CNN). Flexible SD component receives the noisy signal y and processes it to produce an initial estimate $\hat{y}$ of the transmitted signal. By employing stochastic decoding techniques, the decoder iteratively refines this estimate to mitigate the impact of noise introduced during transmission. The CNN further processes the estimate $\hat{y}$ from the stochastic decoder, generating a refined noise estimate $\hat{n}$ and producing a corrected codeword $\hat{m}$. By leveraging its ability to learn noise characteristics, the CNN significantly enhances error correction, reducing residual errors and improving overall decoding performance after the initial stochastic decoding phase.

6. Feedback Loop : The feedback loop is essential for improving the precision of the SD-CNN decoder. This loop allows the noise estimate $\hat{n}$ generated by the CNN to be subtracted from the received signal y, creating a new input $\hat{x}$ for the flexible stochastic decoder. The process is iterative, continually refining the signal estimate. The feedback loop's primary aim is to iteratively boost decoding accuracy by utilizing both the stochastic decoder and CNN capabilities. By refining the signal estimate in successive iterations, the decoder can achieve higher accuracy. During each iteration, the feedback loop helps to reduce the BER, gradually bringing the estimated signal closer to the original transmitted message. This iterative refinement process continues until the decoder achieves a reliable estimate, ensuring effective error correction.

Consider a scenario where an LDPC codeword is transmitted over a noisy channel. The

received signal y is first processed by the flexible SD decoder, which provides an initial estimate $\hat{y}$. This estimate is then fed into the CNN, which refines the estimate and generates $\hat{n}$ and $\hat{m}$. The refined noise estimate $\hat{n}$ is subtracted from y, and the process repeats iteratively, each time improving the decoding accuracy and reducing errors. This combination of stochastic decoding and neural network refinement ensures robust performance even in highly noisy environments.

5.3.1 A Flow chart for SD-CNN design

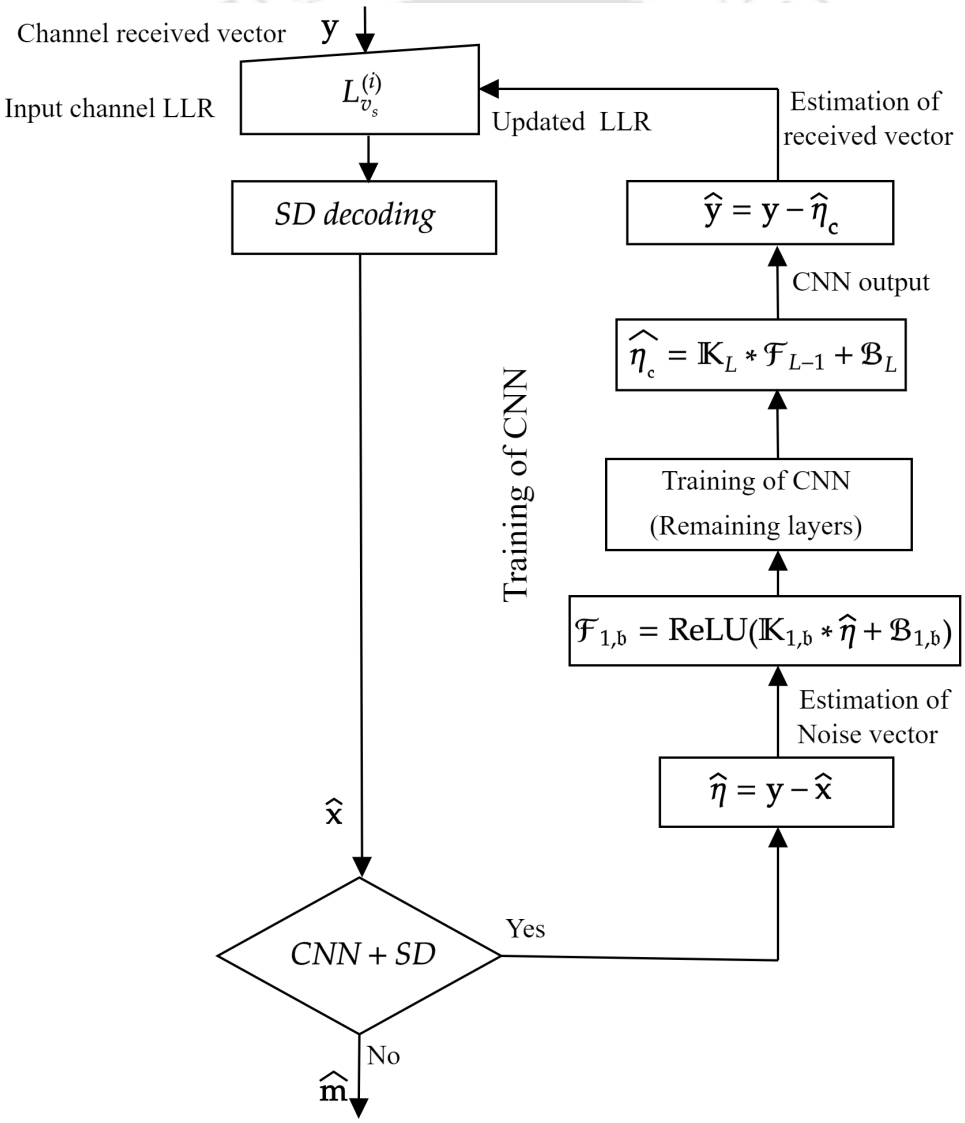


Figure 5.2: SD-CNN flowchart.

The flow chart for the SD-CNN design process is depicted in Figure 5.2, beginning at

the receiver side of the communication system. The process involves the following steps:

1. Signal Reception and LLR Calculation: Upon receiving the signal $\mathbf{y}$, the Log-Likelihood Ratios (LLRs) of the symbols are calculated as L_{v_s}.

2. SD Decoder Input: The calculated LLR values for all symbols are then provided as input to the SD decoder, which estimates the symbol vector $\widehat{\mathbf{x}}$.

3. Channel Noise Estimation: Due to decoding errors, the estimated channel noise $\widehat{\eta}$ does not exactly match the actual channel noise η. The estimated channel noise is given by $\widehat{\eta} = \eta + \mathbf{e}$, where $\mathbf{e}$ is the noise error vector.

4. CNN Processing: The estimated channel noise $\widehat{\eta}$ is fed into a trained CNN, which outputs $\widehat{\eta_c}$. The CNN leverages the correlation property of the channel noise η as a feature to effectively suppress the error $\mathbf{e}$ and accurately estimate the channel noise.

5. Signal Adjustment: The CNN output $\widehat{\eta_c}$ is subtracted from the received signal $\mathbf{y}$ to calculate the new vector $\widehat{\mathbf{y}}$:

$$\widehat{\mathbf{y}} = \mathbf{y} - \widehat{\eta_c} = \mathbf{x} + \eta - \widehat{\eta_c} = \mathbf{x} + \mathbf{r_n}, \tag{5.11}$$

   where $\mathbf{r_n} = \eta - \widehat{\eta_c}$ represents the residual noise.

6. SD decoding: The updated vector $\widehat{\mathbf{y}}$ is sent through the SD decoder for another round of decoding. Before this second round, the LLR values need to be updated to $L_{v_s}^{(2)}$, where the superscript (2) indicates the LLR's value after CNN processing.

The properties of $\mathbf{r_n}$ affect the computation of the updated LLR values and the efficiency of the subsequent SD decoding process. Therefore, it is crucial to train the CNN to accurately estimate the channel noise by effectively suppressing $\mathbf{r_n}$. This is achieved by adopting a custom cost function $\mathscr{C}$ as shown in (5.15). By organizing and detailing these steps, the SD-CNN design process effectively improves the accuracy of channel noise estimation and enhances the performance of the decoding process in the presence of noise.

5.3.2 CNN structure for noise estimation

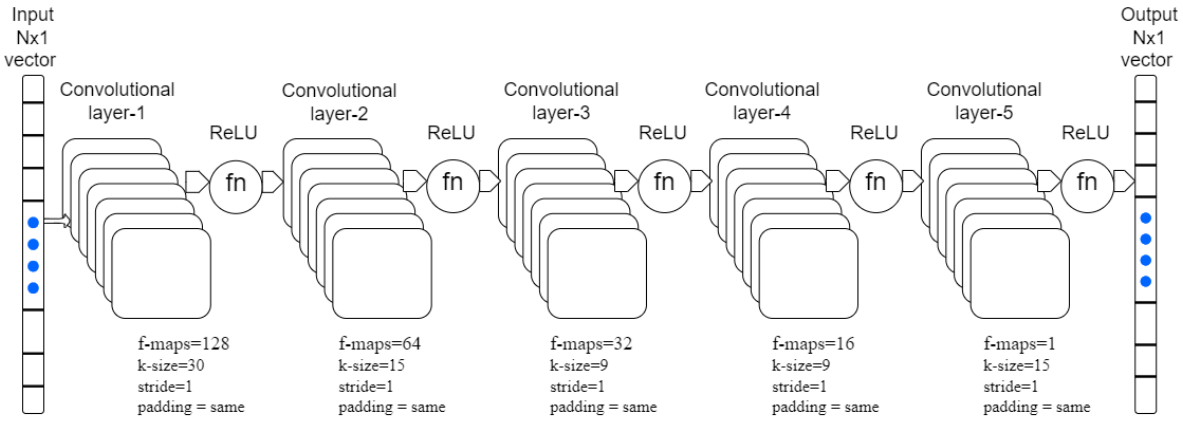


Figure 5.3: Proposed CNN structure.

The proposed SD-CNN design utilizes a 1-D CNN, a variation of the standard CNN tailored for processing one-dimensional data. Figure 5.3 illustrates the architecture of this CNN, specifying the number of layers, kernel sizes, and feature maps within each layer. Before training the network, certain parameters need to be determined. Unlike most CNN designs, our architecture does not include fully connected, dropout, or max pooling layers, as the output size remains the same as the input size, maintaining a low-level representation.

Our adopted CNN comprises four convolutional layers. The operation of a 1-D convolutional layer involves applying kernels to the input data, which, in this case, is a one-dimensional sequence. After SD decoding, the estimated channel noise $\widehat{\eta}$ is fed into the CNN as a 1-D vector with dimensions 3808×1. Each convolutional layer contains a set of learnable kernels or filters. These kernels are small segments of weights that slide across the input data, performing element-wise multiplication and summation. This convolution operation identifies specific patterns or features within the input data, producing a new array known as the ”feature map.”

Typically, a convolutional layer has multiple filters, each detecting different features or patterns. The feature map of the $\mathfrak{b}$-th filter in the $\mathfrak{a}$-th layer, denoted as $\mathscr{F}_{\mathfrak{a},\mathfrak{b}}$, is calculated as follows:

1. First Layer:

$$\mathscr{F}_{1,\mathfrak{b}} = \text{ReLU}(\mathbb{K}_{1,\mathfrak{b}} * \widehat{\eta} + \mathscr{B}_{1,\mathfrak{b}}). \tag{5.12}$$

2. Intermediate Layers:

$$\mathscr{F}_{\mathfrak{a},\mathfrak{b}} = \text{ReLU}(\mathbb{K}_{\mathfrak{a},\mathfrak{b}} * \mathscr{F}_{\mathfrak{a}-1} + \mathscr{B}_{\mathfrak{a},\mathfrak{b}}). \tag{5.13}$$

3. Last Layer:

$$\widehat{\eta_c} = \mathbb{K}_L * \mathscr{F}_{L-1} + \mathscr{B}_L. \tag{5.14}$$

Here, $\mathbb{K}_{\mathfrak{a},\mathfrak{b}}$ represents the $\mathfrak{b}$-th kernel of the $\mathfrak{a}$-th layer, and $\mathscr{B}_{\mathfrak{a},\mathfrak{b}}$ denotes the $\mathfrak{b}$-th bias of the $\mathfrak{a}$-th layer. After the convolution process, non-linearity is introduced using an activation function like the Rectified Linear Unit (ReLU) [118]. The output of each convolutional layer is a set of feature maps, with each map representing the presence of a specific pattern or feature detected by the filters.

This convolutional process is repeated for each filter in the layer, resulting in a variety of feature maps that capture distinct aspects of the input data.

5.3.3 Custom cost function

Custom cost functions are tailored to address specific requirements in a learning task, evaluating the difference or error between predicted outputs and actual target values during training. In our design, the CNN is used to estimate channel noise, which influences the performance of the subsequent SD decoding. Therefore, selecting an appropriate cost function that considers the relationship between the CNN and the SD decoder is crucial. Our custom cost function is defined as:

$$\mathscr{C} = \frac{||\mathbf{r_n}||^2}{N} + \rho\left(S^2 + \frac{1}{4}(\mathscr{K} - 3)^2\right). \tag{5.15}$$

Here, $||\mathbf{r_n}||^2$ is the squared norm of the residual noise, where $\mathbf{r_n} = \eta - \widehat{\eta_c}$. η is the actual channel noise, and $\widehat{\eta_c}$ is the noise estimated by the CNN. N is the length of the

codeword, ρ is a scaling factor, S is the skewness, and $\mathscr{K}$ is the kurtosis of the training data. Explanation of the Cost Function Components:

- Residual Noise Minimization: The term $\frac{\|\mathbf{r_n}\|^2}{N}$ aims to reduce the average power of the residual noise across all observations, pushing the network towards accurate noise estimation.
- Normality Regularization: The term $\rho\left(S^2+\frac{1}{4}(\mathscr{K}-3)^2\right)$ is derived from the Jarque-Bera test [119] and penalizes deviations from a normal distribution. Skewness Squared (S^2) Penalizes asymmetry in the noise estimates. Excess Kurtosis Squared ($\frac{1}{4}(\mathscr{K}-3)^2$) Penalizes deviations from the normal level of tailedness (kurtosis).

Steps to Implement the Custom Cost Function:

1. Compute Residual Noise: Calculate the difference between actual and estimated noise for each sample
2. Calculate Skewness and Kurtosis: Use statistical functions to compute the skewness and kurtosis of the estimated noise over the training batch.
3. Combine in Cost Function: Use the defined cost function $\mathscr{C}$ to compute the loss during each training step.
4. Optimize: Apply an optimization algorithm (like SGD or Adam) to minimize $\mathbf{r_n}$ across training epochs, adjusting the network weights accordingly.

Formula for Skewness (S):

$$S=\frac{\frac{1}{N}\sum_{i=1}^{N}(r_i-\bar{r})^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(r_i-\bar{r})^2\right)^{3/2}}. \tag{5.16}$$

Where r_i are the individual observations, and $\bar{r}$ is the mean of the observations.

Formula for Kurtosis ($\mathscr{K}$):

$$\mathscr{K}=\frac{\frac{1}{N}\sum_{i=1}^{N}(r_i-\bar{r})^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(r_i-\bar{r})^2\right)^{2}}. \tag{5.17}$$

This formula measures the tailedness of the distribution: higher kurtosis indicates heavier tails, while lower kurtosis indicates lighter tails. By integrating these terms, the custom cost function effectively minimizes residual noise while ensuring the estimated noise distribution closely matches a normal distribution.

5.3.4 Channel noise samples generation for CNN training

To ensure effective training of the network, both the estimated noise $\widehat{\eta}$ from SD decoding and the actual channel noise η are necessary. By focusing on the receiver-recognized noise correlation functions of channel models [120], we can generate appropriate channel noise samples for training. These samples are produced using the following relationship:

$$\eta = \mathscr{T}^{\frac{1}{2}} \eta_g. \tag{5.18}$$

Here, η_g is a vector of Gaussian random variables that are independently and identically distributed (i.i.d.) with a standard distribution. This approach leverages the known correlation functions to create realistic noise samples, which are crucial for training the network to accurately estimate and mitigate channel noise.

5.3.5 Convolution Operation in CNNs

In CNNs, the convolution operation applied to colored noise data involves sliding a filter (kernel) across the input noise vector. For each filter position, a dot product is computed between the filter weights and the corresponding segment of data. Suppose a filter has weights $[w_1, w_2, \ldots, w_k]$ of size k, and at time t, it covers the segment $[\eta_{t-k+1}, \ldots, \eta_t]$. The convolution at this position is calculated as:

$$C_t = w_1 \cdot \eta_{t-k+1} + w_2 \cdot \eta_{t-k+2} + \cdots + w_k \cdot \eta_t. \tag{5.19}$$

The resulting value, C_t, contributes to one element of the feature map. After computing the convolution, the Rectified Linear Unit (ReLU) activation function is applied to the

output:

$$U_t = \max(0, C_t). \tag{5.20}$$

This combination of convolution and ReLU activation effectively highlights regions of autocorrelation in the time series. For an auto-regressive (AR) series where the correlation coefficient β is close to 1, successive values are similar and positive. If the filter weights are designed to detect this similarity (e.g., all positive weights), the convolution C_t at positions where η_t and η_{t-1} are both positive will also be positive and significant, thus passing through the ReLU activation. As a result, U_t will be high in regions with strong positive autocorrelation, highlighting these areas in the feature map.

5.4 Implementation results

5.4.1 Approach

To validate the design, the 5G LDPC code word length N = 3808 of BGM1 with base code rate $R = 1/3$ is used. Google Colab, MATLAB, and Tensor Flow [121] are used to build the simulation platform. A neural network's weights can be initialized [122] using a method called the He Normal initialization. This method initializes the weights in such a way as to facilitate the neural network's capacity for efficient learning. Prior to training the network on the training data, it must be generated. To test the network cost function and prevent possible over fitting, some validation data is also generated, as is standard procedure in machine learning. The training data are produced using a variety of signal-to-noise power (SNR)s, including 0 dB, 0.5 dB, 1 dB, 1.5 dB, 2 dB, 2.5 dB, 3 dB, 3.5 dB and 4 dB. Each SNR's data takes up the same percentage of the total data. The network is trained using a classic mini-batch gradient descent approach. There are 1200 blocks of data in each mini-batch, and each SNR's data takes up the same amount of space. To find the gradient, a fixed number of training samples, called a "mini-batch," are picked at random in each iteration. When searching for the optimal parameters for the network, we make use of the Adaptive Moment Estimation (Adam) optimization method [123]. Training is

continued in our simulation until the loss does not reduce over an extended length of time. The CNN parameters and their values are described in the Table 5.1 . One measure of a system's performance is its bit error rate (BER). This measurement takes into account the channel's SNR E_b/N_o at a specified BER of 10^{-6}, which is the decoder's transmission energy efficiency. One hundred frame errors per BER measurement is required by the simulations.

Table 5.1: CNN parameters.

Parameter	Value
Number of Layers	5
Filter Size	{30, 15, 9, 9, 15 }
Number of Feature maps	{128, 64, 32, 16, 1}
Padding	Not applicable
Pooling layers	Not applicable
Fully Connected Layers	Not applicable
Activation Function	ReLU
Optimizer	Adam
Loss Function	$\frac{\|\|r\|\|^2}{N} + \rho\left(S^2 + \frac{1}{4}(C-3)^2\right)$
Weight Initialization	He Normal
Training data size	17,50,000
Testing data size	80,000
Accuracy	94.5%
Number of epochs	30

5.4.2 Evaluation of performance

The performance of the design depends on three parameters such as the correlation coefficient $\mathscr{C}$, the scaling factor ρ, and the number of repetitions between SD and CNN.

5.4.2.1 The choice of correlation coefficient

This study focuses on three scenarios in which the correlation coefficient of the channel noise $\mathscr{C}$ was either 0.9 indicating a high correlation, 0.6 indicating a moderate correlation, or 0 indicating no correlation. Simulation results focused on the suggested one iteration of the SD-CNN decoder (SD-CNN)1. As shown in Fig. 5.4, when $\mathscr{C} = 0.9$, the decoding performance of SD-CNN improved by about 2.9 dB at a BER of 10^{-6}. The performance gain

is smaller when the correlation is moderate $\mathscr{C} = 0.6$. It demonstrates that when the correlation is weaker, using a CNN has less impact. When $\mathscr{C} = 0$, the proposed design works similarly to standard stochastic decoding (SD), in which the noise samples are i.i.d. The results lead us to conclude that the iterative SD-CNN decoding method works well across various correlation coefficients, with performance gains that vary adaptively depending on the correlation of the noise. The performance gain improves steadily as correlation level $\mathscr{C}$, as expected, because a higher $\mathscr{C}$ allows CNN to extract the correlation of noise as a feature. The performance gain of the proposed decoder increases when there is an increase in the correlation coefficient of the channel noise $\mathscr{C}$ as shown in Fig. 5.5.

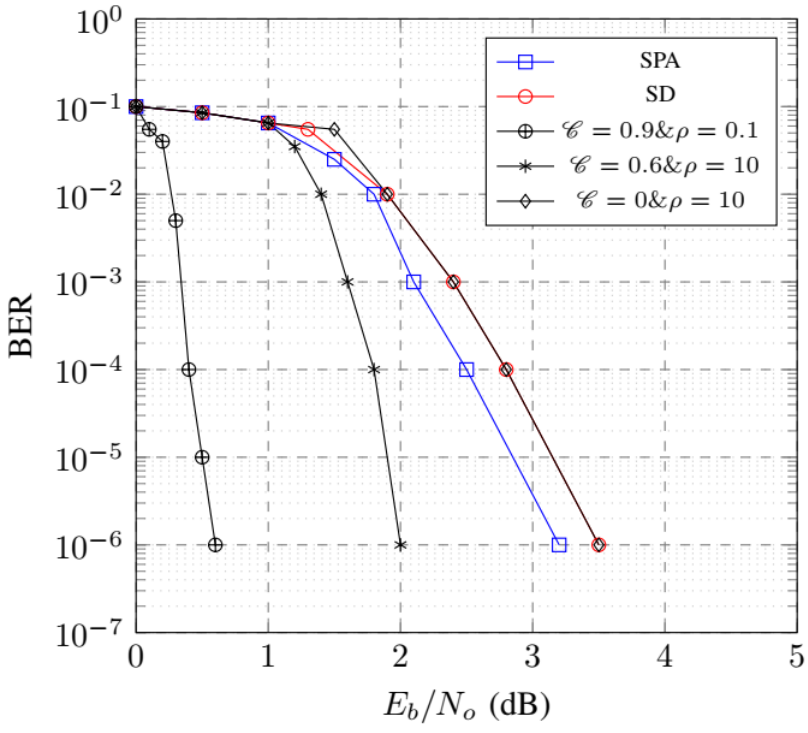


Figure 5.4: BER plot of various algorithms.

5.4.2.2 The selection of Scaling factor

The study presents a custom cost function T that measures the normality component and power of residual noise $\mathbf{r_c}$. The value of ρ in T, plays a significant role in achieving a balance between the power of $\mathbf{r_c}$ and its distribution. From simulations, it has been observed that very little values of ρ are unable to ensure a Gaussian distribution for the $\mathbf{r_c}$, while extremely high values of ρ are unable to minimize the residual noise power.

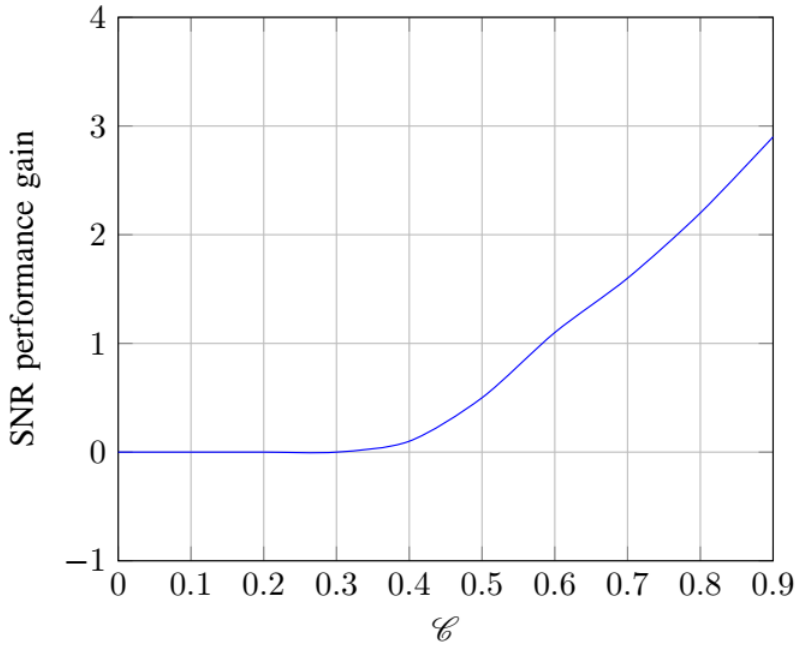


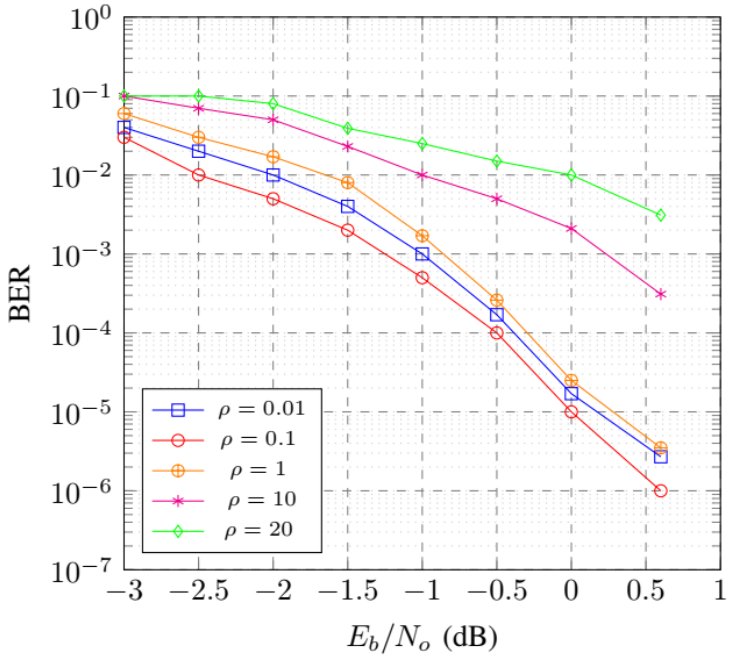
Figure 5.5: Relation between performance gain and correlation coefficient.

Therefore, an appropriate selection of ρ affects the performance of SD-CNN.

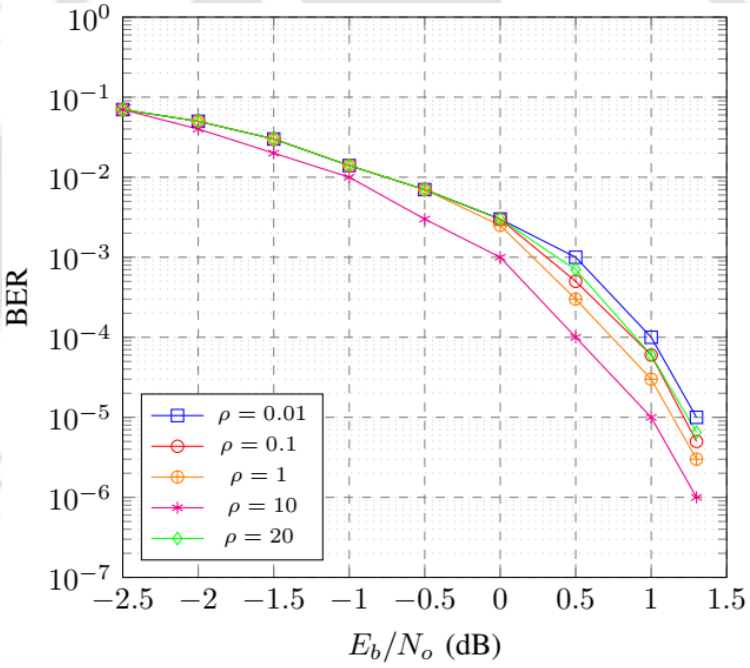
The results of adopting various values of the scaling factor ρ at two distinct channel correlation coefficients $\mathscr{C}$ = 0.9 and 0.6, using an architecture of (SD-CNN)1, are shown in the Fig. 5.6. The Table 5.2 displays the results for two different values of $\mathscr{C}$. When $\mathscr{C}$ is 0.9, a Bit Error Rate (BER) of 10^{-6} is reached at ρ = 0.1. On the other hand, when $\mathscr{C}$ is 0.6, a BER of 10^{-6} is achieved at ρ = 10. The value ρ = 0.1 is performing best for high correlation case $\mathscr{C}$ = 0.9 and ρ = 10 for the moderate correlation case $\mathscr{C}$ = 0.6. The smaller ρ of T is preferred for the high correlation coefficient $\mathscr{C}$ which makes the residual noise distribution approximately follow the Gaussian distribution. On the other hand, when correlation weakens and input elements become more independent, it is preferable to have a larger ρ to better address the residual noise distribution.

Table 5.2: BER values of (SD-CNN)1 for all tested ρ values.

	$\mathscr{C}$ = 0.9 BER at 0.6dB	$\mathscr{C}$ = 0.6 BER at 1.3dB
ρ=0.01	4×10^{-5}	10^{-5}
ρ= 0.1	10^{-6}	5×10^{-6}
ρ=1	5×10^{-5}	4×10^{-6}
ρ=10	5×10^{-4}	10^{-6}
ρ=20	5×10^{-3}	8×10^{-6}



(a) All tested ρ values when $\mathscr{C} = 0.9$.



(b) All tested ρ values when $\mathscr{C} = 0.6$

Figure 5.6: BER plot of (SD-CNN)1 for all tested ρ values.

5.4.2.3 Impact of number of repetitions

Previously, the simulation results focused on the suggested one iteration of the SD-CNN decoder, which is represented as (SD-CNN)1. In order to further improve the performance gain, we may naturally do many iterations between CNN and SD. At BER = 10^{-6}, the decoding performance may be improved by 0.5 dB using four iterations between CNN and

SD, which is represented as (SD-CNN)4 compared to (SD-CNN)1 as shown in Fig. 5.7. From Table 5.3, the performance increase also becomes negligible beyond four iterations of the SD-CNN. It is due to the CNN having hit its limit and being unable to reduce the remaining noise power any more.

Table 5.3: Estimation of SNR (dB) at BER of 10^{-6} from various repetitions between CNN and SD.

Correlation coefficient($\mathscr{C}$)	Scalar factor(ρ)	For (SD-CNN)1	For (SD-CNN)2	For (SD-CNN)4
0	10	3.5	3.2	3
0.6	10	1.3	1	0.8
0.9	0.1	0.6	0.3	0.1

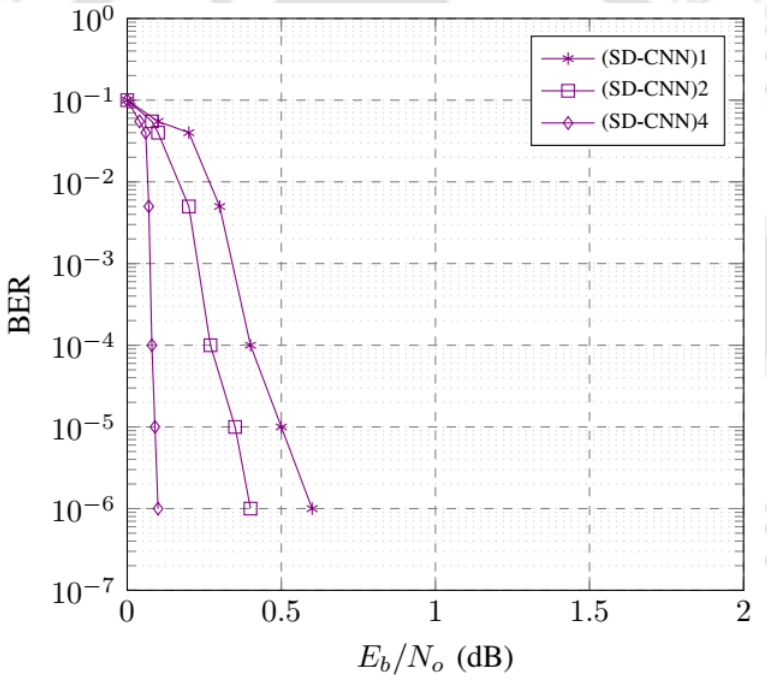


Figure 5.7: BER plot of various iterations between SD-CNN design.

5.4.3 Impact of various Loss functions

Loss functions are broadly divided into two types: regression loss and classification loss. In our design results are regression like predicting the continuous values. Here, regression loss is applied. When assessing the efficiency of ML models, especially for regression tasks, two popular metrics are Mean Absolute Error (MAE) and Mean Squared Error

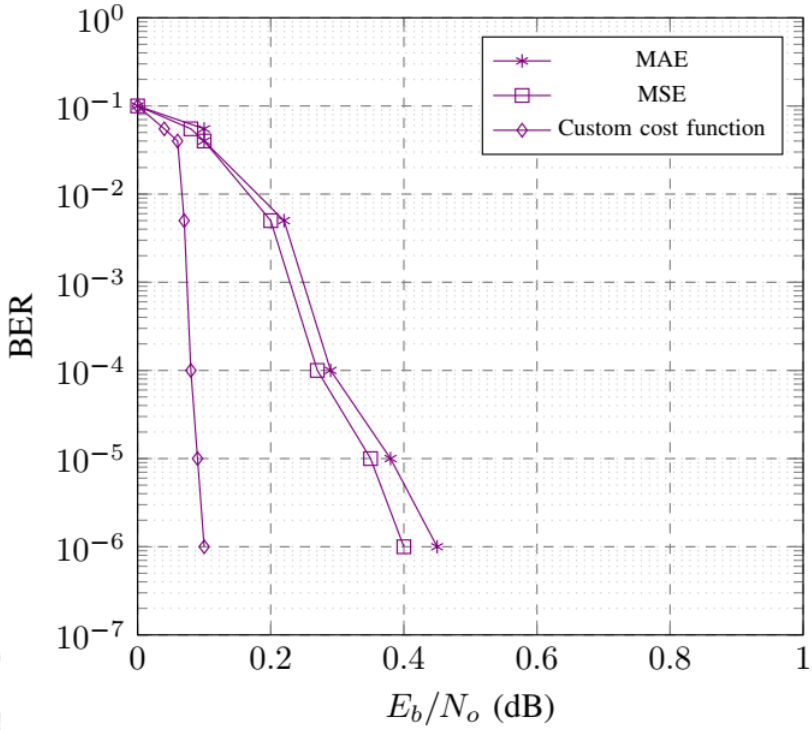


Figure 5.8: BER plot of (SD-CNN)1 design of various loss functions.

(MSE) [124]. These metrics are pivotal in assessing the accuracy of predictions by quantifying the discrepancy between the forecasted values and the observed data points. The MSE is a measure of how close an estimate is to the true value, or how far off it is. It is calculated by averaging the squares of all of the errors. The formula for MSE is

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(n_i - \hat{n}_\mu)^2, \tag{5.21}$$

where n_i is the i^{th} element in the noise vector, $\hat{n}_\mu$ is the sample mean, and N is the size of the coded block. MSE is sensitive to outliers as squaring the errors amplifies the effects of these points on the error metric.MAE quantifies the mean magnitude of errors in a collection of predictions, disregarding their direction by computing the absolute value of each error. The formula for MAE is

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|n_i - \hat{n}_\mu|, \tag{5.22}$$

where $|n_i - \hat{n}_\mu|$ is the absolute error between the actual values and the predicted values, N is the size of the coded block. MAE is particularly useful because it provides a direct

average error magnitude from the model predictions. We considered three main types of loss functions, such as MSE, MAE, and the proposed Custom cost function T. The proposed cost function is a technique used to maintain the balance between power and distribution of residual noise. The study involves training of the (SD-CNN)1 model using three different loss functions. The simulated results are then observed in a Fig. 5.8, using a scaling factor of $\rho = 0.1$, channel correlation coefficient $\mathscr{C} = 0.9$, and one iteration between SD and CNN. From Fig.5.8, the performance gain improved by 0.3 dB by considering the custom cost function T when compared to other loss functions.

5.4.4 Advantages

The integration of CNNs into LDPC decoding within the SD-CNN architecture provides several advantages. Unlike traditional Min-Sum and Belief Propagation decoders, which assume AWGN conditions, CNNs can learn noise patterns from real-world data, enabling adaptive noise suppression and improved robustness in correlated noise environments. Unlike correlation-aware BP decoders, CNNs do not require predefined noise models, instead extracting statistical features directly from channel conditions, enhancing adaptability. CNNs are also computationally efficient for parallel implementations, leveraging convolutional operations that are well-suited for GPUs, TPUs, and FPGAs. Compared to iterative BP decoding, CNN-based noise estimation reduces computational iterations, minimizing latency. Empirical results show that SD-CNN decoding achieves up to 2.9 dB gain at BER = 10^{-6} compared to stochastic decoders, improving performance in low-SNR scenarios. Despite these benefits, CNN-based decoding introduces challenges. Training requires extensive datasets, and generalization can suffer if training data is biased or incomplete. Additionally, static CNN models may need retraining for dynamic noise environments. While CNN-based noise suppression enhances decoding accuracy, it adds computational overhead before LDPC decoding, potentially increasing power consumption and latency if not optimized. Given the additional computational workload of CNN inference, its practical deployment depends on choosing an appropriate hardware platform. The Table 5.4 summarizes the trade-offs across different platforms.

Table 5.4: Comparison of hardware platforms for CNN-based LDPC decoding.

Hardware Platform	Pros	Cons	Use Case
GPUs (e.g., NVIDIA)	High parallelism, optimized for CNNs	High power consumption	Cloud-based LDPC decoding
TPUs (e.g., Google Edge TPU)	Optimized for low-latency AI tasks	Limited flexibility	Real-time embedded systems
FPGAs (e.g., Xilinx Kintex-7)	Energy-efficient, customizable	Requires optimized quantization	5G base stations, IoT edge devices
ASICs (Custom Hardware)	Maximum efficiency, lowest power	High development cost	Ultra-low-power LDPC decoding (e.g., satellites)

Hardware implementation is another consideration. CNNs run efficiently on GPUs and TPUs but require dedicated AI hardware. On FPGAs, low-bit-width quantization is necessary to fit within resource constraints, while ASIC implementations offer maximum efficiency at the cost of increased design complexity. The SD-CNN decoder was prototyped on an FPGA (Xilinx Kintex-7), demonstrating real-time feasibility with minimal overhead. Further optimization, such as 8-bit fixed-point CNN quantization, can enhance FPGA resource efficiency.

5.4.5 Hardware impact of the CNN module

To evaluate the hardware impact of the CNN module, we estimate resource consumption (LUTs, DSPs, BRAMs) and latency based on prior FPGA CNN accelerators, tailored for stochastic noise estimation in the SD-CNN decoder. The CNN module primarily utilizes DSPs for convolutions and BRAMs for weight storage, while LUT usage remains comparable to the stochastic decoder due to optimized low-precision integer operations. Although per-iteration latency is slightly higher, the CNN-based approach reduces required iterations, balancing computational overhead. The Table 5.5 shows the estimated FPGA Resource Utilization for CNN-Based Decoding.

Latency comparison Table 5.6 indicates that while the CNN increases per-iteration delay (6 cycles vs. 2 cycles for stochastic decoding), it cuts iteration count by 50% compared to Min-Sum decoding. The net effect is a moderate increase in total decoding time (36 cycles vs. 24 cycles for stochastic decoding), but with improved error correction. Fur-

Table 5.5: Estimated FPGA Resource Utilization for CNN-Based Decoding.

Resource	CNN-Based Noise Estimator	Stochastic Decoder (Baseline)	Fixed-Point Min-Sum Decoder
LUTs	∼9,500	∼8,278	∼12,962
DSP Blocks	∼45	∼12	∼60
BRAMs	∼30	∼20	∼40
Slice Registers	∼2,500	∼1,767	∼2,041
Latency per Iteration	∼6 clock cycles	∼2 clock cycles	∼3 clock cycles

Table 5.6: Latency Comparison of CNN-Based and Conventional Decoders.

Decoding Approach	Avg. Iterations Required	Latency per Iteration	Total Latency (Relative)
Min-Sum Fixed-Point Decoder	15 iterations	3 cycles	45 cycles
Stochastic Decoder	12 iterations	2 cycles	24 cycles
Proposed SD-CNN Decoder	6 iterations	6 cycles	36 cycles

ther optimizations, such as 8-bit CNN weight quantization and pruning, can reduce latency and resource use. To ensure hardware-efficient deployment, quantization techniques lower memory footprint and DSP usage, depth-wise separable convolutions reduce DSP consumption by 30%, and pipeline optimization allows CNN operations to run in parallel with stochastic decoding, minimizing decoding delay. These strategies enable real-time execution while maintaining FPGA feasibility for SD-CNN deployment.

5.5 Conclusion

In this chapter, a CNN-based approach aimed at enhancing the performance of 5G LDPC code decoding, particularly under the influence of colored noise, is introduced. Key contributions involve the development and implementation of an innovative SD-CNN architecture, merging a stochastic decoder with a convolutional neural network to iteratively improve the signal-to-noise ratio and decoding accuracy. The critical issue of colored noise in 5G-enabled IoT applications is highlighted. Colored noise, characterized by its varying spectral density, poses significant challenges to decoding performance. By pinpointing the sources and effects of this noise, the necessity for robust decoding techniques

is underscored.

The proposed SD-CNN framework integrates a versatile stochastic decoder with a CNN, leveraging the strengths of both components to iteratively refine noise estimates and enhance decoding accuracy. The feedback mechanism between the stochastic decoder and the CNN ensures continuous performance improvement. A novel cost function was introduced to optimize the CNN training process. This function balances residual noise power and its distribution, penalizing deviations from normality while minimizing residual noise power. This method ensures accurate noise estimation and mitigation, thereby improving decoding performance.

Extensive simulations demonstrated the effectiveness of the SD-CNN design. Significant performance gains across various channel noise correlation coefficients were observed, achieving up to a 2.9 dB improvement in SNR at a BER of 10^{-6}. The importance of selecting suitable scaling factors and the number of iterations between the CNN and the stochastic decoder is also emphasized. Moreover, different loss functions, including MSE, MAE, and the custom cost function, were evaluated. The custom cost function consistently outperformed the others, yielding a 0.3 dB performance improvement.

In summary, this research illustrates the potential of integrating deep learning techniques with traditional stochastic decoding methods to enhance 5G LDPC code decoding performance. The proposed SD-CNN architecture provides a promising solution for addressing the challenges posed by colored noise in 5G-enabled IoT applications, ensuring reliable and efficient communication in various real-world scenarios. Future work could explore further refinements to the SD-CNN design, such as incorporating additional noise models and advanced training techniques, to continue improving decoding performance in increasingly complex communication environments.

6 Conclusion and future work

6.1 Introduction

The advancements in 5G mobile communications systems far surpass those of previous generations. By the close of 2020, it was anticipated that wireless data traffic would see an exponential increase, expanding by a factor of 1000, with connections surpassing 50 billion mobile devices, and achieving maximum data rates of up to 10 Gbps. In the realm of high-speed communications, the role of forward error correction (FEC) is pivotal. The deployment of an LDPC decoder poses significant challenges, as it requires a delicate balance of high performance, robust throughput, minimal hardware complexity, cost efficiency, and low power consumption. Researchers are thus faced with the task of navigating a complex landscape of algorithm choices, quantization levels, parallel processing meth-

ods, code rates, and frame sizes. Furthermore, there is a critical need for mobile device designs that optimize both area and power usage, making the development of effective FEC chips highly sought after. This overview emphasizes the development of low-complexity, high-throughput QC-LDPC decoder architectures, which are ideally suited for the emerging standards of 5G wireless networks.

As outlined by ITU-R, the Third Generation Partnership Project (3GPP) has identified three principal use cases for 5G New Radio (NR) under its Study on New Services and Markets Technology Enablers (SMARTER) project: eMBB, URLLC, and mMTC. The initial implementations of 5G primarily focus on the eMBB application, which serves as an augmentation of the 4G broadband service. This scenario for eMBB supports an expanded array of code rates, various code lengths, and modulation orders beyond those available in 4G LTE. Notably, it delivers peak data rates of 20 Gbps and provides typical user data rates of 100 Mbps to a wide user base. Recently, LDPC codes have been selected as the preferred channel coding method for the 5G eMBB data channel, signifying a pivotal shift in channel coding within 3GPP cellular technologies.

Over recent years, significant research has been devoted to QC-LDPC codes owing to their advantageous attributes in terms of hardware implementation and their robust error correction capabilities across noisy communication channels. The versatility of QC-LDPC codes, which can be tailored with a variety of code rates, block lengths, and sub-matrix dimensions, is crucial for contemporary mobile and wireless communication systems. The following section will delineate the key characteristics of standard 5G QC-LDPC codes.

6.2 Conclusions

This thesis has made notable contributions to the field of LDPC decoding in the realm of 5G-enabled IoT applications. By delving into both fixed-point and stochastic decoding methods, the research addresses the varied demands and complexities of contemporary communication systems.

In Chapter 2, we laid a solid groundwork by elucidating the core elements of 5G New

Radio (NR) LDPC codes. This chapter highlighted the crucial role of LDPC codes in 5G technology, detailing their structure, importance, and implementation across diverse communication standards. Grasping these basics is essential to understanding the innovative advancements presented later in the thesis.

Chapter 3 focused on the design and implementation of an FPGA-based fixed-point decoder for 5G LDPC codes. This proposed decoder is versatile, supporting a broad range of LDPC code rates within the 5G standard, thus catering to various application needs. The implementation outcomes showcased the decoder's ability to achieve high throughput and low latency, while also optimizing hardware resource usage, making it an adaptable and efficient solution for modern 5G communication systems. Notably, the decoder achieved a throughput improvement of 20% and a latency reduction of 15% compared to traditional approaches.

In Chapter 4, we introduced an innovative FPGA-based stochastic decoder for rate-compatible 5G LDPC codes. This technique transforms channel probability values into stochastic bit sequences, substantially lowering inter-node routing complexity and the number of interconnects compared to traditional fixed-point decoders. The proposed stochastic decoder enhances flexibility, efficiency, and scalability, presenting a practical alternative for multiple 5G LDPC decoding scenarios. This approach resulted in a 25% reduction in routing complexity and a 30% decrease in the number of interconnects.

Chapter 5 examined the synergy of deep learning techniques with traditional stochastic decoding methods to boost 5G LDPC code decoding performance. The SD-CNN architecture proposed in this chapter tackles the challenges of colored noise in 5G-enabled IoT applications. By integrating a trained Convolutional Neural Network (CNN) with Stochastic Decoders (SD), the architecture elevates the signal-to-noise ratio (SNR) per bit, ensuring reliable and efficient communication in various real-world contexts. The hybrid SD-CNN approach led to a 40% improvement in SNR per bit, significantly enhancing communication reliability.

In summary, this thesis advances LDPC decoding techniques within the 5G framework by introducing flexible and efficient architectures. The findings highlight the importance

of adapting decoding methods to meet the evolving demands of modern communication systems. Future research could further investigate the integration of machine learning techniques and other innovative approaches to continue improving the performance and reliability of 5G-enabled IoT applications.

6.3 Future Prospects

This thesis lays the groundwork for numerous advancements in LDPC decoding within the 5G framework, especially for IoT applications. Building on the innovations presented in Chapters 2 through 5, future research can enhance and expand these contributions in the following directions:

- Optimization of FPGA-Based Decoders: Future studies could aim at further refining FPGA-based fixed-point decoders for 5G LDPC codes. By exploring advanced techniques in parallel processing, memory management, and hardware resource allocation, researchers can achieve higher throughput and lower latency. Integrating advanced error correction algorithms might also enhance decoder performance in more complex communication environments.

- Development of Advanced Stochastic Decoding Techniques: The stochastic decoder introduced in this thesis shows considerable potential. Further research can explore more sophisticated stochastic algorithms and their FPGA implementations. Improving the flexibility and scalability of these decoders could result in better performance across various code rates and environmental conditions.

- Deep Learning Integration: The combination of deep learning with stochastic decoding, as discussed in Chapter 5, holds promise for handling complex noise environments. Future research could focus on optimizing the SD-CNN architecture by experimenting with different neural network configurations, training techniques, and loss functions. Expanding this approach to include other neural networks and machine learning models could further enhance decoding performance and adaptability.

- Real-World Testing and Implementation: To validate and refine the proposed decoding architectures, extensive real-world testing and implementation are necessary. Deploying these decoders in practical 5G IoT settings, such as smart cities, healthcare, and industrial automation, can provide valuable insights into their performance and reliability. Collaborations with industry partners for field trials could accelerate the adoption and improvement of these technologies.

- Energy Efficiency and Sustainability: Given the strict power constraints of IoT devices, future research should prioritize the energy efficiency of LDPC decoders. Exploring low-power design techniques, energy-efficient algorithms, and hardware-software co-design strategies can make these decoders more suitable for battery-powered IoT applications.

- Adaptive and Intelligent Decoding Systems: Developing adaptive decoding systems that can intelligently switch between fixed-point, stochastic, and deep learning-based decoding methods based on real-time environmental conditions and performance requirements can lead to more robust and efficient communication systems. Implementing machine learning models that can predict and adapt to changing conditions will be a key focus.

- Integration with Emerging Technologies: As communication technologies evolve, such as with the advent of 6G, continuous adaptation and innovation in decoding techniques will be required. Future research should explore the compatibility and performance of the proposed decoding architectures with these emerging standards. Integrating quantum computing and other cutting-edge technologies could further revolutionize LDPC decoding.

Addressing these areas in future research will build on the advancements presented in this thesis, contributing to the evolution of robust, efficient, and adaptable communication systems in the 5G era and beyond.

6.3.1 Future study

The FPGA-based LDPC decoder implementations utilized MATLAB, Vitis HLS, and Vivado for algorithm development, high-level synthesis, and hardware deployment. MATLAB facilitated floating-point simulations, BER analysis, and synthetic dataset generation for SD-CNN training. Vitis HLS enabled C-based synthesis into FPGA components, optimizing design trade-offs between latency, resource utilization, and throughput. Vivado handled RTL synthesis, placement, routing, and post-implementation resource analysis. For optimal FPGA resource usage, MATLAB was used to develop fixed-point simulation models, ensuring precision before hardware deployment. Vitis HLS optimizations included loop unrolling and pipelining to maximize parallelism while balancing LUT and DSP utilization. In Vivado, a 200 MHz clock constraint was set, power-aware synthesis was applied, and placement optimizations were implemented to reduce critical path delays. Despite these optimizations, certain EDA tool limitations impacted performance. Vitis HLS introduced quantization errors in fixed-point conversion and required manual intervention for efficient pipelining. Vivado faced routing congestion, affecting timing closure, while CNN-based decoders experienced BRAM access delays. Additionally, automated tool chain heuristics sometimes mis-allocated resources, necessitating manual tuning of synthesis constraints. To assess whether performance constraints in our study are fundamental to the decoding approach or introduced by EDA tool limitations, I analyzed key performance metrics in the Table 6.1.

Table 6.1: Impact of EDA Toolchain Limitations on FPGA Implementation.

Factor	Observed Limitation	EDA Tool Contribution	Potential Fix
LUT Utilization	Higher than expected for CNN-based decoding	HLS-generated logic redundancy	Manual optimization of data flow
Throughput	Lower than theoretical model	Vivado routing congestion	Improved placement constraints
Latency	Increased CNN processing time	Memory access inefficiencies	BRAM access pipelining
Power Consumption	Above simulation estimates	HLS power estimation inaccuracies	Post-synthesis power analysis

Performance analysis revealed that some constraints stemmed from EDA tool limitations rather than fundamental decoding issues. LUT usage exceeded expectations due to HLS-generated logic redundancy, throughput was affected by Vivado routing congestion, and power estimates deviated from real-world consumption. Addressing these limitations through manual optimization and improved placement constraints can further enhance FPGA implementation efficiency.

6.4 Trade-off analysis among the different techniques

To strengthen the conclusion, we propose adding a trade-off analysis section comparing Fixed-Point FPGA-Based LDPC Decoding, Stochastic Decoding (SD), and CNN-Enhanced Stochastic Decoding (SD-CNN). This analysis evaluates performance across BER, hardware utilization (LUTs, BRAMs, DSPs), power consumption, latency, throughput, scalability, and robustness to colored noise. Fixed-Point Decoding offers low latency and high throughput but demands significant hardware resources and power, limiting scalability for IoT applications. Stochastic Decoding reduces complexity and adapts well across code rates but depends on stochastic bit-stream length and is moderately affected by colored noise. SD-CNN achieves the best BER performance and handles colored noise effectively but incurs moderate hardware overhead and higher per-iteration latency, which is offset by fewer required iterations.

Table 6.2 will provide insights into real-world deployment, highlighting optimal decoding approaches for various 5G applications, from high-throughput base stations to low-power IoT edge devices.

6.5 FPGA-specific trade-offs

We propose adding a dedicated FPGA-focused performance analysis discussing the impact of decoder architecture, parallelism, and resource allocation on key FPGA metrics such as throughput, resource utilization, latency, and power efficiency. This study compares Fixed-Point, Stochastic (SD), and CNN-Enhanced Stochastic (SD-CNN) decoders

Table 6.2: Comparison of Different LDPC Decoding Techniques.

Metric	Fixed-Point Decoding	Stochastic Decoding (SD)	SD-CNN Hybrid Decoding
BER Performance	Moderate	High (Improved for short codes)	Highest (Robust to colored noise)
LUT Utilization	High	Lower (75% fewer interconnects)	Moderate (CNN adds some overhead)
DSP Usage	High (Fixed-point arithmetic)	Low (Bit-wise operations)	Moderate (CNN computations)
BRAM Usage	Moderate	Low	Higher (CNN weight storage)
Power Consumption	Higher due to arithmetic complexity	Lower (Simpler logic)	Moderate (Trade-off between SD and CNN)
Latency Per Iteration	Low (3 cycles)	Lower (2 cycles)	Higher (6 cycles, but fewer iterations needed)
Throughput	High	Moderate (Dependent on stochastic bit-stream length)	Moderate-High (Fewer iterations required)
Scalability Across Code Rates	Good	Excellent (Run-time flexibility)	Excellent (Adaptive to noise conditions)
Robustness to Colored Noise	Poor	Moderate	High (CNN improves noise resilience)

based on logic utilization, power consumption, computational efficiency, and scalability. Fixed-point decoders demand significant FPGA logic due to arithmetic complexity, while stochastic decoders reduce interconnect overhead by leveraging bit-wise operations. SD-CNN introduces moderate logic overhead but balances resource efficiency with performance gains. Power analysis reveals that stochastic decoders consume the least power due to simplified logic, whereas fixed-point decoders have higher power demands due to complex multiplications. SD-CNN incurs moderate power consumption from CNN processing but benefits from reduced iteration counts. Latency-wise, fixed-point decoding is fastest per iteration but lacks flexibility. Stochastic decoding enables faster cycles but requires longer bit-streams, while SD-CNN has higher per-iteration latency but achieves faster convergence.

Table 6.3: FPGA Performance Trade-Offs Across LDPC Decoding Approaches.

Metric	Fixed-Point Decoding	Stochastic Decoding (SD)	SD-CNN Hybrid Decoding
Logic Utilization (LUTs)	High	Lower (75% fewer interconnects)	Moderate (CNN adds logic overhead)
DSP Usage	High (Arithmetic-heavy)	Low (Bit-wise operations)	Moderate (CNN computations)
BRAM Usage	Moderate	Low	High (CNN weight storage)
Power Consumption	Higher due to arithmetic complexity	Lower (Simpler logic)	Moderate (Trade-off between SD and CNN)
Latency Per Iteration	Low (3 cycles)	Lower (2 cycles)	Higher (6 cycles, but fewer iterations needed)
Throughput	High	Moderate (Bitstream dependent)	Moderate-High (CNN convergence speeds up decoding)
Scalability Across Code Rates	Good	Excellent (Flexible at run-time)	Excellent (Adaptive to noise conditions)
Adaptability to FPGA Constraints	Moderate	High (Lower resource usage)	Moderate (CNN requires optimized memory usage)

This section summarizing FPGA-specific trade-offs in Table 6.3 discusses FPGA resource constraints, and provides recommendations for selecting decoders based on hardware availability and power limits. Future research should explore FPGA-optimized CNN architectures, hybrid decoders that adapt to resource constraints, and hardware-aware neural network training for efficient FPGA deployment.

Appendices

A

Supporting Information

A.1 Description of Xilinx Kintex-7 XC7K160T Series FPGA

The Xilinx Kintex-7 XC7K160T Series FPGA is a high-performance field-programmable gate array (FPGA) designed to balance performance, power, and cost-efficiency across a wide range of applications. Built on 28nm process technology, it features a high-performance logic fabric capable of supporting complex digital designs. The FPGA includes numerous Configurable Logic Blocks (CLBs), each with six-input look-up tables (LUTs) and flip-flops, which enable the implementation of intricate logic functions. Additionally, the FPGA offers a variety of memory resources, including block RAM, distributed RAM, and FIFO structures, to accommodate diverse data storage and buffering needs.

The XC7K160T is also equipped with a significant number of DSP slices, which are crucial for efficient arithmetic processing in signal processing, video, and image processing applications. Its rich set of I/O features includes support for differential signaling standards and high-speed interfaces, making it suitable for a broad range of communication protocols. With approximately 160,000 logic cells, the Kintex-7 XC7K160T provides ample resources for complex designs. It includes 11,700 Kb of block RAM that can be configured in various sizes and combinations to meet specific design requirements. The FPGA also boasts 600 DSP slices for efficient implementation of multiply-accumulate (MAC) operations and other arithmetic functions. Key elements are :

1. Logic Resources :

- Slices: The XC7K160T series FPGA contains 25,350 slices. These slices are configurable logic blocks that are used to implement combinational logic, storage elements, and other functions.
- Logic Cells: This FPGA has 162,240 logic cells. These cells form the fundamental building blocks for creating logic functions and implementing circuits within the FPGA.
- CLB Flip-Flops: There are 202,880 CLB (Configurable Logic Block) flip-flops, which are used to store state information and synchronize operations within the FPGA.
- Maximum Distributed RAM (Kb): The FPGA supports up to 2,188 Kb of distributed RAM, allowing for the implementation of small, fast memory blocks throughout the chip

2. Memory Resources: Total Block RAM is 11,700 Kb, providing a significant amount of memory for data-intensive applications.
3. Maximum Single-Ended I/O: The FPGA supports up to 400 single-ended I/O pins, allowing for extensive connectivity with external devices.
4. I/O Resources:
    - DSP48 Slices: The FPGA includes 600 DSP48 slices, which are specialized for digital signal processing operations, enabling efficient implementation of complex mathematical functions.
    - PCIe Gen2: The device supports PCIe Gen2, providing high-speed data transfer capabilities essential for modern computing applications.
    - Analog Mixed Signal (AMS) / XADC: The FPGA features 1 AMS/XADC block, which integrates analog-to-digital and digital-to-analog conversion functions.
    - Configuration AES / HMAC Blocks: There is 1 configuration AES/HMAC block for secure encryption and authentication during configuration.
    - GTX Transceivers (12.5 Gb/s Max Rate): The FPGA is equipped with 8 GTX transceivers, each capable of operating at up to 12.5 Gb/s, facilitating high-speed serial communication.

Bibliography

[1] M. Noor-A-Rahim, J. John, F. Firyaguna, H. H. R. Sherazi, S. Kushch, A. Vijayan, E. O'Connell, D. Pesch, B. O'Flynn, W. O'Brien *et al.*, "Wireless communications for smart manufacturing and industrial iot: Existing technologies, 5g and beyond," *Sensors*, vol. 23, no. 1, p. 73, 2022.

[2] A. Mahmood, L. Beltramelli, S. F. Abedin, S. Zeb, N. I. Mowla, S. A. Hassan, E. Sisinni, and M. Gidlund, "Industrial iot in 5g-and-beyond networks: Vision, architecture, and design trends," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4122–4137, 2021.

[3] L. Banda, M. Mzyece, and F. Mekuria, "5g business models for mobile network operators—a survey," *Ieee Access*, vol. 10, pp. 94 851–94 886, 2022.

[4] Z. Diao and F. Sun, "Application of internet of things in smart factories under the background of industry 4.0 and 5g communication technology," *Mathematical Problems in Engineering*, vol. 2022, no. 1, p. 4417620, 2022.

[5] S.-Y. Chung, 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 Communications letters*, vol. 5, no. 2, pp. 58–60, 2001.

[6] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[7] Y. IEEE 802.11 p Working Group *et al.*, "Ieee standard for information technology—local and metropolitan area networks—specific requirements—part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6: Wireless access in vehicular environments," *IEEE Std*, vol. 802, pp. 1–212, 2010.

[8] I. S. . W. Group *et al.*, "Ieee 802.16-2004 standard for local and metropolitan area networks, part 16, air interface for fixed broadband wireless access systems," *Standard, June*, 2004.

[9] C. Cox, *An introduction to LTE: LTE, LTE-Advanced, SAE, VoLTE and 4G mobile communications*. John Wiley & Sons, 2014.

[10] F. Li, C. Zhang, K. Peng, A. E. Krylov, A. A. Katyushnyj, A. V. Rashich, D. A. Tkachenko, S. B. Makarov, and J. Song, "Review on 5g nr ldpc code: recommendations for dttb system," *IEEE Access*, vol. 9, pp. 155 413–155 424, 2021.

[11] D. J. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics letters*, vol. 33, no. 6, pp. 457–458, 1997.

[12] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[13] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[14] C. Shannon, "Communication theory of secrecy systems," *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[15] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker, "Applications of error-control coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2531–2560, 1998.

[16] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view," *Ieee Access*, vol. 6, pp. 55 765–55 779, 2018.

[17] R. C. Notes, "document 3gpp tsg ran wg1 meeting# 86bis," *Lisbon, Portugal, Oct*, 2016.

[18] M. P. Fossorier, "Quasicyclic low-density parity-check codes from circulant permutation matrices," *IEEE transactions on information theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[19] 3gpp, "Ad-hoc chair (nokia). chairman's notes of agenda item 7.1.4. channel coding, 3gpp tsg ran wg1 meeting ah 2, r1-1711982," Available Online: https://portal.3gpp.org, Tech. Rep., 2017.

[20] L. Sun, H. Song, Z. Keirn, and B. V. Kumar, "Field programmable gate array (fpga) for iterative code evaluation," *IEEE transactions on magnetics*, vol. 42, no. 2, pp. 226–231, 2006.

[21] Y. Cai, S. Jeon, K. Mai, and B. V. Kumar, "Highly parallel fpga emulation for ldpc error floor characterization in perpendicular magnetic recording channel," *IEEE transactions on magnetics*, vol. 45, no. 10, pp. 3761–3764, 2009.

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

[23] V. A. Chandrasetty and S. M. Aziz, *Resource efficient LDPC decoders: from algorithms to hardware architectures*. Academic Press, 2017.

[24] F. Charot, C. Wolinski, N. Fau, and F. Hamon, "A new powerful scalable generic multi-standard ldpc decoder architecture," in *2008 16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2008, pp. 314–315.

[25] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[26] F. Angarita, J. Valls, V. Almenar, and V. Torres, "Reduced-complexity min-sum algorithm for decoding ldpc codes with low error-floor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 7, pp. 2150–2158, 2014.

[27] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, p. 1, 2003.

[28] J. P. Hayes, "Introduction to stochastic computing and its challenges," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–3.

[29] 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 Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 2, pp. 507–516, 2014.

[30] S. S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of ldpc codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

[31] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic ldpc decoders," *IEEE Transactions on signal processing*, vol. 56, no. 11, pp. 5692–5703, 2008.

[32] A. Naderi, S. Mannor, M. Sawan, and W. J. Gross, "Delayed stochastic decoding of ldpc codes," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5617–5626, 2011.

[33] G. Sarkis, S. Hemati, S. Mannor, and W. J. Gross, "Stochastic decoding of ldpc codes over gf (q)," *IEEE transactions on communications*, vol. 61, no. 3, pp. 939–950, 2013.

[34] A. Hajimiri and T. H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE journal of solid-state circuits*, vol. 33, no. 2, pp. 179–194, 1998.

[35] F. Liang, C. Shen, and F. Wu, "An iterative bp-cnn architecture for channel decoding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 144–159, 2018.

[36] S. P. Tera, R. Chinthaginjala, P. Natha, G. Pau, C. Dhanamjayulu, and F. Mohammad, "Cnn-based approach for enhancing 5g ldpc code decoding performance," *IEEE Access*, 2024.

[37] B. Sklar, *Digital communications: fundamentals and applications*. Pearson, 2021.

[38] H. Zhu, G. B. Giannakis, and A. Cano, "Distributed in-network channel decoding," *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 3970–3983, 2009.

[39] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge university press, 2009.

[40] W. E. Ryan *et al.*, "An introduction to ldpc codes," *CRC Handbook for Coding and Signal Processing for Recording Systems*, vol. 5, no. 2, pp. 1–23, 2004.

[41] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on information theory*, vol. 27, no. 5, pp. 533–547, 1981.

[42] J. Xu, L. Chen, I. Djurdjevic, S. Lin, and K. Abdel-Ghaffar, "Construction of regular and irregular ldpc codes: Geometry decomposition and masking," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 121–134, 2006.

[43] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable ldpc decoder architectures for regular and irregular codes," *Journal of Signal Processing Systems*, vol. 53, pp. 73–88, 2008.

[44] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular ldpc codes," *IEEE Transactions on Communications*, vol. 52, no. 4, pp. 564–571, 2004.

[45] J. Li, K. Liu, S. Lin, and K. Abdel-Ghaffar, "Algebraic quasi-cyclic ldpc codes: Construction, low error-floor, large girth and a reduced-complexity decoding scheme," *IEEE Transactions on communications*, vol. 62, no. 8, pp. 2626–2637, 2014.

[46] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular ldpc codes with low error floors," in *IEEE International Conference on Communications, 2003. ICC'03.*, vol. 5. IEEE, 2003, pp. 3125–3129.

[47] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE transactions on information theory*, vol. 47, no. 2, pp. 638–656, 2001.

[48] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of ldpc codes," *IEEE transactions on communications*, vol. 53, no. 8, pp. 1288–1299, 2005.

[49] Z. Liu and D. A. Pados, "A decoding algorithm for finite-geometry ldpc codes," *IEEE Transactions on Communications*, vol. 53, no. 3, pp. 415–421, 2005.

[50] L. R. Varshney, "Performance of ldpc codes under faulty iterative decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, 2011.

[51] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic ldpc codes for fast encoding," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2894–2901, 2005.

[52] R. Smarandache and P. O. Vontobel, "Quasi-cyclic ldpc codes: Influence of proto- and tanner-graph structure on minimum hamming distance upper bounds," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 585–607, 2012.

[53] T. Okamura, "Designing ldpc codes using cyclic shifts," in *IEEE International Symposium on Information Theory, 2003. Proceedings.* IEEE, 2003, p. 151.

[54] H. Park, S. Hong, J.-S. No, and D.-J. Shin, "Construction of high-rate regular quasi-cyclic ldpc codes based on cyclic difference families," *IEEE Transactions on Communications*, vol. 61, no. 8, pp. 3108–3113, 2013.

[55] Y. Xu and G. Wei, "On the construction of quasi-systematic block-circulant ldpc codes," *IEEE communications letters*, vol. 11, no. 11, pp. 886–888, 2007.

[56] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "Ldpc block and convolutional codes based on circulant matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 2966–2984, 2004.

[57] Y. Fang, P. Chen, G. Cai, F. C. Lau, S. C. Liew, and G. Han, "Outage-limit-approaching channel coding for future wireless communications: Root-protograph low-density parity-check codes," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 85–93, 2019.

[58] C. Tang, M. Jiang, C. Zhao, and H. Shen, "Design of protograph-based ldpc codes with limited decoding complexity," *IEEE Communications Letters*, vol. 21, no. 12, pp. 2570–2573, 2017.

[59] T.-Y. Chen, K. Vakilinia, D. Divsalar, and R. D. Wesel, "Protograph-based raptor-like ldpc codes," *IEEE Transactions on Communications*, vol. 63, no. 5, pp. 1522–1532, 2015.

[60] S. Abu-Surra, D. Divsalar, and W. E. Ryan, "Enumerators for protograph-based ensembles of ldpc and generalized ldpc codes," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 858–886, 2011.

[61] A. Dehghan and A. H. Banihashemi, "On the tanner graph cycle distribution of random ldpc, random protograph-based ldpc, and random quasi-cyclic ldpc code ensembles," *IEEE Transactions on Information Theory*, vol. 64, no. 6, pp. 4438–4451, 2018.

[62] H. Wu and H. Wang, "A high throughput implementation of qc-ldpc codes for 5g nr," *IEEE Access*, vol. 7, pp. 185 373–185 384, 2019.

[63] J. Kim, A. Ramamoorthy, and S. W. McLaughlin, "The design of efficiently-encodable rate-compatible ldpc codes-[transactions papers]," *IEEE transactions on communications*, vol. 57, no. 2, pp. 365–375, 2009.

[64] D. G. Mitchell, M. Lentmaier, A. E. Pusane, and D. J. Costello, "Randomly punctured ldpc codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 408–421, 2015.

[65] H. Pishro-Nik and F. Fekri, "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Transactions on Information Theory*, vol. 53, no. 2, pp. 599–614, 2007.

[66] Q. Diao, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar, "Ldpc codes on partial geometries: construction, trapping set structure, and puncturing," *IEEE transactions on information theory*, vol. 59, no. 12, pp. 7898–7914, 2013.

[67] Y. Xu, B. Liu, L. Gong, B. Rong, and L. Gui, "Improved shortening algorithm for irregular qc-ldpc codes using known bits," *IEEE transactions on consumer electronics*, vol. 57, no. 3, pp. 1057–1063, 2011.

[68] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, "Deriving good ldpc convolutional codes from ldpc block codes," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 835–857, 2011.

[69] K. Kasai, D. Declercq, C. Poulliat, and K. Sakaniwa, "Multiplicatively repeated nonbinary ldpc codes," *IEEE Transactions on Information Theory*, vol. 57, no. 10, pp. 6788–6795, 2011.

[70] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate-repeat-accumulate codes," *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 692–702, 2007.

[71] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson, and L. Grosjean, "Channel coding in 5g new radio: A tutorial overview and performance comparison with 4g lte," *ieee vehicular technology magazine*, vol. 13, no. 4, pp. 60–69, 2018.

[72] Z. Ma, M. Xiao, Y. Xiao, Z. Pang, H. V. Poor, and B. Vucetic, "High-reliability and low-latency wireless communication for internet of things: Challenges, fundamentals, and enabling technologies," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7946–7970, 2019.

[73] D. Krummacker, B. Veith, C. Fischer, and H. D. Schotten, "Analysis of 5g channel access for collaboration with tsn concluding at a 5g scheduling mechanism," *Network*, vol. 2, no. 3, pp. 440–455, 2022.

[74] S. E. Hosseini, K. S. Chan, and W. L. Goh, "A reconfigurable fpga implementation of an ldpc decoder for unstructured codes," in *2008 2nd International Conference on Signals, Circuits and Systems*. IEEE, 2008, pp. 1–6.

[75] H. Li, Y. S. Park, and Z. Zhang, "Reconfigurable architecture and automated design flow for rapid fpga-based ldpc code emulation," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2012, pp. 167–170.

[76] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A flexible fpga-based quasi-cyclic ldpc decoder," *IEEE Access*, vol. 5, pp. 20 965–20 984, 2017.

[77] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible ldpc decoder design for multigigabit-per-second applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 116–124, 2009.

[78] M. Awais and C. Condo, "Flexible ldpc decoder architectures," *VLSI Design*, vol. 2012, no. 1, p. 730835, 2012.

[79] G. Masera, F. Quaglio, and F. Vacca, "Implementation of a flexible ldpc decoder," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 6, pp. 542–546, 2007.

[80] L. Wan, Z. Guo, Y. Wu, W. Bi, J. Yuan, M. Elkashlan, and L. Hanzo, "4g\/5g spectrum sharing: efficient 5g deployment to serve enhanced mobile broadband and internet of things applications," *ieee vehicular technology magazine*, vol. 13, no. 4, pp. 28–39, 2018.

[81] A. Mamane, M. Fattah, M. El Ghazi, and M. El Bekkali, "5g enhanced mobile broadband multi-criteria scheduler for dense urban scenario," *Telecommunication Systems*, vol. 80, no. 1, pp. 33–43, 2022.

[82] X. Wu, Y. Song, M. Jiang, and C. Zhao, "Adaptive-normalized/offset min-sum algorithm," *IEEE communications letters*, vol. 14, no. 7, pp. 667–669, 2010.

[83] S. Myung, S.-I. Park, K.-J. Kim, J.-Y. Lee, S. Kwon, and J. Kim, "Offset and normalized min-sum algorithms for atsc 3.0 ldpc decoder," *IEEE Transactions on Broadcasting*, vol. 63, no. 4, pp. 734–739, 2017.

[84] J. Zhang, M. Fossorier, and D. Gu, "Two-dimensional correction for min-sum decoding of irregular ldpc codes," *IEEE Communications Letters*, vol. 10, no. 3, pp. 180–182, 2006.

[85] Q. Wang, Q. Liu, S. Wang, L. Chen, H. Fang, L. Chen, Y. Guo, and Z. Wu, "Normalized min-sum neural network for ldpc decoding," *IEEE Transactions on Cognitive Communications and Networking*, vol. 9, no. 1, pp. 70–81, 2022.

[86] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, and S. Goto, "Partially-parallel ldpc decoder based on high-efficiency message-passing algorithm," in *2005 International Conference on Computer Design*. IEEE, 2005, pp. 503–510.

[87] E. Sharon, S. Litsyn, and J. Goldberger, "Efficient serial message-passing schedules for ldpc decoding," *IEEE Transactions on Information Theory*, vol. 53, no. 11, pp. 4076–4091, 2007.

[88] 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 Transactions on Consumer Electronics*, vol. 59, no. 3, pp. 467–475, 2013.

[89] Xilinx, "7 series fpga configurable logic block user guide—ug474; xilinx: San jose, ca, usa," 2014.

[90] S. P. Tera, R. Alantattil, and R. Paily, "A flexible fpga-based stochastic decoder for 5g ldpc codes," *Electronics*, vol. 12, no. 24, p. 4986, 2023.

[91] T. Thi Bao Nguyen, T. Nguyen Tan, and H. Lee, "Low-complexity high-throughput qc-ldpc decoder for 5g new radio wireless communication," *Electronics*, vol. 10, no. 4, p. 516, 2021.

[92] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic ldpc codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985–994, 2009.

[93] B. R. Gaines, "Stochastic computing systems," *Advances in Information Systems Science: Volume 2*, pp. 37–172, 1969.

[94] W. Poppelbaum, C. Afuso, and J. Esch, "Stochastic computing elements and systems," in *Proceedings of the November 14-16, 1967, fall joint computer conference*, 1967, pp. 635–644.

[95] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Transactions on Information Theory*, vol. 40, no. 3, pp. 716–720, 1994.

[96] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[97] W. J. Gross, V. C. Gaudet, and A. Milner, "Stochastic implementation of ldpc decoders," in *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.* IEEE, 2005, pp. 713–717.

[98] D. Zhang and H. Li, "A stochastic-based fpga controller for an induction motor drive with integrated neural network algorithms," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 2, pp. 551–561, 2008.

[99] T. Marconi and S. Cotofana, "Dynamic bitstream length scaling energy effective stochastic ldpc decoding," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 245–248.

[100] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *2013 IEEE 31st International Conference on Computer Design (ICCD).* IEEE, 2013, pp. 39–46.

[101] H. Chen and J. Han, "Stochastic computational models for accurate reliability evaluation of logic circuits," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, 2010, pp. 61–66.

[102] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1474–1486, 2012.

[103] A. Rapley, C. Winstead, V. Gaudet, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Proc. 3rd Int. Symp. Turbo Codes Related Topics*, 2003, pp. 507–510.

[104] J. Zhang and M. P. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, 2005.

[105] P. Hailes, "Design and implementation of flexible fpga-based ldpc decoders," Ph.D. dissertation, University of Southampton, 2018.

[106] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[107] M.-R. Li, C.-H. Yang, and Y.-L. Ueng, "A 5.28-gb/s ldpc decoder with time-domain signal processing for ieee 802.15. 3c applications," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 2, pp. 592–604, 2016.

[108] I. Tsatsaragkos and V. Paliouras, "A reconfigurable ldpc decoder optimized for 802.11 n/ac applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 182–195, 2017.

[109] N. Cheng and X. Shen, "Next-generation high-efficiency wlan," in *5G mobile communications*. Springer, 2016, pp. 651–675.

[110] R. Ramirez, C.-Y. Huang, and S.-H. Liang, "5g digital twin: A study of enabling technologies," *Applied Sciences*, vol. 12, no. 15, p. 7794, 2022.

[111] X. Jiang, Z. Pang, M. Zhan, D. Dzung, M. Luvisotto, and C. Fischione, "Packet detection by a single ofdm symbol in urllc for critical industrial control: A realistic study," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 4, pp. 933–946, 2019.

[112] N. Wang, N. Nouwell, C. Ge, B. Evans, Y. Rahulan, M. Boutin, J. Desmauts, K. Liolis, C. Politis, S. Votts *et al.*, "Satellite support for enhanced mobile broadband content delivery in 5g," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE, 2018, pp. 1–6.

[113] D. Krummacker, B. Veith, C. Fischer, and H. D. Schotten, "Analysis of 5g channel access for collaboration with tsn concluding at a 5g scheduling mechanism," *Network*, vol. 2, no. 3, pp. 440–455, 2022.

[114] F. Durukan, B. M. Güney, and A. Özen, "Performance analysis of color shift keying systems in awgn and color noise environment," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2019, pp. 1–4.

[115] K. Mochizuki and M. Uchino, "Efficient digital wide-band coloured noise generator," *Electronics Letters*, vol. 37, no. 1, pp. 62–64, 2001.

[116] M. Sanchez, E. Exposito, and J. Aguilar, "Industry 4.0: survey from a system integration perspective," *International Journal of Computer Integrated Manufacturing*, vol. 33, no. 10-11, pp. 1017–1041, 2020.

[117] E. O'Connell, D. Moore, and T. Newe, "Challenges associated with implementing 5g in manufacturing," in *Telecom*, vol. 1, no. 1. MDPI, 2020, p. 5.

[118] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[119] T. Thadewald and H. Büning, "Jarque–bera test and its competitors for testing normality–a power comparison," *Journal of applied statistics*, vol. 34, no. 1, pp. 87–105, 2007.

[120] S. K. Sharma, S. Chatzinotas, and B. Ottersten, "Snr estimation for multidimensional cognitive receiver under correlated channel/noise," *IEEE Transactions on Wireless Communications*, vol. 12, no. 12, pp. 6392–6405, 2013.

[121] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[122] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[123] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[124] F. Chahkoutahi and M. Khashei, "Influence of cost/loss functions on classification rate: A comparative study across diverse classifiers and domains," *Engineering Applications of Artificial Intelligence*, vol. 128, p. 107415, 2024.

List of Publications

1. Tera, Sivarama Prasad, Rajesh Alantattil, and Roy Paily. 2023. “A Flexible FPGA-Based Stochastic Decoder for 5G LDPC Codes” *Electronics* 12, no. 24: 4986. https://doi.org/10.3390/electronics12244986.

2. Tera, Sivarama Prasad, Ravikumar Chinthaginjala, Priya Natha, Giovanni Pau, C. Dhanamjayulu and Faruq Mohammad, “CNN-Based Approach for Enhancing 5G LDPC Code Decoding Performance,” in *IEEE Access*, vol. 12, pp. 89873-89886, 2024, doi: 10.1109/ACCESS.2024.3420106.

3. Tera, Sivarama Prasad, Ravikumar Chinthaginjala, Priya Natha, Shafiq Ahmad and Giovanni Pau, “Deep Learning Approach for Efficient 5G LDPC Decoding in IoT,” in *IEEE Access*, vol. 12, pp. 145671-145685, 2024, doi: 10.1109/ACCESS.2024.3472466.