Politecnico di Torino

Master's Degree in Electronic Engineering

Master's Degree Thesis

A Design Space Exploration Tool for LDPC Decoder Architectures in 5G NR Applications

Supervisors
Prof. Maurizio Martina
Prof. Guido Masera

Candidate
Alessandro Barrera

April 2023

Abstract

Modern communication systems require sophisticated channel coding to enable high performance and low complexity trade-off implementations. A wide variety of decoding techniques have been proposed throughout the years to meet these requirements, while Moore's law slows down and the improvements in silicon process technology are not sufficient to reach the desired data rates. Low-density parity-check (LDPC) codes are one of the most promising techniques in terms of error-correcting capabilities and throughput performance, but the abundance of existing architectures makes difficult to choose optimal solutions.

For this reason, this master thesis proposes a design space exploration tool, having the goal to offer a wide range of possible LDPC decoder architectures, comparing their performances and guiding an hypothetical decoder designer towards an effective implementation.

The tool's scope includes partially-parallel architectures implementing Min-sum algorithm and decoding Quasi-cyclic LDPC codes compliant with the 5G New Radio standard, based on base-graph 1. Results are obtained on a 45 nm technology library. The design space variables concern design choices such as scheduling scheme, building blocks' architectures, and techniques such as loop unrolling and pipelining, along with a few system parameters.

Estimations of the decoders' characteristics such as area, operating frequency and throughput are provided by the tool, exploiting an architectural model of the decoders, and either the direct synthesis results or an empirical model of the fundamental components of these architectures. The output obtained is a rapid and accurate overview of the decoder architectural solutions, highlighting their advantages and disadvantages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital communications permeate everyday tasks in modern society. Work, study, entertainment, almost every field of human activity heavily uses communication or at least obtains great benefits from it. This is reflected in the continuously increasing number of connected devices, wireless applications, and ultimately the increasing of communication requirements. As a matter of fact, data rates has increased with every new mobile communication generation, going from 1 Gb/s in LTE-A, to a range between 10 to 100 Gb/s in 5G, and towards 1 Tb/s is foreseen in Beyond-5G. Moreover, as new applications arise, many other features are required. Suffice it to consider Internet of Things, Virtual Reality, Memory Storage, all have very different needs in latency, throughput, area and power constraints.

Improvements in the silicon process technology has served for many years the goal of achieving better throughput, latency and power consumption. Unfortunately, as the Moore's law slows down, these are not sufficient to meet the future requirements, thus an approach which also includes algorithms and hardware implementations is needed. This issue impacts in particular forward error correction (FEC) techniques and the corresponding decoders since their complexity at high data rates makes their implementation a critical issue.

Low-density parity-check (LDPC) codes were proposed by R. Gallager in the 1960s [1], but were long ignored due to the considerable hardware requirements and decoding complexity. Following the development of Turbo-codes [2], they were rediscovered in 1992 by MacKay [3]. The excellent performances of LDPC compared to their relatively low complexity, made them suitable for a vast amount of applications. In fact, they are currently adopted in many standards, such as DVB-S2, 10GBASE-T Ethernet, Wi-Fi 802.11n, 802.11ac, 802.11x, and not least 5G New Radio (NR).

The design of LDPC decoders can be challenging because of the complexity of iterative decoding and the vast solution space that derives from it.

1.1 The proposed design space exploration tool

This work proposes a design space exploration tool capable of modeling several LDPC decoders, distinct in the combination of the design choices within the considered design space. For each of the modeled architecture, the tool provides area, frequency and throughput estimations.

This analysis is based on the area and delay characteristics of the building blocks of the considered architectures, which can either be retrieved from the syntheses of the exact designs needed in the LDPC decoder, or from a mathematical model built from fitted data.

The proposed approach consists in finding all the valid combinations, creating a database of LDPC decoder architectures. An architectural-algorithmical model provides for each of them an analysis which includes: critical path and maximum operating frequency estimations, analysis of the components list, memory requirements analysis, latency cycles and eventually area and throughput estimations.

This methodology constitutes a fast and low-investment analysis, while also easy to customize. It provides an overview of the possibilities that LDPC decoders can offer, thus it could guide a designer towards the most interesting solutions for their use case. On the other hand, it is not intended as a deep analysis on each solution and results must be considered as estimations to narrow down the solution space to a more contained subset. Finally, the tool can be easily customized to retrieve specific information on each studied architecture (e.g., the amount of units employed, or the percentage of area occupied by the memories), or to include different components' designs.

1.2 Organization

This thesis is organized as follows.

Chapter 2 - LDPC codes Presentation and description of the LDPC codes, Quasi-cyclic LDPC and LDPC in 5G NR standard.

Chapter 3 - LDPC decoding Description of LDPC iterative decoding. Decoding algorithms, parallelism, scheduling schemes and the corresponding architectures are presented.

Chapter 4 - Architectures in the design space Details on the architectures in the design space, including processing elements, routing networks and memories.

Chapter 5 - A design space exploration tool Presentation on the DSE tool, the proposed approach and its validation.

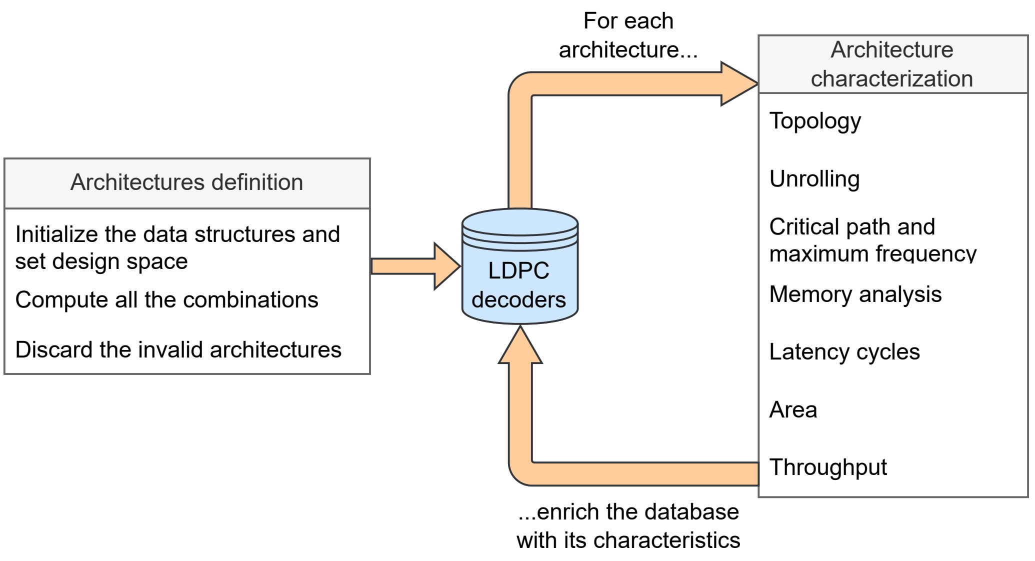

For each architecture...
Architecture characterization
Topology
Unrolling
Critical path and maximum frequency
Memory analysis
Latency cycles
Area
Throughput
Architectures definition
Initialize the data structures and set design space
Compute all the combinations
Discard the invalid architectures
LDPC decoders
...enrich the database with its characteristics

Figure 1.1: Organization of the tool's approach.

Chapter 2

LDPC codes

Low-density parity-check (LDPC) codes are error-correcting codes capable to approach the channel capacity.

This chapter provides the fundamentals required to understand the LDPC decoding. First, a brief background on channel coding is presented. Then, LDPC codes are introduced along with their description with parity-check matrices and Tanner graphs. Lastly, Quasi-cyclic LDPC and 5G NR standard are detailed, as they are of particular interest in the design space exploration tool application.

2.1 Background on channel coding

Transmitting a signal over a real communication channel inevitably introduces a noise, which is received with the proper signal. This perturbation can lead to transmission errors when retrieving the original information, interpreting bits by their opposite values.

A general communication system is composed as in Figure 2.1. The *encoder* and the *decoder* are used to employ a *forward error correction* technique, which is capable of detecting and correcting most of the errors that can happen during transmission over a noisy channel. In particular, given a K-bit message $\boldsymbol{m} = m_1\, m_2 \ldots m_K$, the encoder computes M redundancy bits, concatenating them to the original message and obtaining the N-bit codeword $\boldsymbol{c}$ to be modulated and transmitted.

A simple example of modulation is the *Binary Phase-Shift Key* (BPSK), which

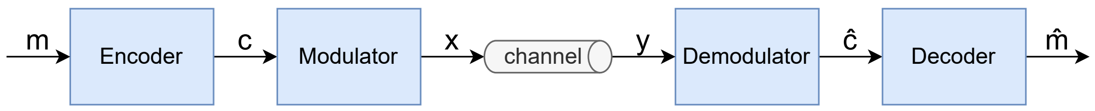


Figure 2.1: A general communication system.

consists in computing each modulated symbol x_i according to

$$x_i = \begin{cases} +\sqrt{E_s}, & \text{if } c_i = 0 \\ -\sqrt{E_s}, & \text{if } c_i = 1 \end{cases} \tag{2.1}$$

where E_s is the energy transmitted per symbol.

The $\boldsymbol{x}$ signal is sent through the noisy channel, usually modeled as an Additive White Gaussian Noise (AWGN) channel, which entails that the signal at channel output is given by

$$y_i = x_i + \varepsilon, \tag{2.2}$$

where $\varepsilon \sim \mathcal{N}(0, N_0^2)$ is the additive white noise contribution with normal distribution and noise power spectral density N_0.

The demodulator retrieves the codeword $\hat{\boldsymbol{c}}$ from the received $\boldsymbol{y}$ signal. This can consist in a hard decision according to the sign of the y_i value, but soft decision provides better error correcting capabilities of the decoder. In particular, it is usually implemented by using fixed point representation of *log-likelihood ratios* (LLR), which are presented and motivated in Section 3.1.

The decoder's task is to recover the message $\hat{\boldsymbol{m}}$ detecting and correcting the errors contained in the received codeword $\hat{\boldsymbol{c}}$. If $\hat{\boldsymbol{m}} = \boldsymbol{m}$, then the message is successfully being recovered and received. Some coding techniques are capable of just detecting errors, which requires a re-trasmittion of the codeword, but LDPC codes are also capable of correcting most of the errors, thus increasing performances and transmission throughput.

2.2 LDPC codes description

Low-density parity-check (LDPC) codes are forward error correction (FEC) codes, capable to approach the channel capacity. LDPC are also linear block codes, which means that when K information bits must be transmitted, M parity bits are computed as linear combinations of the information bits. Hence, the total number of transmitted bits is $N = K + M$. The *code rate* R is defined as $R = K/N = \frac{N-M}{N}$. This set of equations defines the parity-check matrix H, where each row represents a parity-check equation and each column represents a transmitted bit (whether it be an information bit or a parity bit). If the element $H(i,j) = 1$ it means that the j-th bit is present in the i-th equation. LDPC codes are characterized by a parity-check matrix H of dimensions $M \times N$, which is *sparse*, hence the name *low-density*.

A received message $\boldsymbol{c} = c_1\, c_2 \dots c_N$ is considered a *codeword* if the parity-check equations:

$$H\, \boldsymbol{c}^T = \mathbf{0} \tag{2.3}$$

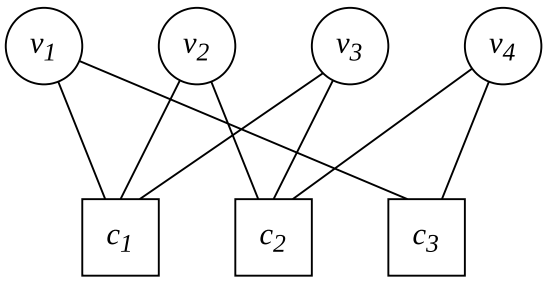


Figure 2.2: An example of Tanner graph

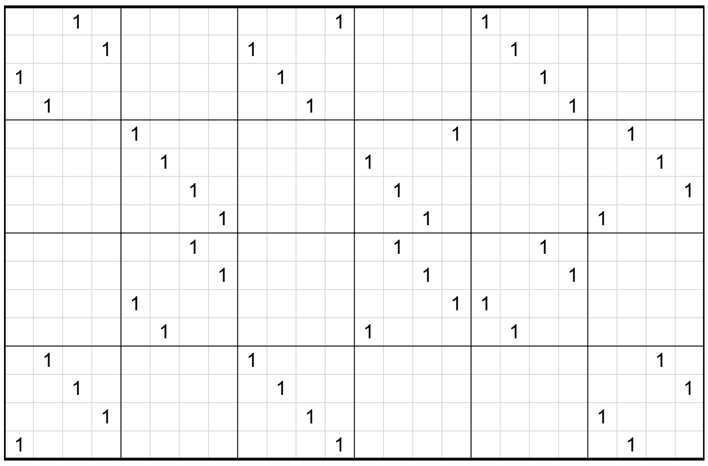
are satisfied in modulo 2 arithmetic. An example of a parity-check matrix is the following:

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The number of ones in a row is called the *row degree*, while the number of ones in a column is the *column degree*. If the row degree is the same for each row and the column degree is the same for each column the code is said to be *regular*, otherwise it is *irregular*. In order to have a good error correcting performance, the H matrix must be sparse, as previously said, and the code length should be of at least hundreds or thousands of bits. LDPC codes with lower code length are outperformed by other codes such as Turbo codes. Better performances are generally achieved in randomly constructed codes, but they are more difficult to implement in hardware. A good trade-off is achieved with Quasi-cyclic LDPC codes, presented in Section 2.3.

A popular description of LDPC codes consists in the Tanner graph [4]. This is a *bipartite graph*, which graphically represents the parity-check equations. The Tanner graph in Figure 2.2 corresponds to the same code of the parity check matrix in Equation (2.3). With reference to Figure 2.2, the circular vertexes are *variable nodes*, which represent indiscriminately either data bits or parity bits; the squared vertexes are *check nodes*, which represent the parity-check equations. An edge connects a variable node to a check node if the corresponding bit is present in the corresponding equation. Thus, there is a direct correspondence between the edges in the Tanner graphs and the '1' elements in the parity-check matrix H.

This graphical representation is useful to understand the idea behind the LDPC decoding, in particular message-passing algorithms. The matrix representation is going to be used in particular to understand the scheduling schemes.



(a) QC-LDPC parity-check matrix H.

2	-1	3	-1	0	-1
-1	0	-1	3	-1	1
-1	2	-1	1	2	-1
1	-1	0	-1	-1	2

(b) Corresponding base matrix H_p

Figure 2.3: Example of a QC-LDPC matrix description. Here $N_p = 6$, $M_p = 4$, $Z = 4$. Zeroes are omitted in (a) for clarity.

2.3 QC-LDPC codes

Quasi-cyclic LDPC (QC-LDPC) codes are characterized by a parity-check matrix H of dimension $M \times N$ composed of $Z \times Z$ sub-matrices which can either be null or circularly shifted identity matrices, also called permutation matrices. In order to compress the description of such a parity-check matrix, a base matrix H_p is used, where each element corresponds to a sub-matrix in H. In particular, a null matrix is represented by a negative element (usually -1), while the circularly shifted identity matrices are represented by the shift value, including possibly 0. The base matrix H_p has dimensions $M_p \times N_p$, which are in relation to the dimensions of the parity-check matrix H according to $N = N_p \cdot Z$ and $M = M_p \cdot Z$. An example of this is shown in Figure 2.3.

The utilization of such permutation matrices is particularly advantageous in hardware implementations since the code length can be very large, while having Z consecutive rows or columns which do not share ones in the same columns or rows, respectively. In other words, parallelism can be easily exploited guaranteeing no common elements in the parity equations. This is especially useful in the decoding of QC-LDPC, increasing throughput and achieving good error correcting performances. Moreover, the matrix structure can be more efficiently stored in memory using the base matrix representation H_p.

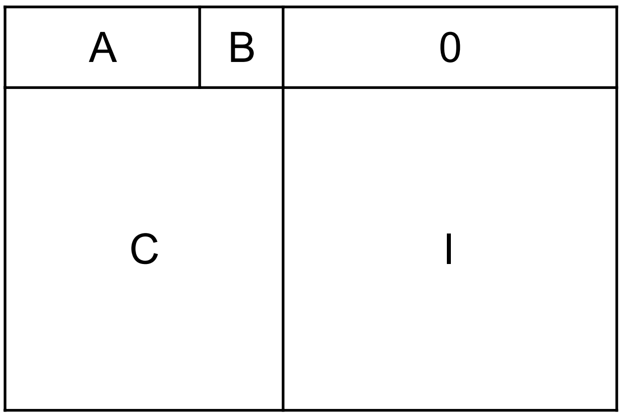


Figure 2.4: Base graph 1 structure

2.4 5G New Radio standard

5G New Radio (NR) is the standard defined by the 3GPP[1] for 5G networks. The first definition was published with release 15, where LDPC codes and Polar codes were chosen as channel coding schemes for data and control channels, respectively. An overview on LDPC codes compliant with the 5G NR standard is given in [5].

5G NR standard employs QC-LDPC codes, whose base matrix is officially called *base graph*, since it derives from the lifting procedure of a protograph. There are two predefined base graphs, namely base graph 1 (BG1) and base graph 2 (BG2), which are expanded by a lifting size Z chosen within a specific set, grouped by *permutation matrix design*. In [6], the description of the base graph corresponding to the permutation matrix design can be found.

Base graph 1 and 2 are distinct according to the desired code rate or code length. In particular, BG1 is best suited for larger code rates and longer block lengths. In fact, base graph 1 has a size of 46×68, obtaining code rate of $1/3$, while base graph 2 has size 42×52 and code rate $1/5$. In both cases, the first two information bits are not transmitted, as they are always *punctured*[2]. This work focuses on BG1.

The structure of BG1 is shown in Figure 2.4, where it is portioned in six sub-matrices. A is an irregular code and B has a dual-diagonal structure. Combined, they form the high-rate irregular repeat accumulate (IRA) code, with sub-matrix of size 4×26, code rate $R = 11/12$, and first two information bits punctured. The maximum row degree $d_c^{max} = 19$ is in this region. The sub-matrix C corresponds

[1]3rd Generation Partenership Project

[2]*punctured* bits are not considered in the code rate computation

Parameter	Value
N_p	68
M_p	46
R	1/3
tot_edges	$316 \cdot Z$
d_c^{max}	19
d_c	6.87
d_v^{max}	30
d_v	4.65

Table 2.1: BG1 parameters needed in the presented work's analysis

to the extension code. Along with the identity matrix I and the null matrix *0*, they are used to support the IR-HARQ (incremental redundancy-hybrid automatic repeat request).

The punctured bits are *information* bits, thus they must be recovered from the decoder (as opposed as if they were parity bits). In fact, the first two columns present a much higher column degree $d_v^{max} = 30$, which increases the chances of retrieving the correct values.

The base graph 1 was chosen in this work. Summarized in Table 2.1, the corresponding parameters necessary for the proposed analysis are: the base graph dimensions N_p and M_p, the total number of edges, the maximum and the average row degree, the maximum and the average column degree.

Chapter 3

LDPC decoding

LDPC decoders convert the channel information $\boldsymbol{y}$ in the binary message $\hat{\boldsymbol{c}}$, which must satisfy the parity equations in (2.3) in order to have the codeword detected. The traditional decoding technique for LDPC codes is an *iterative decoding* based on a *message-passing algorithm*, where check nodes and variable nodes exchange messages in the form of soft-information. This proved to enable better performance compared to decoding with hard-decision bits.

The decoding process of LDPC codes is often visualized exploiting the Tanner graph representation mentioned in Chapter 2, Section 2.2. During the decoding, variable nodes and check nodes exchange messages with their neighbouring vertexes, computing at each iteration the *log-likelihood ratio* (LLR) values on which hard-decisions are performed. With each iteration, LLR values become more accurate until the parity check equations are satisfied. Usually, *early-stop* techniques can be used to halt the sequence of iterations when a valid codeword is found, otherwise the iterations are repeated up to a fixed number of iterations.

Several *decoding algorithms* have been proposed, defining the operations that must be performed, in particular by check nodes and variable nodes, to decode the codeword. *Parallelism* determines how many processing units are employed and the *scheduling schemes* indicate the order with which these operations are performed.

LDPC decoding is presented in this chapter. Firstly, a brief recall on log-likelihood algebra is given. Belief propagation and Min-sum algorithms are then introduced, with emphasis on LLR domain. Additionally, the architectures' parallelism are defined. Then scheduling schemes are detailed, in particular flooding, sliced message-passing and row-layered schemes.

3.1 Log-likelihood algebra

When a symbol is demodulated, the received information could be expressed as hard decision or soft decision. While hard decision may seem the most intuitive solution, iterative decoding exploits soft values to enhance the decoding performance by exchanging information between decoding units. Usually, log-likelihood ratios are used as soft value

Log-likelihood ratios were first used in [7], where it is shown how the multiplicative operations needed in the iterative decoding can be avoided by moving to the logarithm domain, using additions instead, which are easier to implement in hardware.

Given a binary random variable x which can take the values in $\{0, 1\}$, the log-likelihood ratio $L(x)$ expresses how likely is that $x = 0$ over $x = 1$, and it is given by

$$L(x) = \log \frac{P(x=0)}{P(x=1)}.$$

The sign of $L(x)$ can be used as hard decision on the most likely value of x, and the magnitude of $L(x)$ expresses the confidence of this decision. Using the Bayes theorem, the LLR value on conditioned likelihood is given by:

$$\begin{aligned} L(x|y) &= \log \frac{P(x=0|y)}{P(x=1|y)} \\ &= \log \frac{p(y|x=0)}{p(y|x=1)} + \log \frac{P(x=0)}{P(x=1)} \\ &= L(y|x) + L(x). \end{aligned} \tag{3.1}$$

In [7] it is shown how to compute the LLR value $L(x_1 \oplus x_2)$

$$L(x_1 \oplus x_2) = \log \frac{1 + e^{L(x_1)} e^{L(x_2)}}{e^{L(x_1)} + e^{L(x_2)}},$$

which can be extended in the general case:

$$L\left(\sum_i \oplus x_i\right) = 2 \tanh^{-1}\left(\prod_i \tanh\left(L(x_i)/2\right)\right). \tag{3.2}$$

This expression is fundamental for the *extrinsic* information calculation as detailed below.

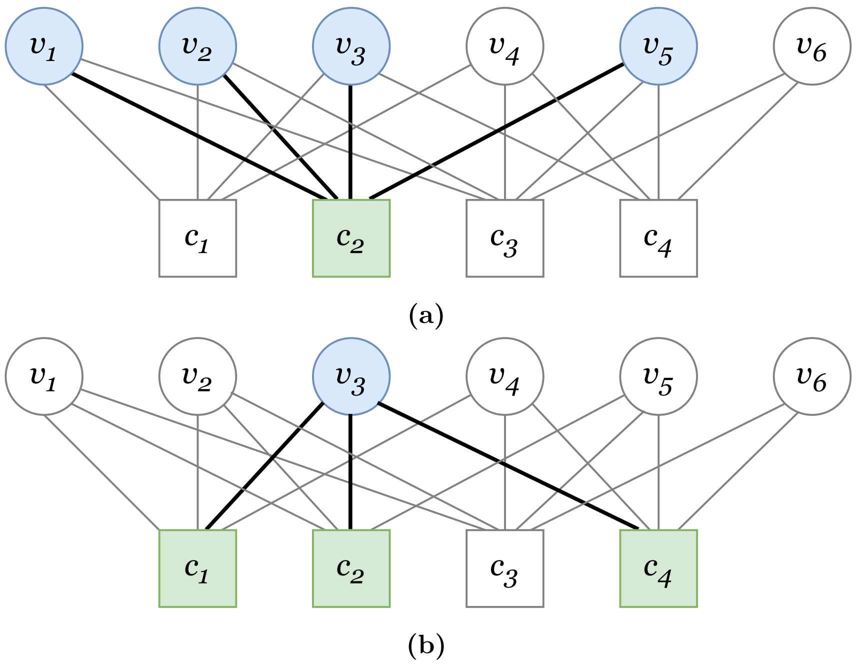


Figure 3.1: The Tanner graph in (a) highlights the set $S_v(2)$ of variable nodes connected to check node 2. Similarly does the Tanner graph in (b) for the set $S_c(3)$ of check nodes connected to variable node 3.

3.2 Decoding algorithms

Message-passing algorithms are usually employed in LDPC decoding, consisting in an iterative decoding technique where information is exchanged between check nodes and variable nodes, as defined in the Tanner graph (see Chapter 2, Section 2.2). Here the decoding process is briefly presented, but a more detailed and rigorous elaboration on the theory behind it can be found in [8].

Assuming K is the message length, M is the number of parity bits and $N = K + M$ is the length of the codeword. Let $\boldsymbol{c} = (c_1, c_2, \dots, c_N)$ denote the codeword to be modulated and sent through the channel, and $\boldsymbol{y} = (y_1, y_2, \dots, y_N)$ the received values. The soft value y_n at the channel output is related to the n-th bit c_n transmitted. The *variable-to-check* message sent from variable node n to check node m is indicated with $\alpha_{m,n}$, while the *check-to-variable* message sent from check node m to variable node n is denoted with $\beta_{m,n}$. The set of check nodes connected to variable node n is indicated with $S_c(n)$, while the set of variable nodes connected to check node m is indicated with $S_v(m)$.

Considering the soft input y_n to a decoding unit, it is possible to express the likelihood of receiving that signal value given the transmitted bit was $c_n = 0$ or $c_n = 1$ as $p(y_n|c_n = 0)$ and $p(y_n|c_n = 1)$, respectively. The *a posteriori* likelihood of the sent bit can be computed according to the Bayes theorem as:

$$P(c_n|\boldsymbol{y}) = \frac{p(y_n|c_n)\,P(c_n|\{y_i, i \neq n\})}{p(y_n)}. \tag{3.3}$$

Moving to the logarithm domain, this relation can be expressed using Equation (3.1) as

$$\begin{aligned}\tilde{\gamma}_n &= \log\frac{P(c_n = 0|\boldsymbol{y})}{P(c_n = 1|\boldsymbol{y})}\\ &= \log\frac{p(y_n|c_n = 0)}{p(y_n|c_n = 1)} + \log\frac{P(c_n = 0|\{y_i, i \neq n\})}{P(c_n = 1|\{y_i, i \neq n\})}\\ &= \gamma_n + L(c_n).\end{aligned} \tag{3.4}$$

This quantity is the *a posteriori* LLR information of bit c_n. The sign expresses whether the bit was most likely a 0 or a 1, while the magnitude expresses the confidence of this guess. Two contributions can be identified: the channel information γ_n and the *a priori* information $L(c_n)$. The channel information is only determined by the measure of y_n and the channel characteristics. For example, in case of an AWGN channel $\gamma_n = 2\sqrt{E_s}/\sigma^2\, y_n$.

Assuming $P(c_n = 0) = P(c_n = 1)$, the *a priori* probability is initially zero. In iterative decoding, *extrinsic* information is exchanged among decoders and is used as *a priori* information in the next iteration. This process can either increase the likelihood magnitude, reinforcing the confidence on the guess, or reduce it until $\tilde{\gamma}_n$ changes sign, which is an error correction.

The *extrinsic* information is derived from *all* the parity check equations, taking into account the information on bit c_n that can be retrieved from the channel values of the *remaining* bits. For this reason, check-to-variable message $\beta_{m,n}$ is computed using the information from the messages $\alpha_{m,j}, j \in S_v(m), j \neq n$, as shown in Figure 3.2. Likewise, this information comes from the *a posteriori* probabilities computed in the variable nodes, which include the channel information. The iterations would accumulate the channel information incurring into a bias, thus it is excluded in the variable-to-check message $\alpha_{m,n}$ computation, taking into account $\beta_{i,n}, i \in S_c(n), i \neq m$. This is shown in Figure 3.3.

3.2.1 Belief propagation algorithm

Also known as *Sum-product*, *Belief propagation* (BP) algorithm ([9], [10]) is the traditional decoding method in LDPC codes. The usage of LLR values greatly simplifies the implementation of this algorithm. In fact, several multiplications and additions would be required in BP, both for the check node and variable node operations. Thus, opting for the logarithm domain, multiplications become

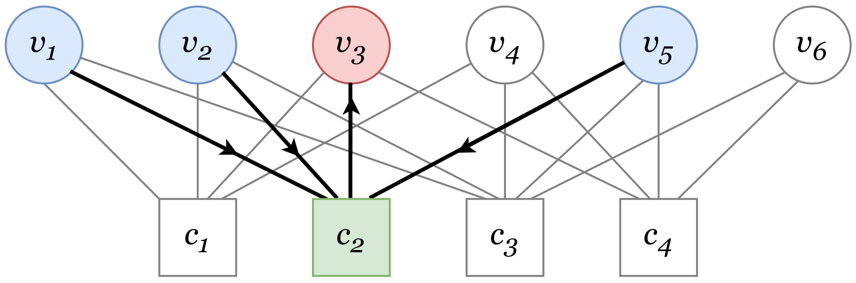


Figure 3.2: Example of check-to-variable message computation $\beta_{2,3}$. Check node 2 computes this message considering the $\alpha_{2,j}$, excluding the one that comes from the recipient variable node 3.

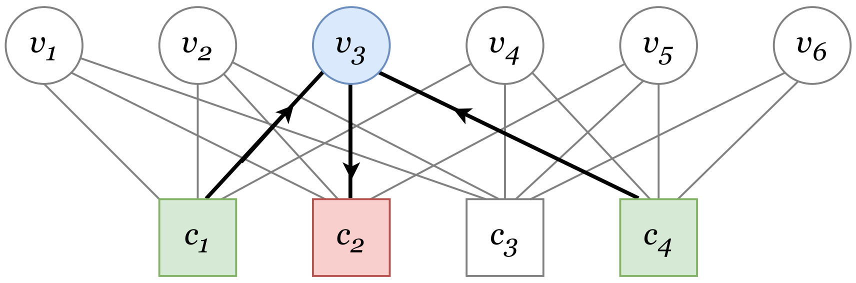


Figure 3.3: Example of variable-to-check message computation $\alpha_{2,3}$. Variable node 3 computes this message considering the $\beta_{i,3}$, excluding the one that comes from the recipient check node 2.

additions and both the α and β-messages computations are simplified. More information can be found in [7] and [11].

At the start of the decoding, no *a priori* information is available, thus the input to the decoder consists solely in the LLR channel information

$$\gamma_n = \log \frac{p(y_n|c_n = 1)}{p(y_n|c_n = 0)},$$

where y_n is the value received from the channel. This information is used to initialize the variable-to-check messages $\alpha_{j,n}$, where $j \in S_c(n)$.

The check-to-variable message $\beta_{m,n}$ from check node m to variable node n is computed using the variable-to-check messages $\alpha_{m,j}, j \in S_v(m), j \neq n$ as

$$\beta_{m,n} = 2 \tanh^{-1} \left(\prod_{j \in S_v(m), j \neq n} \tanh\left(\alpha_{m,j}/2\right) \right). \tag{3.5}$$

This calculation can be implemented in hardware using lookup tables (LUTs), but the large number of units required to achieve high-throughput dramatically increases the complexity and is therefore very limited [11].

Variable node n receives the messages $\beta_{j,n}, j \in S_c(n)$, and uses them to compute the variable-to-check messages. Symmetrically to the check node, the $\alpha_{m,n}$ computation excludes the message $\beta_{m,n}$ from its calculation:

$$\alpha_{m,n} = \gamma_n + \sum_{i \in S_c(n), i \neq m} \beta_{i,n}. \tag{3.6}$$

Moreover, the *a posteriori* information is computed as:

$$\tilde{\gamma}_n = \gamma_n + \sum_{i \in S_c(n)} \beta_{i,n}.$$

Here it is possible to notice the reformulation

$$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n},$$

which can be exploited in some architectural solutions. The *a posteriori* information $\tilde{\gamma}_n$ expresses the confidence on bit c_n and its sign can be used as a hard decision. This can be used against the parity check equations in (2.3). Therefore, the decoding is repeated either until a valid codeword is obtained or up to a fixed number of iterations.

3.2.2 Min-Sum algorithm

A simplification on the Belief propagation algorithm can be obtained in the computation of the $\beta_{m,n}$ messages. In [7] the following approximation is proposed for Equation (3.2):

$$\begin{aligned} L\left(\sum_i^{\oplus} c_i\right) &= 2\tanh^{-1}\left(\prod_i \tanh L(c_i)/2\right) \\ &\approx \left\{\prod_i \operatorname{sign}[L(c_i)]\right\} \cdot \min_i\{|L(c_i)|\}. \end{aligned}$$

The *Min-sum* (MS) algorithm uses this approximation in the calculation of the check-to-variable messages $\beta_{m,n}$, obtaining

$$\beta_{m,n} = \left\{\prod_{j\in S_v(m), j\neq n} \operatorname{sign}(\alpha_{m,j})\right\} \cdot \min_{j\in S_v(m), j\neq n}\{|\alpha_{m,j}|\}. \tag{3.7}$$

The $\alpha_{m,n}$ and $\tilde{\gamma}_n$ remains the same as in the BP algorithm.

This approximation leads to a great reduction in the complexity of the decoding algorithm. In particular, it now consists in the search for the minimum magnitude $|\alpha_{m,j}|, j \in S_v(m), j \neq n$ and the product of the signs. This is an important improvement with respect to the expression in Equation (3.5).

3.2.3 Improvements of Min-sum algorithm

The Min-sum algorithm reduces the complexity of the Belief propagation algorithm, but this improvement comes with the cost of error correcting performance, usually measured in bit error rate (BER). Two solutions were proposed to mitigate this loss: one is known as the *Normalized Min-sum* (NMS), while the other is known as the *Offset Min-sum* (OMS) algorithm [12]. Normalized Min-sum algorithms correct the check-to-variable message computation in Equation (3.7) by scaling of a factor $S < 1$,

$$\beta_{m,n} = S \cdot \left\{\prod_{j\in S_v(m), j\neq n} \operatorname{sign}[\alpha_{m,j}]\right\} \cdot \min_{j\in S_v(m), j\neq n}\{|\alpha_{m,j}|\}.$$

In Offset Min-sum, the magnitudes of variable-to-check messages are reduced by an offset $\lambda > 0$ before being compared, thus obtaining

$$\beta_{m,n} = \left\{\prod_{j\in S_v(m), j\neq n} \operatorname{sign}[\alpha_{m,j}]\right\} \cdot \min_{j\in S_v(m), j\neq n}\{|\alpha_{m,j}| - \lambda, 0\}.$$

In either case, the S and λ values must be derived by simulations. Hardware implementations may consider to approximate these values in order to simplify their operations. For example, NMS may exploit a shift operation to scale by a factor $S = 2^{-k}, k \in \{1, 2, \dots\}$, or an efficient implementation for S values with a small number of ones in their binary representation [11]. With the optimized factors, NMS and OMS reach BER very close to the BP algorithm, while being only a little more complex than MS.

3.3 Decoding parallelism

Check nodes and variable nodes are processed by computing the messages as detailed in Section 3.2. These computations are performed by specialized processing elements denoted check node units (CNUs) and variable node units (VNUs), respectively. The number of instances of these units determines the architecture *parallelism*, which is a critical parameter for the overall decoder performance.

Three categories of architectures are discriminated according to their parallelism: fully-parallel, serial and partially-parallel. Also, unrolled fully-parallel architectures have been proposed to increase the achievable throughput, as required in future standards [13].

Fully-parallel architectures have as many CNUs and VNUs as the number of rows and columns of the parity-check matrix H, respectively. This can also be thought as a direct correspondence between the nodes in the Tanner graph and the CNUs and VNUs in the decoder. The advantages of this architecture are granted by having a dedicated processing element for each node, thus providing the highest parallelization degree, very high throughput and no penalty brought by irregular codes. On the other hand, it lacks in flexibility as it can only decode one specific code. Moreover, the hardwired connections among the units lead to routing congestion problems.

Unrolling and pipelining this architecture allows to reach even higher throughput, possibly over the 1 Tb/s mark, making it the high throughput solution for Beyond-5G LDPC codes in the European Horizon 2020 EPIC project as discussed in the EPIC deliverable [13]. This parallelism also alleviates the routing congestion problem thanks to the usage of local connections.

Serial architectures have one CNU and one VNU, which implement the check node and variable node operations in a time-multiplexed way. The two processing elements exchange messages through a single memory. This solution is extremely flexible, but achievable throughput is too limited and it is not sufficient for most applications.

Partially-parallel architectures are a trade-off between the two, and is the most popular solution in today's decoders. They consist in multiple CNUs and

VNUs, but still different nodes need to share the same unit. This solution requires a *scheduling* of the operations, meaning the order with which rows and columns are processed. A critical part of these architectures is the interconnections among these units, which must be routed according to the nodes being processed. They can be complex and occupy a large portion of the total area, but are still less problematic than the fully-parallel hardwired interconnections. These routing networks can be simplified in the case of QC-LDPC codes, implementing the circularly-shifted identity matrix as barrel shifters, each taking care of one sub-matrix. This also enables a higher parallelization within block rows and block columns corresponding to a sub-matrix of the complete parity-check matrix.

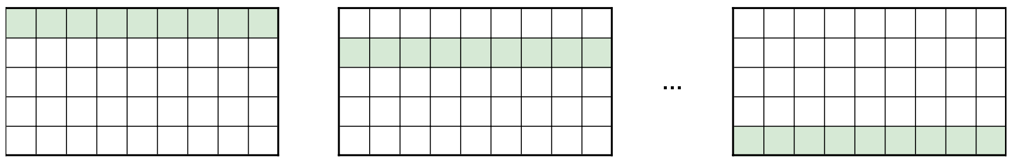
The proposed design space exploration tool focuses on partially-parallel architectures and on QC-LDPC codes compliant with 5G NR standard.

3.4 Scheduling schemes

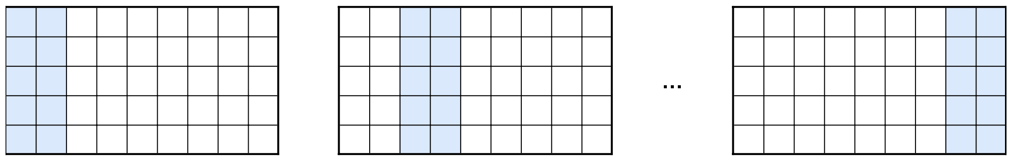
By definition, partially-parallel decoders have a certain number of CNUs and VNUs which are not sufficient to process all check and variable nodes at the same time. For this reason, a **scheduling scheme** must be employed, determining which operations are served and when. It also determines which operations can be scheduled in parallel, affecting latency, memory requirements, convergence speed and more. The scheduling schemes presented in this section are considered in the design space of the proposed tool. The parity-check matrix representation of a QC-LDPC code is used throughout this section, where each block row and block columns correspond to a row and column of sub-matrices. In other words, each row of the shown matrices corresponds to Z rows, where Z is the lifting-size. Similarly with columns. Thanks to the QC-LDPC codes construction, each block row and column elaboration can be parallelized granting no conflict. Thus it is possible to simplify the analysis considering the base matrix and then expanding rows and columns in sub-matrices.

3.4.1 Flooding scheme

The simpler scheduling is called the **flooding scheme** [14]. According to this scheduling scheme, the decoding iteration is divided in two phases: firstly, all check nodes are processed by the CNUs, while the VNUs are idle; then all variable nodes are processed by the VNUs, while the CNUs are idle. This scheduling is also the only possible for fully-parallel architectures, since the VNUs must wait for the β-messages before computing the α-messages and vice versa. An example of this type of scheduling is shown in Figure 3.4a, where the scheduling of check node operations in the first phase is shown. During this phase, VNUs are all idle.



(a) Example of check node processing



(b) Example of variable node processing

Figure 3.4: Flooding scheme. Check nodes and variable nodes are processed alternately. In this example, in the first phase **(a)** check nodes are processed one block row at the time. In the second phase **(b)** variable nodes are processed two block columns at the time.

Similarly, Figure 3.4b shows the second phase of the decoding iteration, during which CNUs are idle and variable nodes are processed. At this point, the input parallelism of either CNUs or VNUs is not specified yet, which could entail that multiple cycles are needed to elaborate each row/column.

An improvement could be obtained by interleaving two frames, so that when the CNUs are elaborating frame 1 the VNUs elaborate frame 2, and then exchange messages at the same time. This is more efficient if check node and variable node processing have the same latency, considering both the number and the latency of the processing elements. In this work, interleaving is not considered.

The flooding scheme can be implemented with the simple architecture shown in Figure 3.5, which defines the first building blocks of any LDPC decoder. In particular, the following elements can be distinguished:

CNUs and VNUs These are the processing elements that compute the α and β-messages required in the Min-sum algorithm. The number of these components and their input parallelism determine the latency in both check nodes and variable nodes processing.

Routing networks There are two sets of networks, which perform reciprocal routings. The one denoted *routing network* is used to drive the variable-to-check messages towards the appropriate check node, while the *reverse routing network* drives back the check-to-variable messages towards the variable nodes. This routing operations are implemented exploiting barrel shifters to reproduce the circularly-shifted identity matrix in the QC-LDPC code. Note

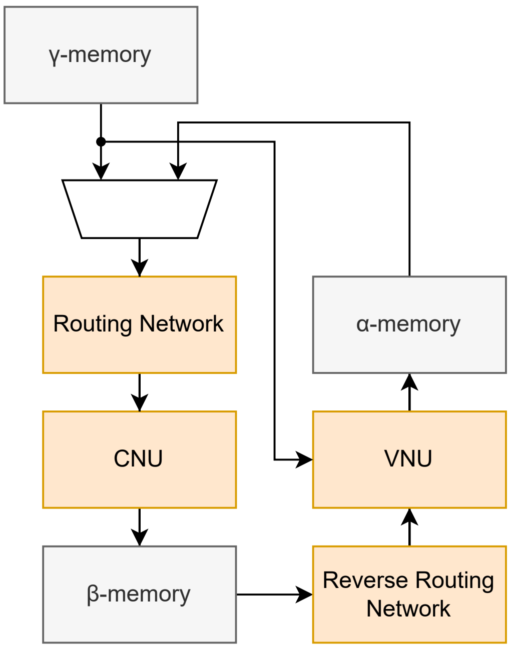


Figure 3.5: Decoder architecture with flooding scheme scheduling.

that the shifting coefficient must be read from a read-only memory which stores the parity-check matrix compact representation.

Memories Here three types of memories are shown. The γ-memory is used to store the channel information, which is required throughout the whole decoding to compute the variable-to-check messages according to Equation (3.6). The α-memory and the β-memory store the variable-to-check α messages and check-to-variable β messages, respectively. In different scheduling schemes, the *a posteriori* information $\tilde{\gamma}$ could be stored instead of the γ values.

3.4.2 Sliced message-passing scheme

Sliced message-passing scheme was presented in [15], proposing a scheduling where check nodes and variable nodes of successive iterations are processed in parallel. In particular, during the decoding iteration i, VNUs may have completed the processing of a few columns, thus the corresponding α-messages have been completed and are typically stored in the α-memory, but some more variable-to-check messages are still to be computed. If CNUs read only some input messages per cycle, they could start to elaborate these α-messages for the next iteration $i + 1$ while other variable nodes are still being processed. Then, they complete the β-messages when the last α-messages are delivered.

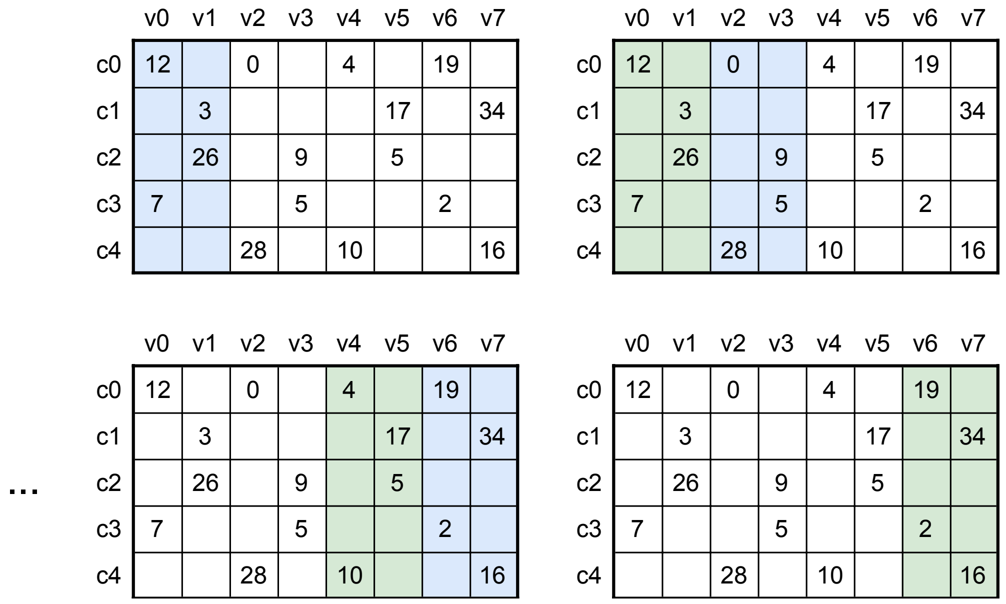


Figure 3.6: Sliced message-passing. In this example, there are as many CNUs as check nodes, elaborating up to two input messages per cycle. VNUs process the complete block column in one cycle and computes up to two α-messages per row in each cycle.

An example of this scheduling is reported in Figure 3.6, where the evolution of check nodes and variable nodes processing is highlighted in green and blue, respectively. Here two block columns are elaborated at a time by the VNUs. After the first cycle, the generated α-messages are immediately consumed by the CNUs. In this example, there are as many CNUs as the check nodes, thus elaborating all the rows in parallel for the next iteration, reading only up to two input per cycle. After the last α-messages are computed, the CNUs complete the β-messages for iteration $i + 1$, and VNUs can start the processing for iteration $i + 1$ with only one cycle of idle.

It is worth noting that the decoding iteration starts with check node processing, thus VNUs are idle for the whole first phase. Then, the overlapping between VNUs processing iteration i and CNUs processing iteration $i+1$ have each of them idle for one cycle per iteration. Eventually, variable nodes are processed while CNUs are idle for the last iteration. Nonetheless, the idle cycles are greatly reduced compared to flooding scheme, increasing the achieved throughput.

The architecture for this scheduling scheme is very similar to the flooding scheme, except that the β-memory must be doubled in size in order to permit to write the β-messages for iteration $i + 1$ and to read the ones from iteration i at the same time, instead of overwriting the same memory location. Nonetheless, if all the

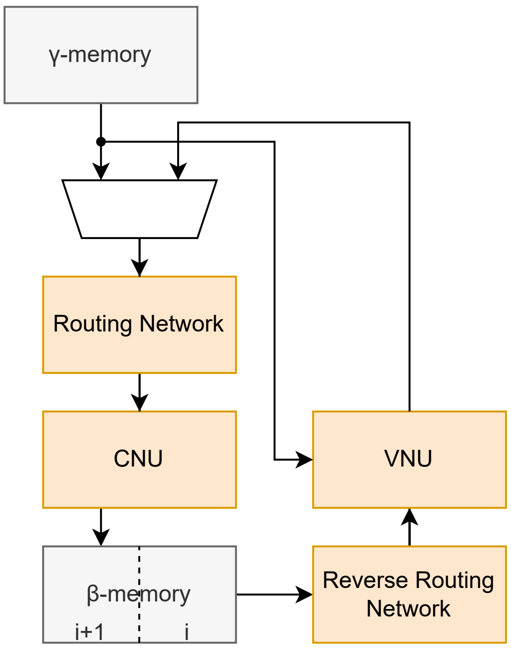


Figure 3.7: Decoder architecture with sliced message-passing scheme

α-messages are immediately consumed, then they do not need to be stored, thus the memory requirement increase can be attenuated by removing the α-memory. Such architecture is shown in Figure 3.7.

3.4.3 Row-layered scheme

The most popular decoding scheme is the layered approach. This is the solution with lower complexity, but still very high throughput thanks to its higher convergence speed. These properties made LDPC decoding suitable for multiple applications, proved by its inclusion into several communication standards. In this work, only row-layered scheme is considered, but also a column-based approach exists.

Row-layered scheme [16] consists in alternating the processing of check nodes and variable nodes in layers. In particular, in row-layered scheme, one block row is a layer. This is processed generating check-to-variable messages which are used to update the variable-to-check messages, immediately used in the next layer.

In QC-LDPC codes, layers traditionally correspond to the rows of the sub-matrices, exploiting their property of either being a circularly shifted identity matrix or a null matrix. This entails that there are no columns with two or more ones. In Tanner graph terms, each variable node is connected at most to one check node. Such property enables parallel processing as the α-messages should only update at most one β-message. As previously, henceforth the base matrix is analyzed for simplicity.

The idea behind this type of scheduling is to update the variable-to-check messages every time a new β-message is computed. The new α-message is used in the new layer, instead of the next iteration. Let us denote the variable-to-check message $\alpha^{(k,l)}$ and the check-to-variable message $\beta^{(k,l)}$ computed in layer l, during the decoding iteration k. Using Equation (3.6), this translate to the computation:

$$\begin{aligned}\alpha^{(k,l)} &= \gamma + \sum_{j=0}^{l-1} \beta^{(k,j)} + \sum_{j=l+1}^{M_p-1} \beta^{(k-1,j)} \\ &= \gamma + \sum_{j=0}^{l-2} \beta^{(k,j)} + \beta^{(k,l-1)} + \sum_{j=l}^{M_p-1} \beta^{(k-1,j)} - \beta^{(k-1,l)} \\ &= \alpha^{(k,l-1)} + \beta^{(k,l-1)} - \beta^{(k-1,l)}\end{aligned}$$

This relation shows that the α-message at each layer can be computed as the α-message at the previous layer, adding $\beta^{(k,l-1)}$ computed in the previous layer, and subtracting $\beta^{(k-1,l)}$, which is the message of the current layer computed in the during the last iteration. It is worth noting that the *a posteriori* LLR can be computed as

$$\tilde{\gamma}^{(k,l-1)} = \alpha^{(k,l-1)} + \beta^{(k,l-1)}, \tag{3.8}$$

thus finally

$$\alpha^{(k,l)} = \tilde{\gamma}^{(k,l-1)} - \beta^{(k-1,l)}, \tag{3.9}$$

as already stated previously. Updating the α-messages more often, the convergence speed with this scheduling scheme increases, requiring about 50% of the iterations.

An example of the row-layered scheduling is shown in Figure 3.8. The first two matrices shows the VNU and CNU elaborations of the first layer. The α-messages computed are used within the same layer immediately. The CNUs compute the β-messages which are used to update the *a posteriori* information addressed to the next layer. These sub-iterations are repeated through the whole matrix, using the information derived from the last layer as input to the first layer in the next iteration.

Figure 3.9 shows the architectural overview of a decoder with row-layered scheduling scheme. It presents a few different elements than the previous architectures, in particular due to the variable node processing. The $\tilde{\gamma}$-memory is initialized with the channel information γ. These values are used to initialize the variable-to-check messages, but they are not needed in the successive variable node

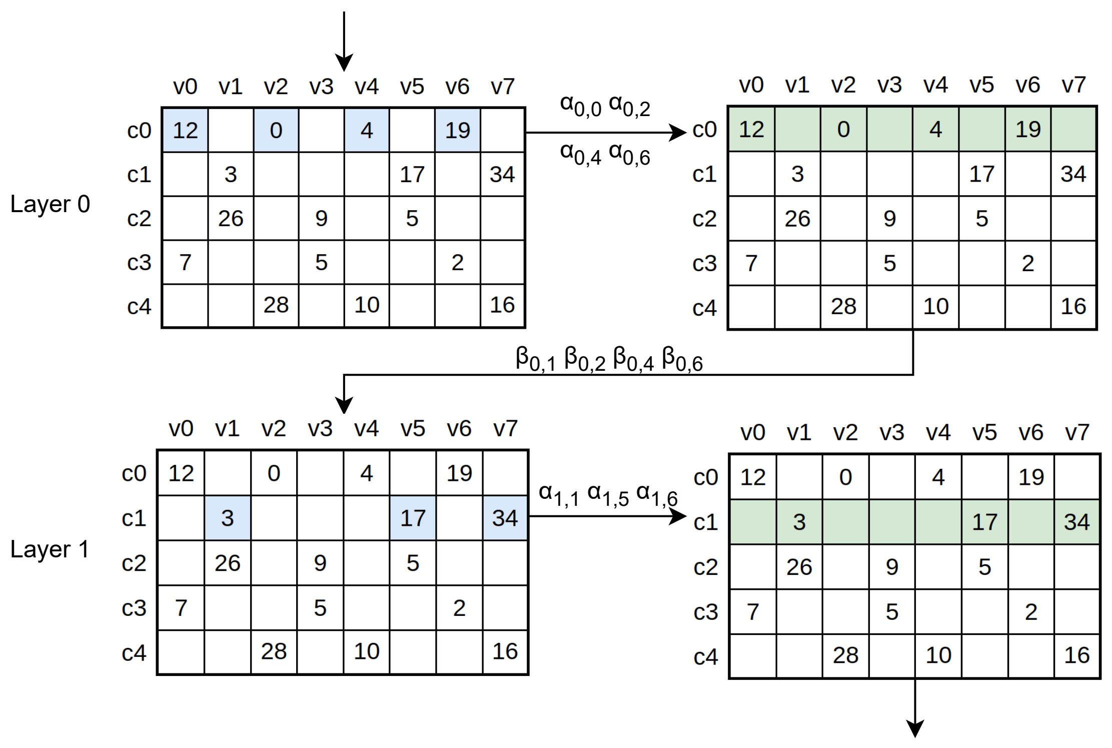


Figure 3.8: Row-layered scheme. The first two layers elaboration is shown, in particular the α-messages are updated and immediately sent to the CNUs, computing the β-messages. These are used in to update the $\tilde{\gamma}$ and in turns the α-messages in the next layer.

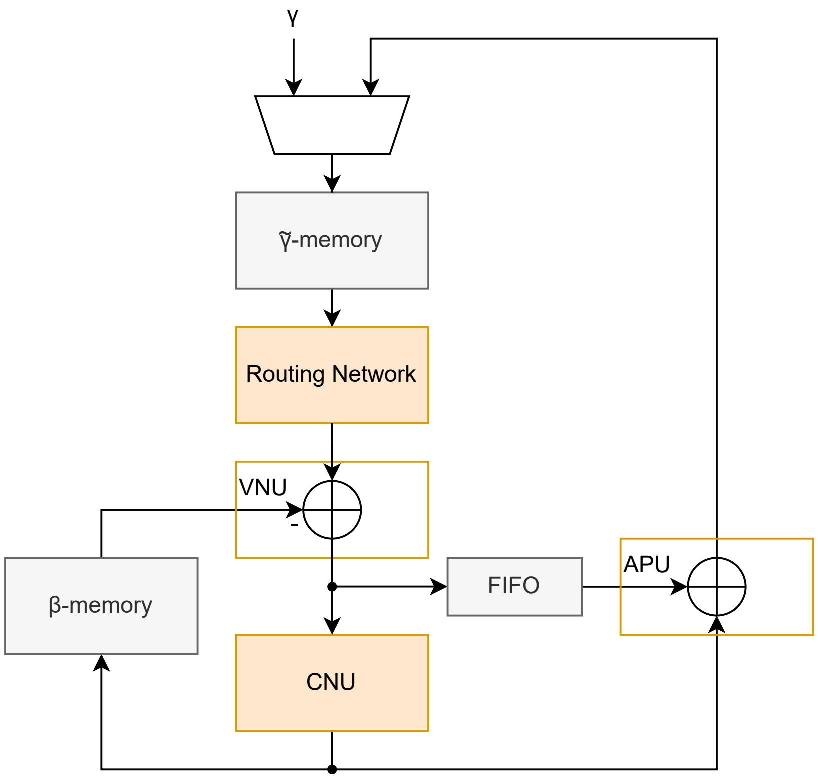


Figure 3.9: Decoder architecture with row-layered scheduling scheme

elaborations, as their information is contained in the *a posteriori* LLR probabilities, used in Equation (3.9). Thus, $\tilde{\gamma}$ values are updated and stored in the corresponding memory for each layer.

The VNU computes the $\alpha^{(k,l)}$ according to Equation (3.9), which is sent to both the CNU and a FIFO. This buffer stores the messages for the same number of cycles as the CNU latency, namely until they are needed for the *a posteriori* information update. The *a posteriori update unit* (APU) computes Equation (3.8), as it performs the addition of $\alpha^{(k,l-1)}$ and $\beta^{(k,l-1)}$ messages. This message is stored in the $\tilde{\gamma}$-memory, as previously stated.

The β-messages are stored in the β-memory as before, and are read when the same layer is elaborated during the next iteration from the VNU, which only computes Equation (3.9), where a simple subtraction is needed. In other words, the overall variable node processing is divided in two steps, each implemented with a simple addition/subtraction. This greatly simplifies the VNU operation compared to the Min-sum operation in Equation (3.6).

Chapter 4

Architectures in the design space

The design space exploration (DSE) tool exploits the area and delay characteristics of the building blocks which compose the final LDPC decoders. These values are obtained by synthesis of the HDL designs via **Synopsys Design Compiler**. Alternatively, a few synthesis can be performed to extract a mathematical model of the area and delay characteristics of certain building blocks against some parameter.

Regardless of the method employed, the main building blocks must be designed, tested and synthesized at least a few times. Each component may have a few possible design choice, thus in the following sections the explored architectures are presented. Check node units (CNU) are chosen with three different input parallelism: single-input, full-row or half-row. Variable node units (VNU) are designed either full-column for the flooding and sliced message-passing scheduling, or the special VNU architecture for layered scheme. These scheduling schemes are presented in Chapter 3, Section 3.4. Routing networks are modeled exploiting the synthesis of barrel shifters, which are the predominant part of these structures. Lastly, memories are modeled taking into account the memory organization needed to avoid memory access conflicts.

All these architectures are designed to be compliant with the 5G New Radio standard, presented in Chapter 2, Section 2.4. Some parameters of interest are the number of columns of the base matrix N_p, its number of rows M_p, the maximum row degree d_c^{max}, the maximum column degree d_v^{max} and lifting size Z. It is worth noting that it is possible to design different components' architectures and to integrate their characteristics within the tool, in case the user wants to expand the evaluated design space or they are interested in different architectures from those presented.

4.1 CNU architectures

The proposed DSE tool considers three CNU architectures according to their input parallelism: single-input, full-row and half-row. According to the employed architecture, the overall check node operation latency is affected and the DSE tool must consider how each solution impacts the whole circuit.

The CNU must perform the check node operations as presented in Chapter 3, Section 3.2.2. The variable-to-check α messages are provided in sign-magnitude representation in order to simplify the check node operation. Moreover, to reduce the memory requirements, the check-to-variable β messages are stored in *compressed form*, meaning that for each check node, the CNU must compare all the α-messages, finding its minimum, second to minimum, index of the first minimum and the signs of the β-messages. Before the actual VNU, a *decompression* unit followed by a sign-magnitude to two's complement conversion unit will take care to retrieve the actual check-to-variable message starting from the variable node index.

The memory saving is effective. In fact, considering a row weight d_c^{max} and word length of the message w, the total memory required to store the complete check-to-variable messages would be of $d_c^{max}\,w$ bits for each check node. Using compressed messages, only $2\,(w-1) + \lceil\log_2 d_c^{max}\rceil + d_c^{max}$ bits are needed for each check node. Using as an example the maximum row weight in 5G NR standard $d_c^{max} = 19$, a word length of $w = 5$ bits, the non compressed messages would need 95 bits against the 32 bits needed for the compressed messages. It is worth noting that a further compression is possible storing only the sign product of the α-messages, but this implies that if the variable-to-check messages are not stored, their signs are still needed to recover the actual check-to-variable messages in the VNUs.

In this section, the CNU architectures are presented, focusing on the calculation of min1, min2 and idx1. The signs computation is omitted for simplicity, since its computation is trivial.

4.1.1 Single-input CNU

At each clock cycle, the CNU takes one variable-to-check message, compares its magnitude to the temporary first and second minimums, possibly updating the new minimums and/or the corresponding variable node index. When a new input is received, its magnitude can be smaller than min1 and min2, between these two values, or larger than min2. These three cases can be easily distinguished with two comparisons as in the architecture shown in Figure 4.1. Two *compare* units, each taking the input variable-to-check message magnitude and either min1 or min2, are used to select the proper input to the minimum registers. Also the minimum index idx1 is updated when a new min1 is found.

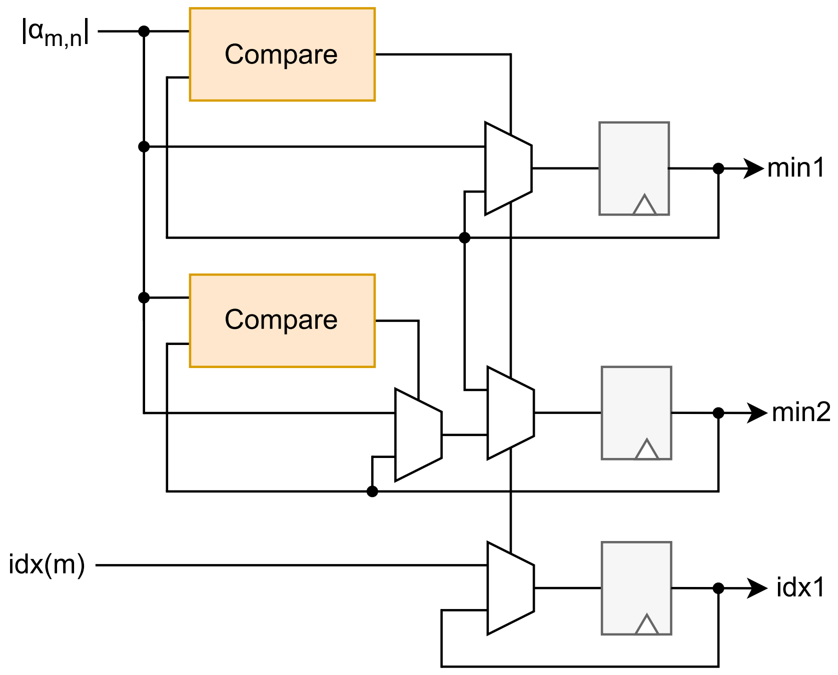


Figure 4.1: Single-input sorter CNU.

4.1.2 Full-row CNU

This architecture must compare all the variable-to-check messages at the same time, computing the compressed check-to-variable message. This operation is often implemented exploiting comparisons in a tree organization. In particular, the two most popular solutions are compared in [17].

The solution implemented in this work is the one proposed in [17], where it is referred to as the *tree structure approach*, and it is showed how it uses more comparisons with respect to the conventional solution ([18]), but greatly simplifies the min2 computation. The idea behind this architecture is to compare two variable-to-check message values in the two-input *compare and switch* unit (CS2), ordering them in ascending order. From then on, two pairs of ordered values are compared in the four-input *compare and switch* (CS4) units, extracting a new pair of minimums. These units are then organized in a tree structure, obtaining at the root node the first and second absolute minimums. The results of the comparisons in both CS2 and CS4 units are combined through the tree and eventually used to select the proper variable node index.

An 8-input binary tree is shown as an example in Figure 4.2. Messages min1 and min2 computed from each node are passed to the parent node along with a sequence of bits $idx_0, idx_1, ..., idx_l$, where l is the height of the node. This sequence

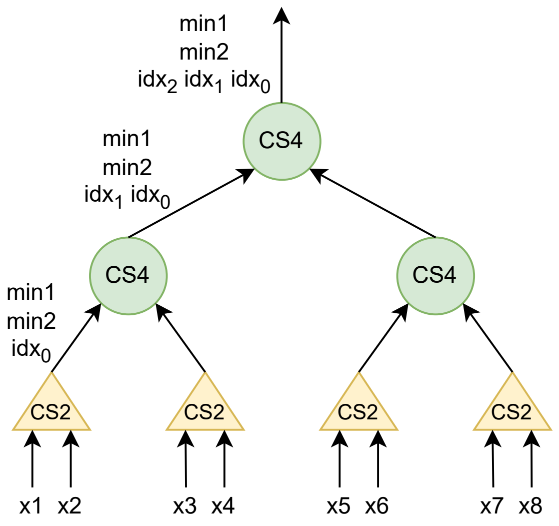


Figure 4.2: Example of a 8-input binary tree in full-row CNU architectures.

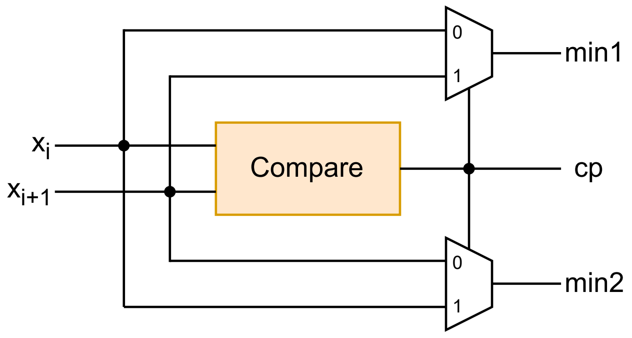


Figure 4.3: CS2 architecture.

is computed in each node, adding one bit in each level according to the provenance of the minimum between its child nodes. For example, the leftmost CS4 node subtending the input values from $x1$ through $x4$, is going to insert a bit $idx_1 = 0$ if min1 is from the leftmost CS2, which compares x1 and x2. Otherwise, min1 is one among x3 and x4, thus $idx_1 = 1$.

The CS2 unit is shown in Figure 4.3. It needs a simple *compare* unit, using the result to switch the positions of the two messages at output. The two minimums along with the comparator output are passed to the parent node, the CS4 unit.

The CS4 unit is shown in Figure 4.4, assuming that min1A, min2A and idxA are respectively the min1, min2 and index from child node A. Similarly, min1B, min2B and idxB are the output of child node B. This structure has to select the absolute minimum and second minimum among the four magnitudes at input. In order to

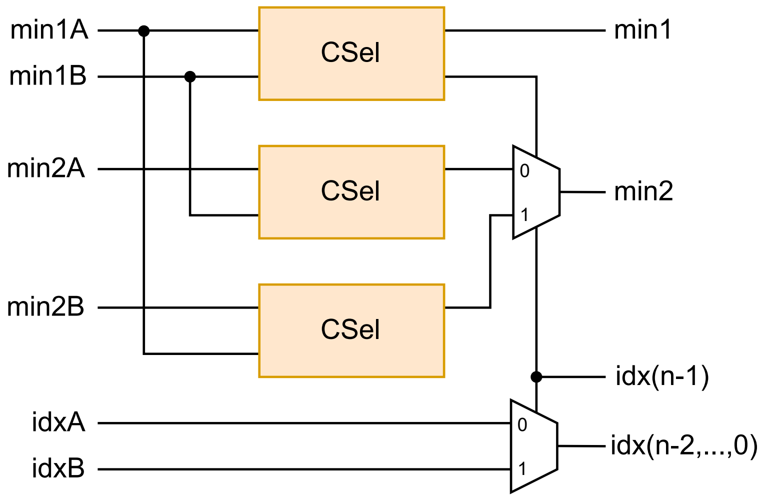


Figure 4.4: CS4 architecture.

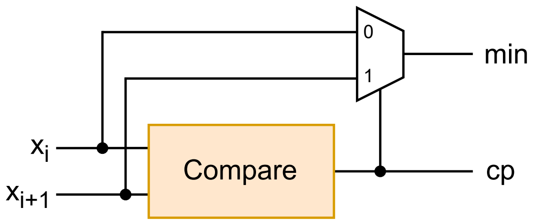


Figure 4.5: CSel architecture.

complete this operation, three comparisons are required: one between the two first minimums to find the absolute minimum, and two between the first minimum and the second minimum from different nodes. The result of the first comparison is used to select which of the other two is going to deliver the overall second minimum. These comparisons are similar to CS2, but only the actual minimum is needed at the output. For this reason, a *compare and select* (CSel) unit is employed. Lastly, the index is selected between idxA and idxB, concatenating the min1 comparison result. This hierarchy is reflected in the HDL design since it provides an easy and immediate view of the architecture.

Unfortunately, these units are sufficient only in perfect binary trees, but an arbitrary number of inputs may need some attention. In particular, 5G NR code has a maximum row degree of 19, thus a full-row CNU must have 19 inputs. An easier implementation can be achieved using a code generator, which exploits the programming language flexibility to compute the correct number and type of nodes of the tree. Moreover, the same generator can be used in half-row CNU design as explained in the following Section. A Python code generator was employed in this

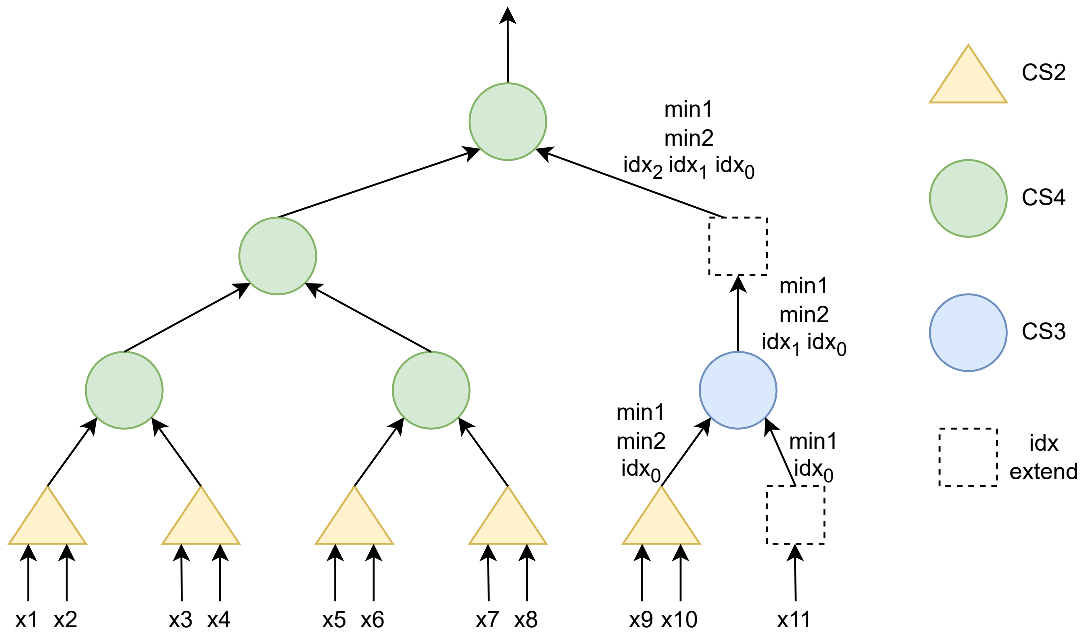


Figure 4.6: Example of an 11-input tree in full-row CNU architectures.

work.

In a generic tree, two more kind of nodes must be considered: CS3 and *index extend* nodes. CS3 nodes take as input a pair of min1 and min2 to be compared with a single input, thus it is a simplified version of CS4. The *index extend* is a node without comparisons, taking as input one single value or a pair of min1 and min2. In both cases, this kind of node only needs to extend the index value as if a comparison would have been made. In fact, even if there are no comparisons, the messages are passed to the next level of the tree, thus an index bit must be computed. These nodes do not translate into any unit since there is no logic other than the index assignment, which is fixed to '0' independently from the input values. Figure 4.6 shows an example of a binary tree with 11 inputs.

4.1.3 Half-row CNU

Reading only half of the variable-to-check messages, this CNU architecture must work in two cycles. In the first cycle it elaborates the first half of the messages and stores the temporary min1, min2 and idx1. In the second cycle, it reads the second half of the messages, find the minimums among those values and compares them with the ones of the previous cycle. In the proposed architecture, the half-row CNU uses the same tree structure seen in full-row CNUs with a number of inputs

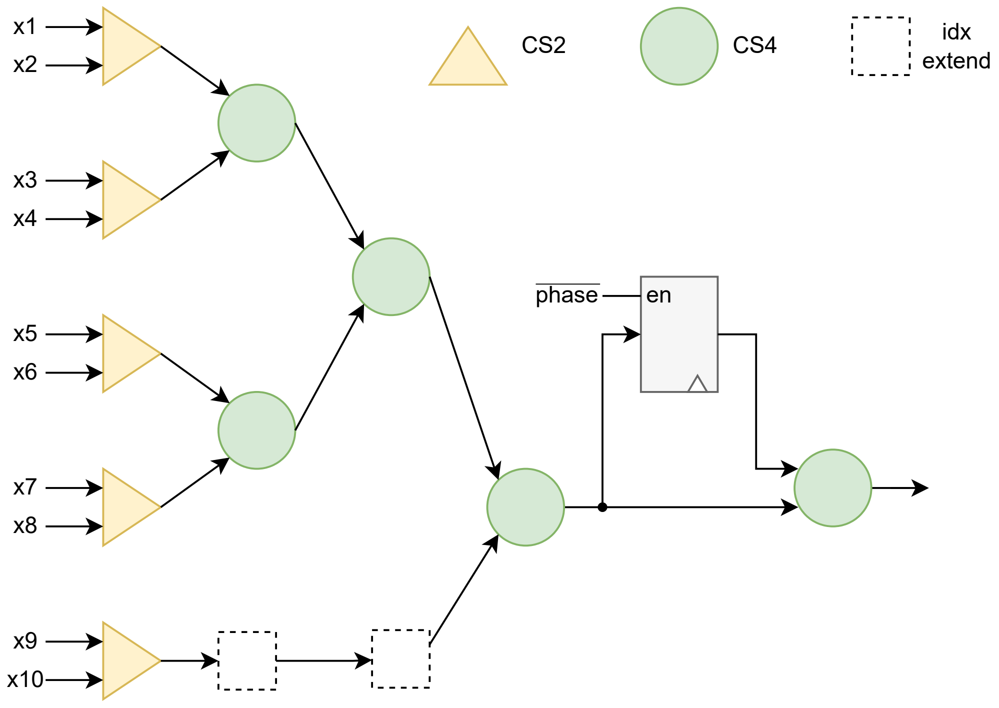


Figure 4.7: 10-input half-row CNU architecture

equal to $\lceil d_c^{max}/2 \rceil$, and uses a register to store the results to be compared as seen in the single-input CNU.

For this purpose, the code generator developed for the full-row architecture can be reused setting the proper number of input values. The resulting design can be then wrapped in a VHDL entity which includes the output registers and the final comparison. Since this comparison must find min1 and min2 among the four values, and update the final index selector by adding the last bit, this unit corresponds to a CS4. The final architecture for a 10-input half-row CNU is shown in Figure 4.7. This architecture is compliant with the 5G NR standard since the maximum row degree is $d_c^{max} = 19$.

4.2 VNU architectures

Variable nodes are processed according to the Min-sum algorithm presented in Chapter 3, Section 3.2.2. This computation is modified in the layered scheduling scheme as shown in the same Chapter, Section 3.4.3, thus a specific architecture is needed in that case.

In both cases, check-to-variable messages must be first retrieved from the compressed messages, thus *decompression* units are employed. Then, these messages

data	min1	min2	idx1	signs
bits	$w-1$	$w-1$	$\lceil \log d_c^{max} \rceil$	d_c^{max}

Table 4.1: Compressed β-message.

must be converted from the sign-magnitude representation to two's complement, which is more suitable for the sum operations to be performed [11]. In order to optimize the variable node processing, first the *a posteriori* LLR value is computed, and then the variable-to-check messages are retrieved subtracting the check-to-variable message to the LLR value.

VNUs also performs hard-decision based on the sign of the a posteriori LLR value, which can be used to determine whether a codeword is found and potentially early terminate the decoding.

4.2.1 Decompression unit

The check-to-variable β messages are stored in compressed form, meaning that all the β-messages related to a specific check node are compacted in one message. This form stores the magnitude of the absolute minimum min1, of the second minimum min2, the index of the first minimum and the signs of the actual β messages, as shown in Table 4.1. Storing all of the signs removes the necessity to store the signs of the α messages, which would be needed during the decompression.

According to the Equation (3.7), the β message magnitude is the minimum of the α messages magnitudes, excluded the one coming from the same variable node. In other words, the β message magnitude is equal to the min1 value, except when idx1 corresponds to the variable node the β message is addressed to. Only in that case, $|\beta|$ equals the min2 value. The implementation is straightforward with a multiplexer whose selection signal is the output of a *compare* unit, which compares idx1 with the variable node index.

4.2.2 VNU standard architecture

Variable node n is elaborated by this VNU architecture exploiting the variable-to-check message reformulation shown in Chapter 3, here reported for convenience:

$$\tilde{\gamma}_n = \gamma_n + \sum_{i \in S_c(n)} \beta_{i,n}$$
$$\alpha_{m,n} = \tilde{\gamma}_n - \beta_{m,n}.$$

This architecture takes as input the $\beta_{i,n}, i \in S_c(n)$, that is the set of β-messages addressed to variable node n. Firstly, the *a posteriori* LLR value $\tilde{\gamma}_n$ is computed

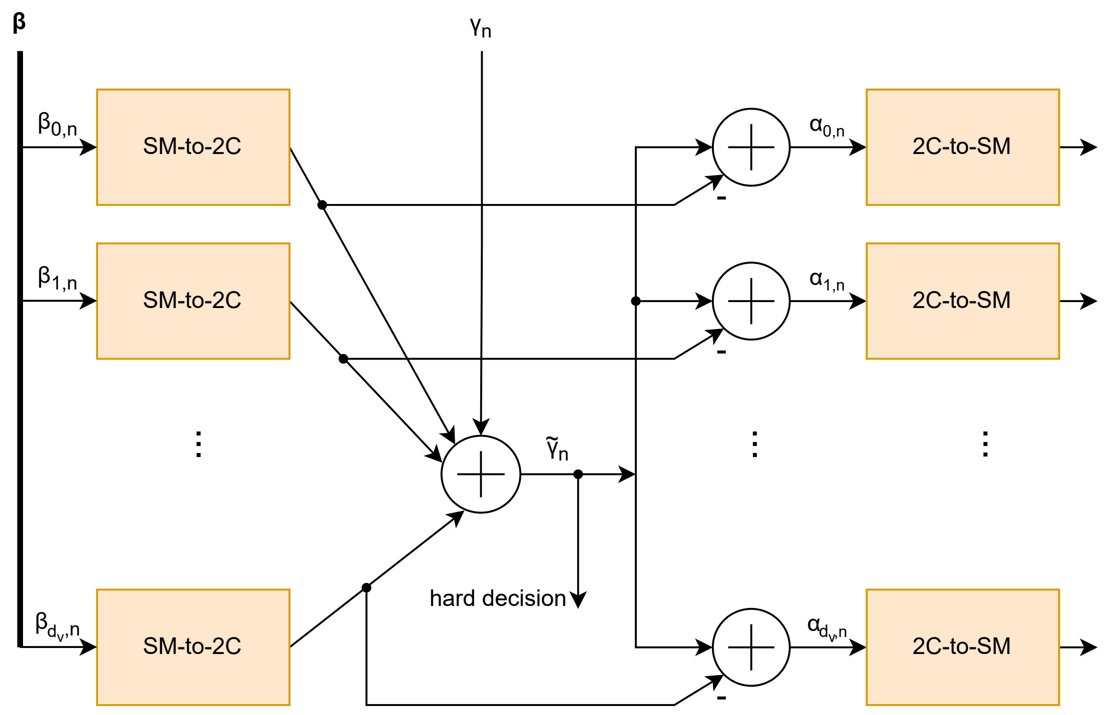


Figure 4.8: Standard VNU architecture. This is a full-column architecture as it reads all the β-messages and produces all the α-messages related to the processed variable node.

by summing all the β-messages along with the LLR channel information γ_n. Then, each $\beta_{i,n}$ message is subtracted to the result in order to compute each $\alpha_{i,n}$. Also, the sign of $\tilde{\gamma}_n$ is used as hard decision.

The described architecture is shown in Figure 4.8. The α-messages computation is more efficient in two's complement [11], while the β-messages are easier to compute in sign-magnitude representation, thus a *sign-magnitude to two's complement* (SM-to-2C) conversion is employed. The computed α-messages are converted back with a *two's complement to sign-magnitude* (2C-to-SM) conversion.

This work focuses only on Min-sum algorithm, but Normalized Min-sum and Offset Min-sum can be implemented by applying the proper modification at the end of the α computation.

4.2.3 VNU architecture for layered scheme

Row-layered scheduling is described in Chapter 3, Section 3.4.3, where it is shown how the variable-to-check message computation can be modified to take into account

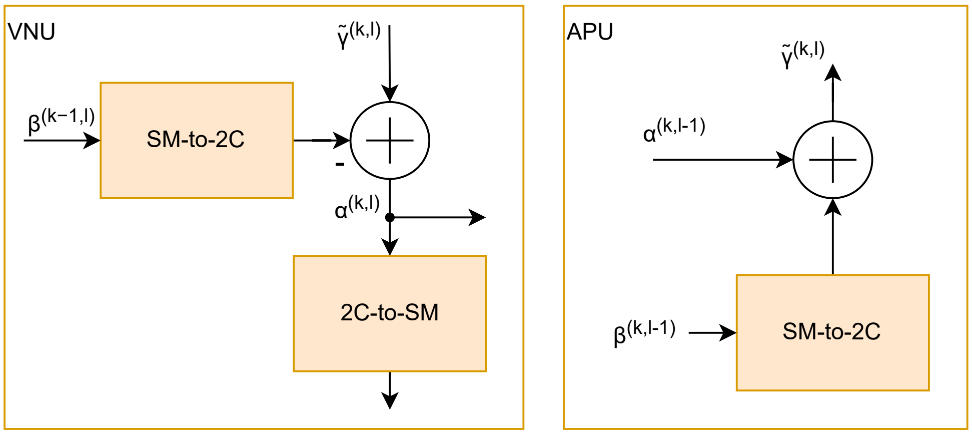


Figure 4.9: VNU architecture in the layered scheme. This architecture includes the APU block and the conversion units. The VNU output goes to the CNUs and the FIFO, which provide the input to the APU. The APU updates the VNU input in the next layer.

the β-messages computed in each layer, in order to use the updated information in the next layer. This entails to separate the α-message computation from the $\tilde{\gamma}_n$ update. The general architecture in Figure 3.9 uses two blocks: one VNU and one A posteriori probability update unit (APU). The VNU block takes care of the α-message computation as expressed in Equation (3.9). The APU block uses the update mechanism detailed in the layered scheduling scheme section and expressed in Equation (3.8). These two equations are reported here for convenience:

$$\begin{aligned}\tilde{\gamma}^{(k,l)} &= \alpha^{(k,l-1)} + \beta^{(k,l-1)},\\ \alpha^{(k,l)} &= \tilde{\gamma}^{(k,l)} - \beta^{(k-1,l)}.\end{aligned}$$

Therefore, the VNU block consists in a subtractor, while the APU block is an adder. Similarly to the VNU standard architecture, these computations are easier in two's complement, thus SM-to-2C and 2C-to-SM conversions are necessary.

Figure 4.9 shows both the VNU and APU. On the left, the VNU reads the β-message from the β-memory and computes the α-message. This is output both in sign-magnitude and in two's complement representation since the first goes to the CNU and the latter is stored in the FIFO, which holds it until it is consumed in the APU update operation. On the right, the APU is shown, which reads the α-message held in the FIFO and the β-message computed in the CNU.

4.3 Routing networks

One of the most complex components in partially-parallel decoders is the routing network. This is necessary to route correctly the messages between variable and check nodes. In fact, in general there are two reciprocal routing networks. If the parallelism of either CNUs or VNUs is *fully-parallel* (that is to say that there are as many of this processing element as the nodes) and all their input messages are read simultaneously, the corresponding routing network can be removed in favour of hardwired connections.

An important simplification is given by the employment of Quasi-cyclic LDPC codes, presented in Chapter 2, Section 2.3. In fact, these codes let us gather both variable-to-check α messages and check-to-variable β messages in groups of Z messages, which must switch their position with the same degree as the circular shift of the identity matrix in the corresponding sub-matrix. In other words, the messages must circularly shift n times, where n is the coefficient from the base-matrix.

The implementation of such routing network exploits some barrel shifters, which produce exactly this behaviour. The number of barrel shifters depends on the decoder architecture. For example, row-layered decoders with full-row CNU need d_c^{max} barrel shifters, where d_c^{max} is the maximum row degree of the parity check matrix. This number can be obtained according to the scheduling scheme, the number of units and their input parallelism, as discussed in Section 4.5.2.

4.4 Memory organization

Memories and routing networks can easily occupy the majority of the decoder area. For this reason, memory requirements are a crucial part of the decoders analysis. According to the scheduling and code parameters, memory types, requirements and access policy can be defined. As discussed in Chapter 3, the scheduling schemes may require different memories, which can be summarized as:

α-memory This memory records the α messages while they wait to be consumed from a CNU.

β-memory Symmetrically to the α-memory, it stores the β compressed messages produced by the CNUs and read by the VNUs. For simplicity, the compressed message length will be indicated by $w\_cmp = 2(w-1) + \lceil \log(d_c^{max}) \rceil + d_c^{max}$.

γ-memory Stores the LLR channel information for the whole duration of the codeword decoding if the γ values are needed in the VNU computations.

$\tilde{\gamma}$-memory Replacing the γ-memory in layered architectures, it stores the updated *a posteriori* LLR information $\tilde{\gamma}$.

FIFO This special memory element is exclusively employed in layered architectures in order to hold the α-messages during the layer elaboration, and until this information is consumed in the *a posteriori* LLR information update.

The combination of scheduling scheme and processing elements' architecture determines the memory access policy and the memory requirements. In turns, this also indicates the most appropriate memory type.

As discussed in Section 4.5.2, In the presented work, each scheduling scheme has a specific node parallelism, that it is to say the number of CNU or VNU against the number of edges per column or row, respectively. In particular, the flooding and sliced message-passing schemes have $M = M_p \cdot Z$ CNUs, and Z VNU. In other words, all the rows are processed in parallel, while one block column corresponding to one sub-matrix is elaborated per cycle. Layered architectures have Z CNUs, while the number of VNUs depends on the input parallelism of the CNU, as needed in this scheduling scheme. All LLR values are represented with w bits.

4.4.1 Memory requirements

Flooding scheme adopts the γ-memory to store the channel information, reading Z values per cycle during the VNU elaboration. Thus this memory must be able to store $N = N_p \cdot Z$ words, each of w bits, for a total of $N_p \cdot Z \cdot w$ bits. The β-memory must store $M = M_p \cdot Z$ compressed messages, each of length w_cmp. The last memory employed in flooding scheme architectures is the α-memory, which must store up to d_v^{max} messages per variable node, for a total of $d_v^{max} \cdot Z \cdot N_p$ messages with w bits.

Sliced message-passing scheme requires the same γ-memory as in flooding scheme architectures, but it doubles the β-memory size in order to be able to store the check-to-variable messages for iteration $i + 1$ at the same time as to read the β-messages of iteration i. Nevertheless, the α-memory can be removed if the α-messages are consumed immediately when they are computed. For this reason, the number of VNUs employed depends on the CNU's input parallelism and the parity-check matrix organization. *Decompression* units may still need the α-messages' signs if the check-to-variable compressed messages do not store the β-messages signs, but only the total sign product. This is not needed if the compressed messages explicit all the β signs.

Row-layered scheme architectures replace the γ-memory with the $\tilde{\gamma}$-memory, which is initialized with the LLR channel information and then stores the updated *a posteriori* LLR probabilities $\tilde{\gamma}$. The difference between γ- and $\tilde{\gamma}$-memory is

Scheduling	Flooding	Sliced MP	Layered
γ-/$\tilde{\gamma}$-memory	$N \cdot w$	$N \cdot w$	$N \cdot w$
β-memory	$M \cdot w\_cmp$	$2 \cdot M \cdot w\_cmp$	$M \cdot w\_cmp$
α-memory	$d_v^{max} \cdot N \cdot w$	-	-
FIFO buffer	-	-	$Z \cdot \text{CNU\_lat} \cdot \#\text{CNU\_in} \cdot w$

Table 4.2: Memory requirements.

merely functional, as the memory requirements and access policies are identical. The β-memory has the same storage requirements as previous scheduling schemes. Finally, the α-memory is removed as the α-messages are computed and immediately consumed by the CNU. On its place, a FIFO buffer is employed to store these messages until they are needed in the APU for the $\tilde{\gamma}$ update operation. This memory needs to store as many α-messages as they are computed in one processed layer, and for as many cycles as the CNU latency.

The memory requirements are summarized in Table 4.2.

4.4.2 Memory access policy

According to the scheduling schemes and the CNU input parallelism, memory access policies can vary, entailing different memory types. In this work, the input parallelism of VNUs is not a free parameter, otherwise it should be considered as well.

In **flooding** scheme, the γ values are read in two occasions: in the first check node iteration, and in all variable node elaborations. Since the chosen decoder architectures have Z VNUs, they always read a group of Z consecutive w-bit values, corresponding to one sub-matrix in the QC-LDPC code. Nevertheless, the CNU input parallelism may require to read more γ-values in the same cycle. In particular, single-input CNUs read one group of Z LLR values per cycle; full-row CNUs read all N γ channel information; half-row CNUs at least $N/2$ γ, but this number can increase according to the positions of ones in the parity-check matrix.

Since M CNUs work in parallel, the β-memory must be able to write all M messages at the same time at the end of the check nodes processing, independently on the input parallelism. VNUs read in each cycle up to d_v^{max} random groups of Z consecutive compressed messages, according to the position of the non-zero elements in the corresponding block column of the parity-check matrix (or base matrix).

After the first iteration, CNUs receive their input values from the α-memory, hence it must be able to be read similarly to the γ-memory, i.e. according to the

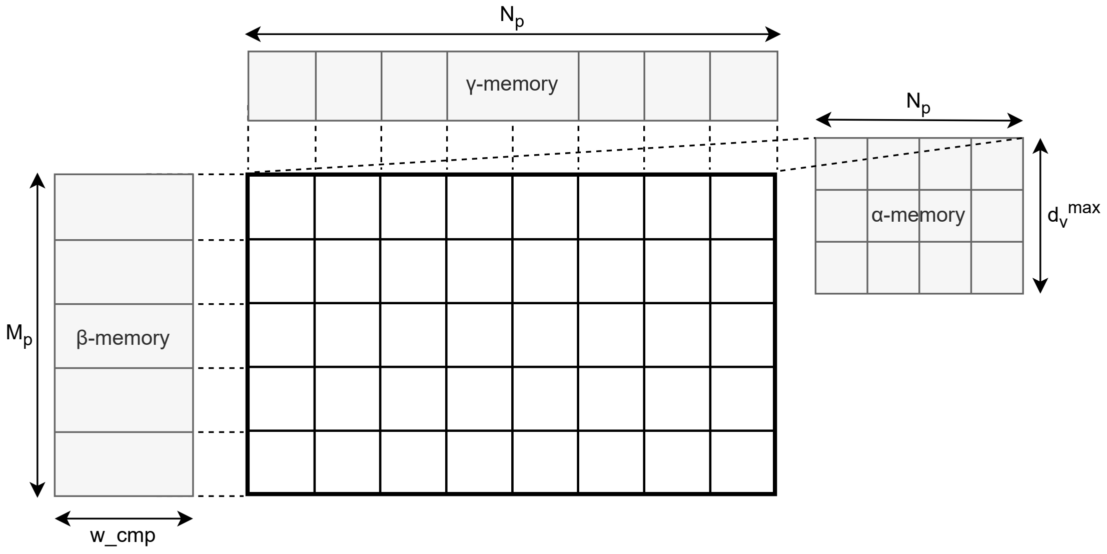


Figure 4.10: Relation of the $\gamma/\tilde{\gamma}$-memory, α-memory and β-memory to the parity check matrix.

CNUs' architecture, but also extending the word parallelism to the d_v[1] messages. In fact, single-input CNUs need to read up to $Z \cdot d_v^{max}$ messages per cycle; full-row CNUs reads all the α-messages; and half-row CNUs need to read about half the α-messages. The write operation parallelism is given by the Z VNUs, each writing up to d_v^{max} messages.

In **sliced message-passing**, the γ-memory access is the same as in the flooding scheme. As previously, M of the β messages must be written all in parallel by the CNUs, while at the same time up to d_v^{max} random groups of Z messages are read from the VNUs.

Layered scheme employs the $\tilde{\gamma}$-memory which must be updated with each layer elaboration, thus reading and writing d_c^{max} random groups of Z LLR values, according to the non-zero elements in the corresponding row of the parity-check matrix (or the base matrix). On the other hand, the β-memory only reads and writes Z consecutive compressed messages corresponding to the processed layer, updating the β messages for the next iteration. Lastly, the FIFO buffer must be able to read and write all produced α messages in each cycle. The CNU input parallelism determines the number of VNUs, thus the number of α messages produced in one cycle, the number of cycles needed to process one layer and at last, the size of this buffer. Full-row CNUs may avoid to use the buffer at all being a combinational circuit.

[1]d_v denotes the average row degree.

Scheduling	CNU architecture	Memory
Flooding		
γ-memory	single-input	SRAM: $N_p \times Z \cdot w$ bits
	full-row	N_p regs of $Z \cdot w$ bits
	half-row	N_p regs of $Z \cdot w$ bits
β-memory	all	M_p regs of $Z \cdot w\_cmp$ bits
α-memory	single-input	SRAM: $N_p \times Z \cdot d_v^{max} \cdot w$ bits
	full-row	$N_p \cdot d_v$ regs of $Z \cdot w$ bits
	half-row	$N_p \cdot d_v$ regs of $Z \cdot w$ bits
Sliced MP		
γ-memory	single-input	SRAM: $N_p \times Z \cdot w$ bits
	half-row	N_p regs of $Z \cdot w$ bits
β-memory	all	$2 \cdot M_p$ regs of $Z \cdot w\_cmp$ bits
Row-layered		
$\tilde{\gamma}$-memory	all	N_p regs of $Z \cdot w$ bits
β-memory	all	SRAM: $M_p \times Z \cdot w\_cmp$ bits
FIFO buffer	single-input	Z buffers of $d_c^{max} \times w$ bits
	full-row	-
	half-row	Z buffers of w bits

Table 4.3: Memory types.

4.4.3 Memory choice and results

Summarizing, QC-LDPC codes allow to group Z consecutive values to be read from and written to all memories, while routing networks will take care of exchanging their position within the same group. The actual memory type employed heavily depends on the required access policy. SRAM would be more preferable for the superior area efficiency, but the small number of ports considerably limits the parallel accesses. Thus, usually dual-port SRAM memories are used whenever possible, in order to ensure write and read operations within the same cycle. However, if random access of several groups of Z values are needed, registers become unavoidable. The chosen memory types are summarized in Table 4.3.

4.4.4 Memory conflicts in layered architectures

Architectures with the layered scheduling scheme can incur into memory conflicts because of the presence of a pipeline between the variable or check nodes elaboration and the *a posteriori* LLR update. In fact, if two successive layers have nonzero overlapping columns, the APU elaborating layer i is updating the $\tilde{\gamma}$ values that should be used during the same cycle by the VNUs and CNUs to elaborate layer $i+1$. This problem is described in [19] and [20]. The baseline solution for the conflict is the insertion of idle cycles, which decrease the decoder performance. These papers analyze the problem, proposing a mitigation of the performance loss by rescheduling the matrix operations, rearranging the column and the rows of the matrix. Both propose to model the optimal scheduling research as a *Traveling Salesman Problem* and solve it with a genetic algorithm, simulated annealing algorithm or similar. This approach can reduce the idle cycles without overhead in the circuit complexity.

Furthermore, the same effects occur during the conventional decoding of a layered architecture whose CNUs require multiple cycles to perform their operations. In fact, the *a posteriori* update operation must wait for the CNUs' to have terminated their processing, since the valid β is obtained only after the last cycle. Again, during the APU processing, the VNUs and the CNUs should be in idle waiting for the update process to compute, which takes as many cycles as the CNU to complete. This problem is tackled in [21], which, in addition to the matrix reordering previously implemented, also propose to modify the *a posteriori* update scheduling to be performed right before VNUs processing. The presented decoder architecture moves away from the conventional solutions and is compliant with the 5G NR standard.

The authors of [22] propose a decoder architecture employing both layered and flooding scheduling schemes, in order to switch when a memory conflict would occur. Moreover, also in this case a genetic algorithm is used to find an optimal matrix rescheduling.

4.5 Decoder architectures in the design space

The design space of LDPC decoders can be extremely wide. For this reason, the proposed tool's scope is limited by a few design choices. For each scheduling scheme a main architecture topology is defined, fixating the number of CNUs and VNUs, routing networks and, as discussed, memory units. Loop unrolling is considered for each architecture, with unrolling degree equal to the maximum number of iterations. This design choice enables the possibility to implement pipelining also in flooding and sliced message-passing schemes, which otherwise would not increase throughput due to the iteration loop.

4.5.1 Quantization

The channel information, the *a posteriori* LLR probabilities, the α and the β messages are all soft-information represented in fixed point. The number of bits used for this representation is the *quantization* and it is denoted with w. A large number of bits would increase the performance, but it also increases the hardware complexity and memory requirements, while reducing the maximum achievable frequency, as stated in [11]. Here, it is claimed that good error-correcting capabilities can be achieved in Min-sum decoders with $w = 5$. For this reason, the present work sets the word length to this value.

4.5.2 Architectures' topology

The scheduling scheme is used to determine the general architecture's structure, shown in Chapter 3, Section 3.4. It entails the number of processing elements employed, routing networks and memories. A few details may depend on whether or not loop unrolling is applied. The memory organization was discussed in Section 4.4, thus it will not be repeated in this section.

Flooding scheme The general architecture for this scheduling scheme entails M CNUs, Z VNUs. With this organization, check nodes are processed in a *fully-parallel* way, while variable nodes are processed one block column per cycle. VNUs are implemented with the standard architecture, thus one clock cycle is sufficient to elaborate the whole block column. To complete the variable node processing, N_p cycles are needed. CNUs can be implemented with any architecture in the considered design space. If single-input CNUs are employed, the α-messages corresponding to one block column are read per cycle, thus a total of N_p cycles are needed to complete the check node processing for one iteration. Full-row CNUs process the whole matrix in one cycle, thus one cycle is sufficient. Half-row CNUs read half of the α-messages, thus a total of two cycles are needed.

The routing network that drives the inputs of the CNUs is implemented with a number of barrel shifters that depends on the CNUs' architecture. It is equal to d_v^{max} for single-input CNUs (since it is the maximum number of α-messages read per cycle) and $\lceil tot\_edges/(2\,Z) \rceil$ in half-row CNUs. Full-row CNUs do not need routing networks because of the *fully-parallel* processing of check nodes. Hardwired interconnections must be placed instead. On the other hand, the *reverse* routing network is always present with a fixated number of barrel shifters in the considered design space, which is equal to the number of VNUs times the number of their input divided by Z, thus they are equal to d_v^{max}.

Sliced message-passing scheme Similarly to the previous case, architectures with single-input CNUs have M CNUs and Z VNUs. Considerations on the VNUs' latency are identical, while the CNUs only bring one extra cycle to complete their elaboration. CNUs cannot be implemented in full-row since this scheduling scheme entails to compute only a portion of the input α-messages per cycle. Moreover, in order to remove the α-memory in decoders with half-row CNUs, VNUs should be able to compute all the α-messages addressed to the CNUs in one cycle, for an overall of two cycles. This implementation is not feasible in most parity-check matrices, thus it is excluded from the design space. If a matrix rearrangement is considered, then $\lceil N/2 \rceil$ VNUs could be employed to elaborate the whole parity-check matrix in two cycles, computing half of the total α-messages per cycle.

The same considerations on routing networks for flooding schemes can be repeated in this case.

Row-layered scheme Layered architectures typically elaborate one layer per sub-iteration, corresponding to one row of sub-matrices in the QC-LDPC code, that is Z consecutive rows of the parity-check matrix. For this reason, in this work Z CNUs are employed to elaborate the same number of check nodes, while the number of VNUs depends on the CNUs' input parallelism. In fact, according to the row-layered scheduling scheme, the VNUs are implemented with the corresponding architecture, to compute the needed α-messages right before the check nodes processing. Therefore, the number of VNUs is given by the number of CNUs multiplied by the number of their input messages. Moreover, there are as many APUs as VNUs, updating the corresponding *a posteriori* LLR probabilities. Lastly, also the routing networks must be capable of driving the same number of messages. Since there are Z CNUs elaborating one layer and each barrel shifter drives one input of each CNU, the number of barrel shifters employed corresponds to the CNUs' input parallelism.

In this case, since all the check nodes are not (and cannot be) elaborated in parallel, the routing networks are always needed, regardless of the number of inputs of the CNUs. That is because with every layer, the input messages for each CNU change also in full-row architectures. On the other hand, the reverse routing network can be removed using the offset permutation scheme proposed in [23], which can be computed offline. With this solution, the critical path can be reduced, reaching higher frequency and better performances.

4.5.3 Loop unrolling and pipelining techniques

The loop unrolling technique involves the allocation of multiple instances of the same sub-decoder, formed by the processing elements that are needed in the

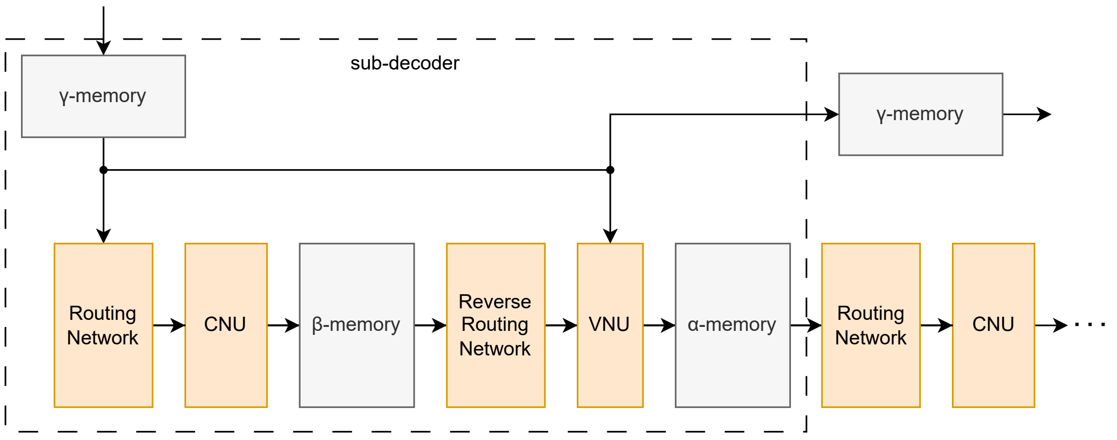


Figure 4.11: Unrolled decoder architecture with flooding scheme. Each sub-decoder has the same number of components as the basic architecture.

elaboration of one decoding iteration. These include CNUs, VNUs and memories. The LDPC architectures considered in the DSE tool are unrolled against the main iteration loop with unrolling coefficient equal to the number of maximum decoding iterations, or *fully-unrolled*. With this configuration, the information on the codeword passes through the sub-decoders in a pipeline-fashion, letting each sub-decoder to continuously elaborate a new set of data. Because of this, the CNUs' and VNUs' operations can be overlapped in any scheduling schemes, as they are working on different data. The result is that the number of cycles needed to complete one decoding iteration can be improved. In Chapter 5, Section 5.3.3 it will be discussed how the number of cycles can be redefined in view of the throughput estimation.

It is worth noting that the layered scheduling scheme uses the information from previous iterations progressively during the layers elaboration. Since the unrolling is only performed against the main decoding iteration, the sub-iteration (i.e., the sequence of layers) is maintained. This entails that with each iteration, the β-memories should be shared and routed accordingly. This is more efficient if multiple, separated layered decoders process in parallel different codewords, rather than resort to loop unrolling.

In decoder architectures without loop unrolling, the presence of the feedback loop prevents the advantages of applying the pipeline technique. In particular, the frequency can be doubled, but so does the latency, without the possibility to process multiple data in different stages of the pipeline[2]. This is especially true for the architectures in the considered design space. Layered decoders are an exception,

[2]interleaving is not considered in this work

since successive layers can be processed before the last one has been completed. This can lead to hazards and performance loss, as discussed in Section 4.4.4.

On the other hand, fully-unrolled architectures have no feedback loop, thus the decoding operations can be pipelined, having different codeword being decoded in successive pipeline stages.

Chapter 5

A design space exploration tool

The proposed design space exploration (DSE) tool aims at providing a fast and low-investment overview of the LDPC decoder solutions to a hypothetical designer, who is going to further investigate the solutions most appealing for their use case. This goal is tackled by building an algorithmic-architectural model of several decoder architectures, exploring the many combinations of the free parameters determined in the scope. For each architecture, area, working frequency and throughput are estimated exploiting either the direct results of building blocks' syntheses, or a mathematical model that can be obtained empirically from simpler syntheses.

The check node units (CNU) and variable node units (VNU) are synthesized with the exact architecture needed from the tool, retrieving the area and throughput of these main components. Routing networks are too large to be synthesized, therefore they are estimated exploiting an empirical model obtained from smaller barrel shifter syntheses, in particular against the lifting size parameter. In this way, it was possible to find a function of the area and delay of the barrel shifters given any lifting size. Similarly, the same procedure was used for memories implemented with registers, while SRAM models were extracted from the library and a mathematical model was retrieved from those.

The DSE tool was implemented in **Python**, while the components designs were described in **VHDL**, tested and synthesized employing a 45 nm technology library.

In this chapter, firstly the considered design space is discussed and the boundaries are motivated. Then, the building blocks' models are discussed. The actual DSE tool architecture is presented, detailing the proposed approach. Lastly, the model is validated comparing its results with the ones found in published decoder architectures, applying a normalization only when strictly necessary.

Parameter	Fixed value
Parallelism	Partially-parallel
Decoding algorithm	Min-sum
LDPC code	5G NR, BG 1
Technology library	45 nm

Table 5.1: Fixed parameters in the design space exploration.

5.1 Design space's scope

The design space of LDPC decoders is extremely wide. For this reason, the proposed tool's scope restricts a few parameters to the most prevalent values, summarized in Table 5.1. Partially-parallel LDPC decoders are the most popular solutions thanks to their excellent low-complexity and high-throughput trade-offs. Similarly, Min-sum decoding algorithm is widely used. Quasi-cyclic LDPC codes offer great performance, while allowing several simplification in the architecture compared to randomly constructed codes. In particular, the chosen QC-LDPC code is compliant with the 5G NR standard, using base-graph 1 (BG1). From the technological point of view, a 45 nm node technology library is selected in the synthesis of the building blocks.

On the other hand, the free parameters can be chosen among specific options, as shown in the following list and summarised in Table 5.2. These values are often not numerical, but descriptive of a design choice.

Scheduling scheme Flooding, sliced message-passing and layered schemes are considered. They are presented in Chapter 3, Section 3.4.

CNU architecture Check node units (CNUs) can be implemented with single-input, full-row or half-row architectures. These designs are described in Chapter 4, Section 4.1.

VNU architecture According to the scheduling scheme, variable node units (VNUs) are implemented with the standard architecture or with the layered version, both detailed in Chapter 4, Section 4.2.

Loop unrolling and pipelining All the architecture are considered with or without loop unrolling against all the decoding iterations. Pipelining is considered only in unrolled architectures or in layered schemes. These techniques are presented in Chapter 4, Section 4.5.3.

Numerical parameters The architectures are studied against some numerical parameter, specifically the number of iterations I and the lifting-size Z.

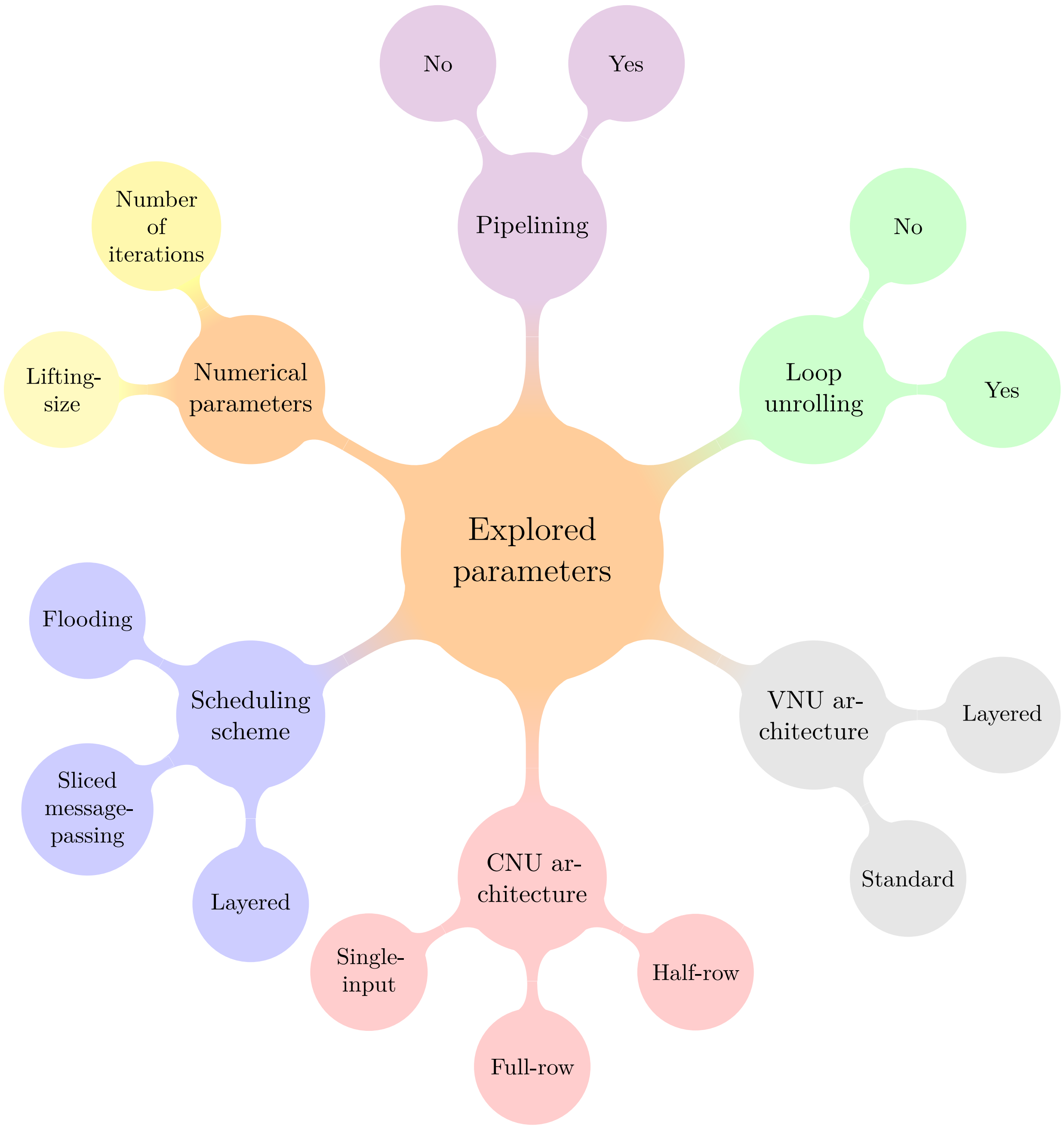


Figure 5.1: Free explored parameters tree. Some parameters can take on qualitative values (scheduling schemes, CNU architectures, etc.), while others can take numerical values (Number of iterations, lifting-size)

Parameter	Possible Values
Scheduling scheme	Flooding, sliced message-passing, layered
CNU architecture	single-input, full-row, half-row
VNU architecture	standard, layered
Loop unrolling	yes, no
Pipelining	yes, no
Numerical parameters	lifting-size, number of decoding iterations

Table 5.2: Free parameters in the design space exploration.

5.2 Architectures modeling

The idea at the heart of the proposed tool is to exploit the characteristics of area and delay of the building blocks that form the decoder in order to make assumptions on the overall architecture. These characteristics can be obtained in two ways: either the synthesis of the exact component architecture is performed, or a mathematical model is exploited. For example, several, smaller architectures of the component can be synthesized, retrieving an empirical model to predict the bigger architecture's characteristics. The first method is used for all the CNU and VNU architectures, as the exact syntheses are not too time-consuming. Moreover, they do not depend on the explored parameters. The same cannot be said in general for the routing networks or the memories, which represent the biggest part of the decoder area.

In order to reduce the computational time in the routing network's synthesis, the overall architecture is approximated by the barrel shifters that form it. In other words, the delay of the routing network is assumed to the delay of one barrel shifter, while the area is assumed to be the sum of the barrel shifters' area. With this approach, a singular barrel shifter's synthesis is sufficient to characterize the complete routing network.

Even with this simplification, syntheses of barrel shifters with the maximum lifting size Z provided in the 5G NR standard are extremely long. This is also true for the block memories. Additionally, these two components are the only ones influenced by the lifting size parameter, while the CNUs and VNUs only change in number of instances. This led to the possibility to extend the design space exploration to any desired value of the lifting size Z, applying a mathematical model.

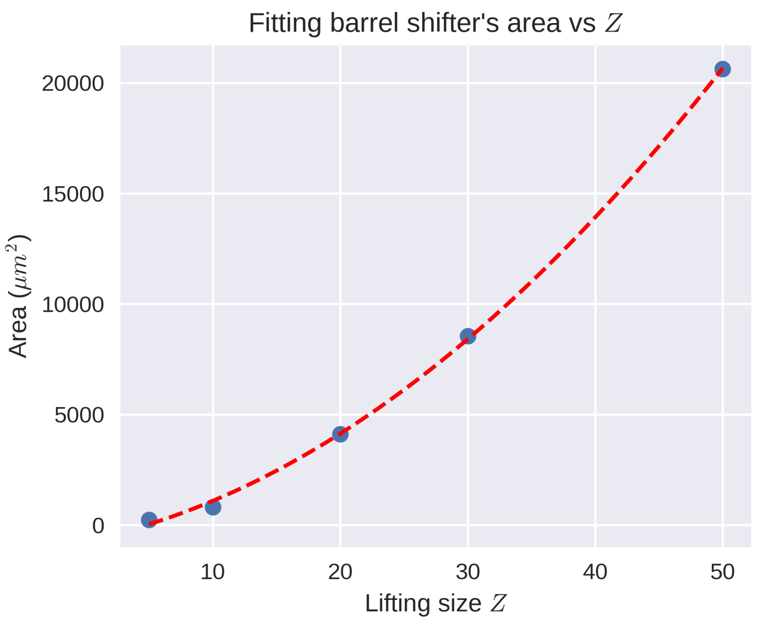


Figure 5.2: Barrel shifter's area against different lifting sizes Z.

5.2.1 Barrel shifter's empirical model

As previously mentioned, the barrel shifter synthesis with large lifting sizes Z is a very time-consuming task for the synthesis tool. In fact, this parameter defines the number of input and output signals in the component, which must be able to perform circular shifting among them. The complexity of the barrel shifter related to the lifting size is expected to be quadratic in the area, and logarithmic in the delay. Therefore, a few syntheses with smaller lifting sizes can be used to retrieve a few points in the area and delay characteristics, which once fitted can provide a mathematical model to estimate these values with different lifting size Z.

The area data collected during the syntheses are shown in Figure 5.2. Here it is possible to notice the quadratic behaviour of the area with respect to the lifting size Z. Noticing that, one additional synthesis with the maximum lifting size in 5G NR standard $Z = 384$ was completed in order to have a more reliable model.

The same methodology is employed for the delay estimation, where the fitting curve was less accurate. Nonetheless, the results obtained and shown in Figure 5.3 were fitted with an expression in the form $a \ln(x) + c$. These empirical models can be applied during the tool's analysis to estimate the area and delay characteristics of a barrel shifter for any used Z, and in turns of the whole routing network.

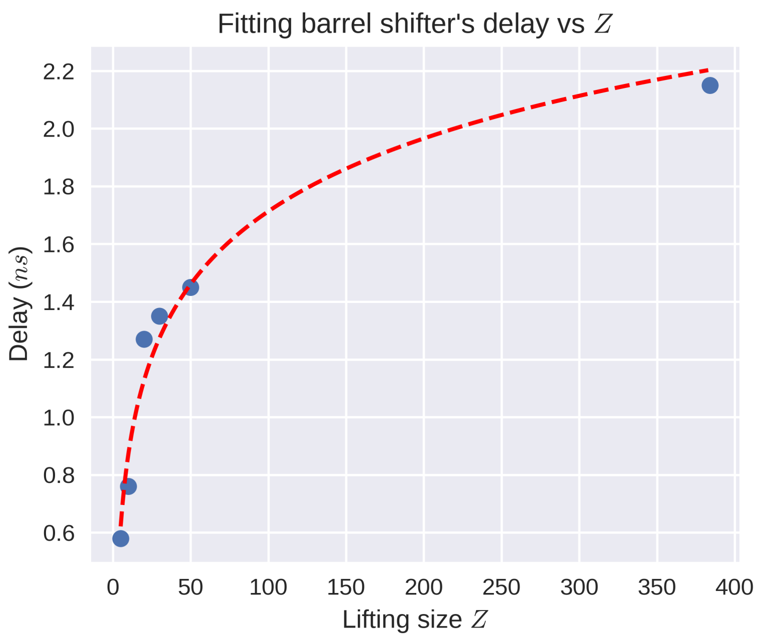


Figure 5.3: Barrel shifter's delay against different lifting sizes Z.

5.2.2 Memories' mathematical model

In LDPC decoders, two main types of memories are employed: registers and SRAMs[1], as discussed in Chapter 4, Section 4.4. By changing the lifting size Z, only the word length of memory locations or registers is extended proportionally.

Two models are provided for two register sizes, $Z \cdot w$ bits and $Z \cdot w\_cmp$ bits, where w_cmp is the length of the compressed check-to-variable message. These could be compacted in a unique model, but they are kept distinct in the tool. The syntheses results are shown in Figure 5.4 for both types of registers, where the fitting curve of the data led to two linear equations.

The memory analysis showed the necessity to model two sizes of SRAMs: one with N_p words and one with M_p words. In other words, the DSE tool needs an SRAM model that takes into account both the number of words and the word size, denoted with *words* and *bits*, respectively. SRAM blocks' dependency on the number of words is not linear because of the peripheral circuitry. Assuming the memory area model can be expressed as

$$A_{tot} = A_{cell} \cdot words \cdot bits + overhead_{tot},$$

[1]Since FIFOs change with the lifting size Z in the number of units employed and not their sizes, they are excluded from this analysis. Moreover, they are rapidly synthesized with the exact requirements, thus not needing a mathematical model.

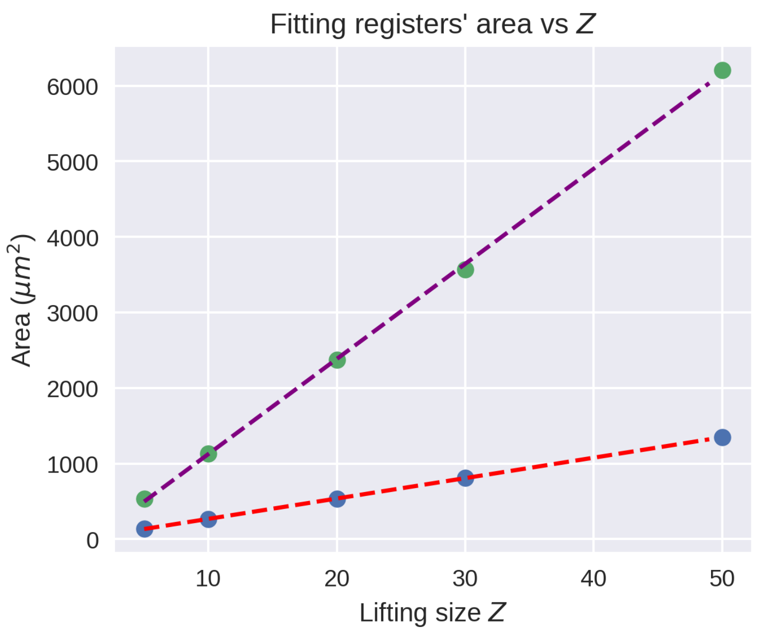


Figure 5.4: $Z \cdot w$ bits (red) and $Z \cdot w\_cmp$ bits (purple) registers' area against different lifting sizes Z.

the area density can be defined as

$$A_{tot/bit} := \frac{A_{tot}}{words \cdot bits} = A_{cell} + overhead_{\mu m/bit}. \tag{5.1}$$

Using the double-port SRAM models in the 65 nm library and the corresponding SRAM cell area A_{cell}, the peripheral circuits' overhead can be computed with:

$$overhead_{\mu m/bit} = A_{tot/bit} - A_{cell}$$

obtaining the values collected in Figure 5.5. These values are fitted against the number of words with the expression

$$overhead_{\mu m/bit} = \frac{1}{a \ln(b + c \cdot words)}.$$

Finally, the memory area can be computed combining the overhead model with the area density definition in Equation (5.1):

$$A_{tot/bit} = A_{cell} + \frac{1}{a \ln(b + c \cdot words)}$$
$$A_{tot} = A_{tot/bit} \cdot words \cdot bits.$$

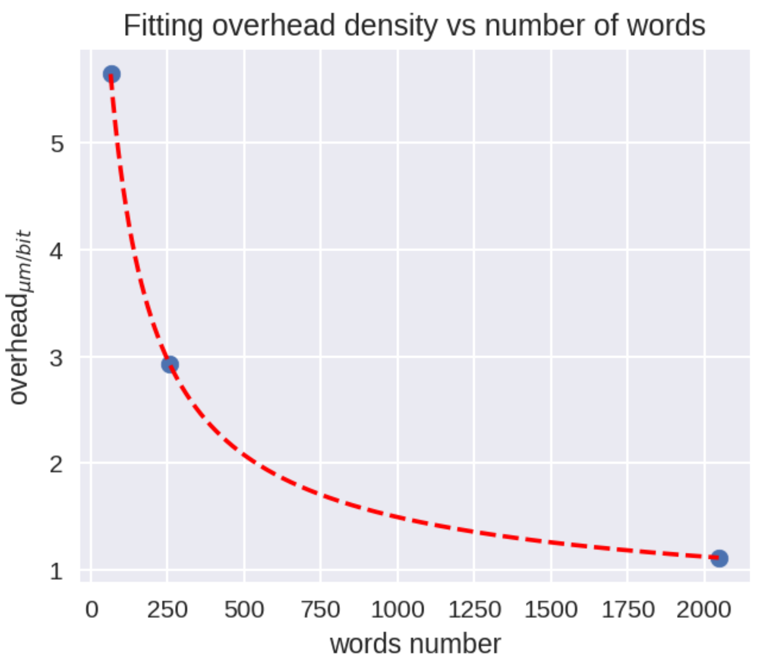


Figure 5.5: SRAM peripheral circuits' $overhead_{\mu m/bit}$ fitting against different word sizes $words$.

In order to apply this model to the 45 nm technology, a scaling is necessary. The factor $(45/65)^2$ is typically suggested in device scaling [24], [25], but a correction is proposed in [26] using the more accurate factor 1.5.

The obtained model can be employed to take into account both the number of words and the word length, thus it can be used to study the memory characteristics against Z. The number of words is selected according to the memory modeled, that is N_p in the case of the γ-memory and of the α-memory, and M_p in the case of the β-memory.

5.3 Tool's architecture

The tool is initialized by defining the fixed decoder parameters, the QC-LDPC code parameters, and reading the input file containing the building blocks' area and delay characteristics. Then, the core algorithm of the tool works in two phases:

1. Exploring the combinations of valid design choices
2. Analyzing each LDPC decoder architecture obtained

The first part defines the possible design choices, creating a database of LDPC decoder architectures to be analyzed in the second part. This database is composed by the Cartesian product of the free parameters, and associates to each parameter (e.g., scheduling) the corresponding design choice (e.g., layered scheme). Since not all combinations of design choices are possible, the invalid LDPC architectures are discarded. For example, decoders with layered scheduling must employ layered VNU architectures, or sliced message-passing schemes are incompatible with full-row CNUs. Once the database holds only valid LDPC decoders, it is analyzed in the second phase, adding both the building blocks' and the overall decoder's characteristics. The implementation of the database is straightforward as Python *dictionaries*.

The second part of the algorithm studies each LDPC decoder instance in the database, estimating its characteristics. The approach of the proposed DSE tool is to employ an algorithmic-architectural model of the decoder under analysis, considering the taken design choices in order to select the proper modeling strategy and building blocks' characteristics. In particular, this problem is unpacked and solved in the following stages:

Critical path delay and frequency Estimations on the minimum clock period and maximum operating frequency. The latter is also used in the throughput estimation.

Memory analysis Study of the number of memory units, their typology, requirements and occupied area.

Number of components, latency and processed edges Elaboration on the decoder architecture which determines the number of CNUs, VNUs and barrel shifters, as well as latency cycles and, eventually, the average processed edges. The latter is another fundamental parameter in the throughput estimation.

Loop unrolling Correction of the previous computations according to a possible loop unrolling.

Area Estimation on the overall decoder area

Throughput Estimation on the decoder throughput

The rest of this section details in depth these stages of the decoders' model analysis.

5.3.1 Critical path and operating frequency

The topology of the decoder architecture is determined above all by the scheduling scheme adopted. Starting from the employed building blocks' delay characteristic

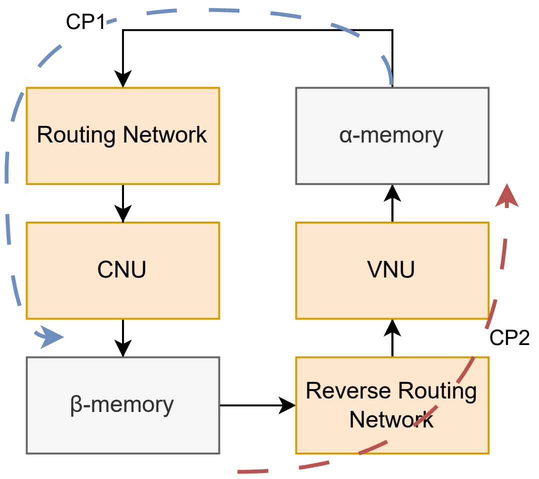


Figure 5.6: Main combinational paths in a flooding decoder.

and the topology of the architecture under analysis, it is possible to define the combinational paths and the associated overall delay. Additionally, the position of possible pipelining registers is taken into account while finding these combinational paths. Specifically, they can either be between two components, e.g. routing networks and VNUs, or in the middle of a component, e.g. a pipelined full-row CNU. For the latter, ad hoc pipelined HDL designs are synthesized, and during this stage the tool draws the characteristics of the appropriate building block's architecture. Also, single-input and half-row CNUs can split a combinational path, since they involve a register to store the temporary output values.

By evaluating the maximum among the combinational paths' delays, which corresponds to the minimum clock period, the operating frequency can be estimated as its reciprocal. An example of combinational paths in a simplified flooding scheme is shown in Figure 5.6. The multiplexer is removed since its impact is minimal. A more complex example is shown in Figure 5.7, where the two main combinational paths in a layered decoder with full-row CNUs and one pipeline stage are indicated.

5.3.2 Memory analysis

According to the decoder's architectural parameters, the DSE tool performs a memory analysis as detailed in Section 4.4, selecting the proper number of memories, their types and sizes, whether it be registers or SRAMs. FIFOs are already included in the VNU's design, thus they are not taken into account during this analysis. This stage must consider the code parameters, for example to evaluate the number of registers, included the lifting size Z.

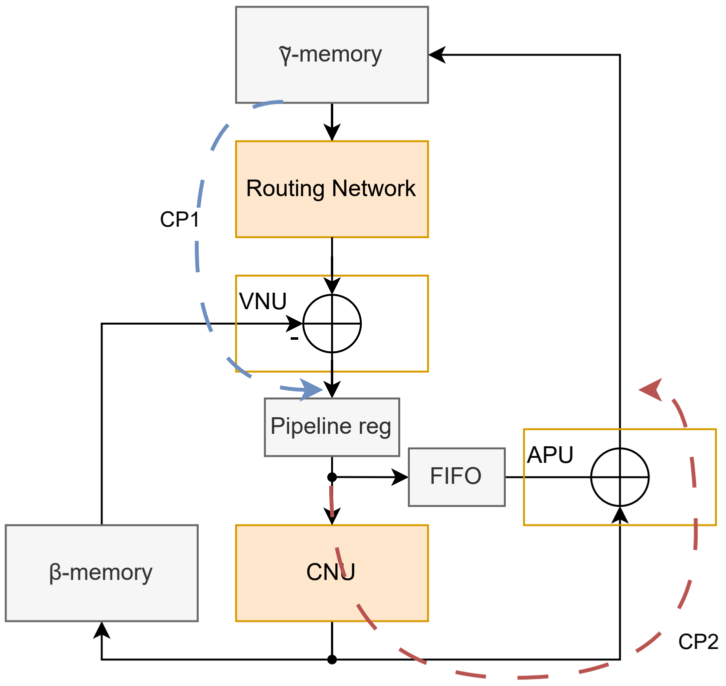


Figure 5.7: Main combinational paths in a layered pipelined decoder.

5.3.3 Topology analysis and latency

As previously described, the decoder architecture's topology strongly depends on the scheduling scheme and secondarily on the CNU architecture. This stage determines the node parallelism, as it evaluates the number of processing elements and the overall decoding latency. Latency is lastly used to estimate the average number of processed edges of the Tanner graph. This parameter is useful for the throughput estimation.

This stage selects the proper number of CNUs, VNUs and barrel shifters employed in the routing networks, according to both the scheduling scheme and the CNUs' input parallelism. In particular, flooding and sliced message-passing schemes employ Z VNUs and M CNUs. The number of barrel shifters is d_v^{max} for the reverse routing network (towards VNUs) and depends on the input parallelism of CNUs for the routing network. Layered schemes employ Z CNUs, and the number of both VNUs and barrel shifters depends on the CNUs' input parallelism.

Furthermore, the number of cycles spent to perform one decoding iteration are computed. For example, a flooding scheme decoder with M single-input CNUs and Z VNUs takes N_p cycles to complete the CNUs' phase, and N_p to complete the VNUs' operations, for a total of $2\,N_p$. On the other hand, full-row CNUs takes only one cycle, for a total of $N_p + 1$.

Latency must also take into account if the loop unrolling technique is applied. In fact, in flooding and sliced message-passing schemes, the CNU and VNU operations can be overlapped when the architecture is fully-unrolled, as each decoding iteration is computed by separated elements. In other words, a pipeline approach results from the unrolling operation, given by the presence of the α and β-memories between each stage of VNUs and CNUs. As a matter of fact, flooding and sliced message-passing schemes can be considered ALAP and ASAP approaches of the same scheduling for unrolled architectures.

It can be noted that without loop unrolling, pipelining technique solely does not increase the number of average processed edges in architectures with flooding and sliced message-passing schemes, since the iteration loop prevents the pipelining of the operations. As discussed in 4.5.3, this is solved in loop unrolled architectures as long as the iterations are fully-unrolled. Also, layered decoders can be pipelined without loop unrolling. That is due to the sub-iterations of the layers, but hazards might occur, thus reducing the throughput performance.

Finally, latency cycles can be used to evaluate the average number of processed edges of the Tanner graph. Since the latency cycles are the cycles needed to perform one decoding iteration, the average number of processed edges *avg_edges* can be computed as

$$avg\_edges = \frac{tot\_edges}{latency},$$

where *tot_edges* is the total number of edges in the Tanner graph and *latency* is the number of latency cycles. If the architecture is unrolled, the parameter *latency* can be used to represents the number of cycles needed to elaborate the same number of edges as in the Tanner graph. With this reinterpretation, it is possible to account for the improvements in performance due to the unrolling and pipelining techniques.

5.3.4 Loop unrolling correction

During this stage, the DSE tool multiplies the number of processing elements and memory instances by the unrolling coefficient, namely the maximum number of decoding iterations. This is particularly convenient in layered architectures since they only require about half decoding iterations, thus also the unrolling coefficient can be halved without losing performance. As already mentioned in Section 4.5.3, barrel shifters can be removed.

It is worth noting that the β-memories in layered architectures should be read and written by successive sub-decoders as the data flow travels through the decoding iterations in order to use the messages from previous iterations. As mentioned in Section 4.5.3, it is more efficient to use multiple decoders rather than loop unrolling.

5.3.5 Area estimation

One of the main characteristics of the analyzed LDPC decoders is the area. The approach of the presented tool estimates the overall circuit's area as the sum of its parts, counting the number of building blocks employed in the desired architecture and extrapolating the area for each of those components either from the synthesized architecture or by applying the mathematical model as explained in Section 5.2.

In particular, using the results of the topology analysis presented in Section 5.3.3 and Section 5.3.4, the number of CNUs, VNUs and barrel shifters is multiplied by the corresponding building block's area. The processing elements' characteristics are estimated using the exact syntheses results in the input file, while barrel shifters' area can be estimated with the empirical model presented in Section 5.2.1. The memory analysis referred in Section 5.3.2 returns the number and types of memories, which can be studied with the models presented in Section 5.2.2, obtaining the associated area. Adding up all these contributions, an estimation of the overall area requirement is obtained.

5.3.6 Throughput estimation

Throughput is estimated according to the formula presented in the European Horizon 2020 EPIC project [13],

$$T = \frac{avg\_edges}{tot\_edges}\,\frac{1}{I}\,N_p\;Z\;R\;f$$

where tot_edges is the number of edges in the Tanner graph, avg_edges is the average number of edges processed per clock cycle, I is the number of decoding iterations, N_p is the number of columns in the base-matrix, Z is the lifting-size, R is the code rate and f is the working frequency.

The average number of processed edges is obtained from the algorithm's stage presented in Section 5.3.3, while the frequency f is estimated with the approach proposed in Section 5.3.1. The remaining factors depend on the code structure, e.g. N_p, R, or Z; or on the number of decoding iteration I.

This approximate formula can be applied to all the decoder architectures, enabling a common analysis and comparison among them. In particular, since only partially-parallel architectures are considered, the ratio avg_edges/tot_edges can be simplified as $1/latency\_cycles$, where $latency\_cycles$ is the number of cycles necessary to complete one decoding iteration. In case of unrolled architectures, this can also be seen as the number of cycles needed to elaborate the same number of edges as tot_edges. This interpretation can easily demonstrate how unrolling not only affects the operation overlapping in various scheduling schemes, but also increment the throughput of the same factor.

5.4 Model validation

The proposed DSE tool aims at providing a fast overview on the otherwise overwhelming offering of LDPC decoder architectures. Its algorithm takes advantage of a *divide et impera* approach, exploiting the building blocks' characteristics to estimate the overall decoder's area, frequency and throughput. This section tries to prove the validity of the developed model by extracting some LDPC decoder architectures from the scientific literature, and comparing the declared characteristics against those provided by the tool in the corresponding architecture. Since the tool strongly uses the assumption of a QC-LDPC code, all the selected architectures are compliant with this family of LDPC. A normalization could be necessary.

The methodology adopted implies to:

(i) set the tool's parameters to the characteristics of the decoder and the targeted code, as close as possible to the ones used in the article;

(ii) produce the results of the design space exploration and select the most appropriate architecture, extracting its characteristics;

(iii) normalize the characteristics of the provided architectures with respect to the possible remaining parameters. A general idea is given in [25].

Some parameters that can be set in the tool are the number of rows M_p and columns N_p of the base matrix, the lifting size Z, the total number of edges in the Tanner graph, the maximum row degree d_c^{max}, the maximum column degree d_v^{max}, and the number of decoding iterations. All these parameters either have no impact on the building blocks' HDL designs, or it is limited and can be estimated with small corrections. It is worth noting that the lifting size would be a very impacting parameter, but employing the barrel shifter's and memories' models solves this issue, being the only influenced components[2].

It is possible that there are still unmatched features between the tool's modeled decoder and the one proposed in literature, which cannot simply be corrected upstream of the DSE analysis. Those differences must be corrected with a few normalization factors. They include the technology node, quantization, and others architectural differences (for example the number of routing networks).

The technology node is normalized using the scaling factor correction proposed in [26]. In the present work, of particular interest are the area and delay scaling of a given technology against the 45 nm node employed in the DSE tool's analysis. For example, to scale an architecture from 45 nm to 65 nm, the area scaling factor is 1.5. In addition, passing from 65 nm to 45 nm, the delay is multiplied by 0.712. The quantization w of LLR information influences all the building blocks designs,

[2]CNUs and VNUs only change in number of units deployed

thus all synthesis should be adapted for a correction upstream of the tool's analysis. In [25], the authors stated that a linear scaling factor of the overall area is sufficient, which is reasonable given that w increases the size of all messages and units.

Some differences cannot be corrected by normalizing the article's architecture, since it may concern only a portion of the circuit, whose percentage is not given. To be more clear, let us assume the article's decoder uses two routing networks, while the tool's proposed decoder presents only one of it. It is not possible to double the area contribution of the article's routing network, given that its single contribution is not given. For this reason, the proposed solution is to adapt the tool's prediction to the article's architecture, since it is extremely easy to distinguish the several contributions of the various building blocks in the proposed architectures. In the example, it is sufficient to know the percentage of the routing network's area on the total, and multiply the total area by a factor of $2 \cdot RN_{\%} + (1 - RN_{\%})$. This correction is compliant with the DSE tool's approach, since this is the same computation as if the tool's considered two routing networks upstream of the DSE analysis.

Given these considerations, the following comparisons normalize the article's architectures' frequency and throughput, while the area of the tool's prediction will be adjusted to the architectures' features when strictly necessary. All of these normalizations are kept as minimal as possible. A relative error is provided, where the plus (+) sign indicates an overestimation, and the minus (-) sign indicates an underestimation. It is worth noting that a worst case analysis implies an overestimation on the area characteristics, and an underestimation on the frequency and throughput. In order to isolate the frequency mismatch effects on the throughput, a normalization with respect to the tool's proposed frequency is also provided, along with the associated relative error.

5G New Radio decoder The first architecture is a decoder compliant with the 5G NR standard, presented in [27]. This decoder employs a layered scheduling scheme in a pipelined architecture. The pipeline registers are positioned after the CNUs, while the tool's pipeline registers in layered architectures are between VNUs and CNUs. Moreover, the CNUs' architecture proposed is a simplified, two-phase, full-row CNU, thus it is smaller, but requires twice the latency cycle per pipeline stage. The results are shown in Table 5.3.

Wi-MAX decoder Even though the difference between LDPC code in 5G NR and Wi-MAX is considerable, the corresponding architectures are not deeply dissimilar. The differences in the code parameters have a very limited impact in the building blocks' designs, except for the full-row CNU, whose number of input and therefore the whole tree construction depends on the maximum row-degree d_c^{max}. In particular, 5G NR implies $d_c^{max} = 19$, while the Wi-MAX $d_c^{max} = 7$. This

technology		Architecture [27] 65 nm		Tool's prediction 45 nm		
Parameters		declared	norm.	estimated	norm.	rel. error (%)
frequency	(MHz)	750	263.3	247.1	–	- 6.16 %
area	(mm^2)	1.49	–	1.15	1.87	+ 25.2 %
throughput	(Gbps)	3.04	1.42	1.32	–	- 7.00 %
Throughput normalized w.r.t. tool's frequency (Gbps):					1.34	- 0.89 %

Table 5.3: Comparison of the tool results with LDPC decoder compliant with 5G NR, presented in [27].

technology		Architecture [28] 65 nm		Tool's prediction 45 nm		
Parameters		declared	norm.	estimated	norm.	rel. error (%)
frequency	(MHz)	250	175.6	159.9	–	- 8.94 %
area	(mm^2)	0.86	–	0.78	1.27	+ 47.8 %
throughput	(Gbps)	1.2	3.37	3.07	–	- 8.94 %
Throughput normalized w.r.t. tool's frequency (Gbps):					3.07	–

Table 5.4: Comparison of the tool's results with LDPC decoder compliant with Wi-MAX, presented in [28].

correction can be estimated by counting the number of CS2, CS3 and CS4 units in the two cases and scaling accordingly. From the tool's results it is evident that the CNUs' area is very limited, thus this approximation is acceptable. The remaining parameters can be set accordingly in the tool, before the results are generated.

The decoder presented in [28] is a layered scheme, non-pipelined architecture. Similarly to the one in [27], the CNU architecture is a two-phase, simplified, full-row unit, thus the same corrections can be considered. Moreover, the APU units were removed in favour of a dual-task VNU/APU unit. The comparison results are shown in Table 5.4.

Pipelined decoder The Min-sum decoder proposed in [29] presents a pipelined architecture similar to the one studied by the DSE tool, where the pipeline registers are placed between the VNUs and the CNUs. Moreover, this architecture implements a Wi-MAX decoding assuming a matrix reordering to avoid stall cycles due to the pipeline, as discussed in Chapter 4, Section 4.4.4. The components are

technology		Architecture [29] 65 nm		Tool's prediction 45 nm		
Parameters		declared	norm.	estimated	norm.	rel. error (%)
frequency	(MHz)	161	226.1	263.2	–	+ 16.4 %
area	(mm^2)	0.77	–	0.81	0.88	+ 14.9 %
throughput	(Gbps)	1.54	4.32	5.05	–	+ 16.9 %
Throughput normalized w.r.t. tool's frequency (Gbps):					5.03	+ 0.43 %

Table 5.5: Comparison of the tool's results with pipelined LDPC decoder compliant with Wi-MAX, presented in [29].

implemented with the conventional architectures, similar to the ones used in the DSE tool. In fact, the results show a more adherent area prediction.

Bibliography

[1] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962. DOI: 10.1109/TIT.1962.1057683.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes. 1,” in *Proceedings of ICC ’93 - IEEE International Conference on Communications*, vol. 2, 1993, 1064–1070 vol.2. DOI: 10.1109/ICC.1993.397441.

[3] D. MacKay and R. Neal, “Near shannon limit performance of low density parity check codes,” *Electron. Letters*, vol. 32, no. 18, pp. 1645–1646, August 1996.

[4] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981. DOI: 10.1109/TIT.1981.1056404.

[5] J. H. Bae, A. Abotabl, H.-P. Lin, K.-B. Song, and J. Lee, “An overview of channel coding for 5G NR cellular communications,” *APSIPA Transactions on Signal and Information Processing*, vol. 8, e17, 2019. DOI: 10.1017/ATSIP.2019.10.

[6] 3GPP, “NR; Multiplexing and channel coding,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, Jul. 2020, Version 16.2.0.

[7] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, 1996. DOI: 10.1109/18.485714.

[8] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* Haboken, NJ, USA: Wiley, 2005.

[9] D. MacKay, “Good error-correcting codes based on very sparse matrices,” *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999. DOI: 10.1109/18.748992.

[10] F. Kschischang, B. 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. DOI: 10.1109/18.910572.

[11] X. Zhang, *VLSI Architecture for Modern Error-Correcting Codes*. Florida, FL, USA: CRC Press, 2017.

[12] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, “Reduced-complexity decoding of LDPC codes,” *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, 2005. DOI: 10.1109/TCOMM.2005.852852.

[13] *B5G Wireless Tb/s FEC KPI Requirement and Technology Gap Analysis*, https://epic-h2020.eu/downloads/EPIC-D1.2-B5G-Wireless-Tbs-FEC-KPI-Requirement-and-Technology-Gap-Analysis-PU-M22.pdf, 2019.

[14] F. Kschischang and B. Frey, “Iterative decoding of compound codes by probability propagation in graphical models,” *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219–230, 1998. DOI: 10.1109/49.661110.

[15] L. Liu and C.-J. R. Shi, “Sliced message passing: High throughput overlapped decoding of high-rate low-density parity-check codes,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3697–3710, 2008. DOI: 10.1109/TCSI.2008.926995.

[16] D. Hocevar, “A reduced complexity decoder architecture via layered decoding of LDPC codes,” in *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004.*, 2004, pp. 107–112. DOI: 10.1109/SIPS.2004.1363033.

[17] C.-L. Wey, M.-D. Shieh, and S.-Y. Lin, “Algorithms of finding the first two minimum values and their hardware implementation,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3430–3437, 2008. DOI: 10.1109/TCSI.2008.924892.

[18] D. E. Knuth, *The Art of Computer Programming*, 3rd edition. New York, USA: Addison-Wesley.

[19] C. Marchand, J.-B. Dore, L. Conde-Canencia, and E. Boutillon, “Conflict resolution for pipelined layered LDPC decoders,” in *2009 IEEE Workshop on Signal Processing Systems*, 2009, pp. 220–225. DOI: 10.1109/SIPS.2009.5336255.

[20] Z. Wu and K. Su, “Updating conflict solution for pipelined layered LDPC decoder,” in *2015 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, 2015, pp. 1–6. DOI: 10.1109/ICSPCC.2015.7338879.

[21] C.-Y. Lin, L.-W. Liu, Y.-C. Liao, and H.-C. Chang, “A 33.2 Gbps/iter. reconfigurable LDPC decoder fully compliant with 5G NR applications,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401329.

[22] V. L. Petrović, M. M. Marković, D. M. E. Mezeni, L. V. Saranovac, and A. Radošević, “Flexible high throughput QC-LDPC decoder with perfect pipeline conflicts resolution and efficient hardware utilization,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 5454–5467, 2020. DOI: 10.1109/TCSI.2020.3018048.

[23] S. Kim, G. E. Sobelman, and H. Lee, “A reduced-complexity architecture for ldpc layered decoding schemes,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1099–1103, 2011. DOI: 10.1109/TVLSI.2010.2043965.

[24] J. R. Hauser, “Handbook of semiconductor manufacturing technology,” in Boca Raton, FL: CRC Press, 2008, ch. MOSFET device scaling.

[25] C.-L. Chen, Y.-H. Lin, H.-C. Chang, and C.-Y. Lee, “A 2.37-Gb/s 284.8 mW rate-compatible (491,3,6) LDPC-CC decoder,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 4, pp. 817–831, 2012. DOI: 10.1109/JSSC.2012.2185193.

[26] A. Stillmaker and B. Baas, “Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm,” *Integration, the VLSI Journal*, vol. 58, pp. 74–81, 2017, http://vcl.ece.ucdavis.edu/pubs/2017.02.VLSIintegration.TechScale/.

[27] 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, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10040516. [Online]. Available: https://www.mdpi.com/2079-9292/10/4/516.

[28] T. T. Nguyen-Ly, T. Gupta, M. Pezzin, V. Savin, D. Declercq, and S. Cotofana, “Flexible, cost-efficient, high-throughput architecture for layered LDPC decoders with fully-parallel processing units,” in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016, pp. 230–237. DOI: 10.1109/DSD.2016.33.

[29] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, “Analysis and design of cost-effective, high-throughput LDPC decoders,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 508–521, 2018. DOI: 10.1109/TVLSI.2017.2776561.